
Lecture2Notes

Release 1.0.0

Hayden Housen

Aug 17, 2021

GETTING STARTED:

1	About	1
1.1	Overview	1
1.2	Components	2
1.3	FAQ	3
1.4	Significant People	4
2	Installation	5
2.1	Overview	5
2.2	Quick-Install (Copy & Paste)	5
2.3	Step-by-Step Instructions	6
3	Tutorial	9
3.1	Summarizing a lecture video	9
3.2	Summarizing a lecture video on YouTube	9
3.3	Summarizing a lecture video programmatically	10
4	Dataset General Information	11
4.1	Directory Structure	11
4.2	Walkthrough (Step-by-Step Instructions to Create Dataset)	12
4.3	Transcripts WER	12
5	Scraper Scripts	15
5.1	1. Website Scraper	15
5.2	1. YouTube Scraper	16
5.3	2. Mass Data Collector	17
5.4	2. Slides Downloader	20
5.5	2. Video Downloader	20
5.6	3. Frame Extractor	21
5.7	3. pdf2image	21
5.8	4. Auto Sort	22
5.9	4. Sort From File	22
5.10	5. Compile Data	22
6	Download Dataset	23
7	Slide Classifier	25
7.1	Overview	25
7.2	Pre-trained Models	25
7.3	Model Architecture	26
7.4	Model Training Approach	26
7.5	Results	27

7.6	Script Descriptions	30
7.7	Slide-Classifer-Pytorch Help	30
8	Slide Classifier API	35
8.1	Class Cluster Scikit	35
8.2	Custom nn.Modules	36
8.3	GradCAM	36
8.4	Learning Rate Finder	37
8.5	Mish	39
8.6	Inference	40
8.7	Slide Classifier Helpers	40
8.8	Slide Classifier PyTorch	40
9	Summarization Models	43
10	E2E General Information	45
10.1	Overall Explanation	45
10.2	Script Descriptions	45
10.3	Main Script Help	47
11	Combination and Summarization	51
11.1	Structured Joined Summarization	51
11.2	Combination	51
11.3	Modifications	52
11.4	Extractive Summarization	52
11.5	Abstractive Summarization	54
12	Perspective Cropping	55
12.1	SIFT Matcher & Perspective Cropping	55
12.2	Corner Crop Transform	56
13	Duplicate Image Removal	57
14	Figure Detection Algorithm	59
15	Slide Structure Analysis (SSA)	61
15.1	SSA Title Identification	61
16	Slide Clustering	63
16.1	Normal Algorithms	63
16.2	Segment Method	63
17	Transcribe (Speech-To-Text)	65
17.1	Chunking	65
17.2	Transcribing Methods	66
17.3	Experiment Results	68
17.4	Script Descriptions	69
17.5	DeepSegment	70
17.6	Other Transcription Options	70
18	Perspective Cropping & Corner Detection	73
18.1	Debug/Testing Mode	73
18.2	Example Images	74
18.3	Script Help	75

19 E2E API	77
19.1 Main Summarizer Class	77
19.2 Transcribe	77
19.3 Cluster	85
19.4 Corner Crop Transform	85
19.5 Text Detection	88
19.6 Figure Detection	89
19.7 Frames Extractor	91
19.8 Helpers	91
19.9 Image Hash	91
19.10 OCR	92
19.11 Segment Cluster	92
19.12 SIFT Matcher	93
19.13 Slide Classifier	96
19.14 Slide Structure Analysis	96
19.15 Spell Check	98
19.16 Summarization Approaches	98
19.17 Transcript Downloader	103
19.18 YouTube API	104
20 Indices and tables	105
Python Module Index	107
Index	109

1.1 Overview

Lecture2Notes is a project that **summarizes lectures videos**. At a high level, it parses both the **visual** and **auditory** components of the video, **extracts text** from each, **combines** them, and then **summarizes** the combined text using **automatic** summarization algorithms. These pages document the code for the entirety of “[Lecture2Notes: Summarizing Lecture Videos by Classifying Slides and Analyzing Text using Machine Learning](#).”

To get started, visit [the tutorial](#).

The project is broken into four main components: the [slide classifier](#) (including the [dataset](#)), the [summarization models](#) (neural, non-neural, extractive, and abstractive), the [end-to-end-process](#) (one command to convert to notes), and finally the [website](#) that enables users to process their own videos.

Process:

1. Extract frames from video file
2. Classify extracted frames to find frames containing slides
3. Perspective crop images containing the presenter and slide to contain only the slide by matching temporal features
4. Cluster slides to group transitions and remove duplicates
5. Run a Slide Structure Analysis (SSA) using OCR on the slide frames to obtain a formatted transcript of the text on the slides
6. Detect and extract figures from the set of unique slide frames
7. Transcribe the lecture using a speech-to-text algorithm
8. Summarize the visual and auditory transcripts
 1. Combine
 2. Run some modifications (such as only using complete sentences)
 3. Extractive summarization
 4. Abstractive summarization
9. Convert intermediate outputs to a final notes file (HTML, TXT, markdown, etc.)

The summarization steps can be toggled off and on (see [Combination and Summarization](#)).

Note: Notice an issue with this documentation? A typo? Missing or incorrect information? Please open an issue on [GitHub](#) or click “Edit on GitHub” in the top right corner of the page with issues. Documentation is almost as important

as code (what's the point of having code that can't be understood). Please report problems you find (even if its one letter). Thanks.

Abstract: Note-taking is a universal activity among students because of its benefits to the learning process. This research focuses on end-to-end generation of formatted summaries of lecture videos. Our automated multimodal approach will decrease the time required to create notes, increase quiz scores and content knowledge, and enable faster learning through enhanced previewing. The project is broken into three main components: the slide classifier, summarization models, and end-to-end-process. The system begins by extracting important keyframes using the slide classifier, a deep CNN. Then, unique slides are determined using a combination of clustering and keypoint matching. The structure of these unique slides is analyzed and converted to a formatted transcript that includes figures present on the slides. The audio is transcribed using one of several methods. We approach the process of combining and summarizing these transcripts in several ways including as keyword-based sentence extraction and temporal audio-slide-transcript association problems. For the summarization stage, we created TransformerSum, a summarization training and inference library that advances the state-of-the-art in long and resource-limited summarization, but other state-of-the-art models, such as BART or PEGASUS, can be used as well. Extractive and abstractive approaches are used in conjunction to summarize the long-form content extracted from the lectures. While the end-to-end process and each individual component yield promising results, key areas of weakness include the speech-to-text algorithm failing to identify certain words and some summarization methods producing sub-par summaries. These areas provide opportunities for further research.

1.2 Components

1. Slide Classifier

- **Overview:** The slide classifier is a computer vision machine learning model that classifies images into 9 categories as listed on [its documentation page](#).
- **Key Info:** Most importantly are the `slide` and `presenter_slide` categories which refer to slides from a screen capture and slides as recorded by a video camera, respectively. When a video camera is pointed at the slides, the frame will usually include the presenter, which is the reasoning behind the name choices. Frames from the `slide` class are processed differently than those from the `presenter_slide` class. Namely, those from `presenter_slide` are automatically perspective cropped while those from `slide` are not.
- **Dataset:** The dataset was collected using the scraper scripts in `dataset/scraper-scripts`. To learn about how to collect the dataset visit [Walkthrough \(Step-by-Step Instructions to Create Dataset\)](#). You can view information about each script in [Scraper Scripts](#).

2. Summarization Models

- **Locations:** The neural summarization models are located in `models` while the non-neural algorithms are implemented in [Combination and Summarization \(end_to_end/summarization_approaches\)](#).
- **Neural Extractive Models:** <https://github.com/HHousen/TransformerSum>
- **Neural Abstractive Models:** <https://github.com/huggingface/transformers> & <https://github.com/HHousen/DocSum>
- **More Info:** See [Summarization Models](#).

3. End-To-End Process

- **Overview:** Brings everything together to summarize lecture videos. It requires only one command to summarize a lecture video. That command can contain 20 arguments or only 1: the path to the file. See [the tutorial](#).

- **API Documentation:** *E2E API*, use if you want to modify the scripts, if you want to write new components (*pull requests welcome*), or if you want to use certain components programmatically (*guide to programmatically summarize a lecture*).
- **General Info:** *E2E General Information*, use if you want to fine-tune the parameters used for conversion.
- **Summarization Approaches:** *Combination and Summarization*, specific information about how the lecture is summarized

4. Website

- <https://lecture2notes.com>

The directory structure of the project should be relatively easy to follow. There is essentially a subfolder in the `lecture2notes` folder for each major component discussed above (documentation is in `docs/` at the root level of the repository).

Note: The slide classifier dataset is located in `dataset` and the model is located in `models/slide_classifier`. This separation was made to disconnect the data collection code from the model training code, since they are two distinct stages of the process that require little interaction (the only interaction is the copying of the final dataset).

- `dataset`: Data collection code for the slide classifier.
- `end_to_end`: Contains all the code (except `lecture2notes.models.slide_classifier.inference` and some summarization models) required to summarize a lecture video. This includes frame extraction, OCR, clustering, perspective cropping, spell checking, speech to text, and more.
- `models`: Contains the slide classifier model training code and the legacy neural summarization model repository (<https://github.com/HHousen/DocSum/>) as a git module.

1.3 FAQ

Want to add to the FAQ? Open an issue on GitHub or click “Edit on GitHub” above. All contributions are greatly appreciated. If you’re asking it, someone else probably is too.

1.3.1 Where are the summarization models?

TL;DR: <https://github.com/HHousen/TransformerSum>

The neural-based summarization models that were created as a major component of this research are not part of this repository. While initially developed as part of this repository, they were broken off due to the complexity of the code and the applicability to future projects. You can view and run the training code and use 10+ pre-trained models at <https://github.com/HHousen/TransformerSum>. Essentially, the models are more accessible to other researchers for projects unrelated to lectures if they reside in their own repository.

See *Summarization Models* for more information.

1.4 Significant People

The project was created by [Hayden Housen](#) during his sophomore, junior, and seniors years of high school as part of the Science Research program. It is actively maintained and updated by him and the community.

INSTALLATION

2.1 Overview

Installation is made easy due to conda environments. Simply run `conda env create -f environment.yml` from the root project directory and conda will create an environment called `lecture2notes` with all the required packages from `environment.yml`.

..note:: Read [the paper](#) for more in-depth explanations regarding the background, methodology, and results of this project.

2.1.1 Info About Optional Components

Certain functions in the End-To-End `transcribe.py` file require additional downloads. If you are not using the transcribe feature of the End-To-End approach then this notice can safely be ignored. These extra files may not be necessary depending on your configuration. To use the similarity function to compare two transcripts a spacy model is needed, which you can learn more about on the spacy [starter models](#) and [core models](#) documentation.

The default transcription method in the End-To-End process is to use `vosk`. You need to download a `vosk` model from the [models page](#) ([Google Drive Mirror](#)) to use this method or you can specify a different method with the `--transcription_method` flag such as `--transcription_method wav2vec`.

The End-To-End `figure_detection.py` contains a function called `detect_figures()`. This function requires the [EAST \(Efficient and Accurate Scene Text Detector\)](#) model by default due to the `do_text_check` argument defaulting to `True`. See the docstring for more information. You can download the model from [Dropbox](#) (this link was extracted from the [official code](#)) or [Google Drive](#) (my mirror). Then just extract the file by running `tar -xvzf frozen_east_text_detection.tar.gz`.

2.2 Quick-Install (Copy & Paste)

```
git clone https://github.com/HHousen/lecture2notes.git
cd lecture2notes
conda env create
conda activate lecture2notes
python -m spacy download en_core_web_sm
gdown "https://drive.google.com/uc?id=1eXwWQujo_0HVffuUx0Fa6KydjW8h4gUb" -O
↪ lecture2notes/end_to_end/model_best.ckpt
```

Extras (Linux Only):

Install extras only after the above commands have been run.

```
sudo apt install curl
sudo curl -L https://yt-dl.org/downloads/latest/youtube-dl -o /usr/local/bin/youtube-dl
sudo chmod a+rx /usr/local/bin/youtube-dl
sudo apt install ffmpeg sox wget poppler-utils
```

Commands to download a Vosk model (needed for speech-to-text) are available on the [4. Vosk](#) transcription method page.

2.3 Step-by-Step Instructions

1. Clone this repository: `git clone https://github.com/HHousen/lecture2notes.git`.
2. Change to project directory: `cd lecture2notes`.
3. Run installation command: `conda env create`.
4. Activate newly created conda environment: `conda activate lecture2notes`.
5. Run `gdown "https://drive.google.com/uc?id=1eXwWQujo_0HVffuUx0Fa6KydjW8h4gUb"` `-O lecture2notes/end_to_end/model_best.ckpt` from the project root to download the *slide classification model* and put it in the default expected location.
6. **Other Binary Packages:** Install `ffmpeg`, `sox`, `wget`, and `poppler-utils` with `sudo apt install ffmpeg sox wget poppler-utils` if on linux. Otherwise, navigate to the [sox homepage](#) to download `sox`, the [youtube-dl homepage](#) (GitHub) to download `youtube-dl`, and follow the directions in this [StackOverflow answer](#) (Windows) to install `poppler-utils` for your platform. `ffmpeg` is needed for frame extraction in Dataset and End-To-End. `sox` is needed for automatic audio conversion during the transcription phase of End-To-End.¹ `wget` is used to download videos that are not on youtube as part of the `video_downloader` scraper script in Dataset.
7. **End-To-End Process Requirements (Optional)**
 1. **Spacy:** Download the small spacy model by running `python -m spacy download en_core_web_sm` in the project root. This is required to use certain summarization and similarity features (as discussed above). A spacy model is also required when using spacy as a feature extractor in `end_to_end/summarization_approaches.py`.²
 2. **DeepSpeech/Vosk:** Download the DeepSpeech model (the `.pbmm` acoustic model and the scorer) from the [releases page](#). To reduce complexity save them to `deepspeech-models` in the project root.³ **Alternatively, it is recommended** to download the small vosk model using the commands on the [4. Vosk](#) transcription method page.
 3. **EAST:** Download the EAST model from [Dropbox](#) or by running `gdown https://drive.google.com/uc?id=1ZVn7_g58g4B0QNYNFE6MzRzpirsNTjwe`. Extract it to the End-To-End directory by running `tar -xzf frozen_east_text_detection.tar.gz -C end_to_end/`
8. **Dataset Collection Requirements (Optional) YouTube API**
 1. Run `cp .env.example .env` to create a copy of the example `.env` file.
 2. Add your YouTube API key to your `.env` file.

¹ If your audio is 16000Hz, 1 channel, and `.wav` format, then `sox` is not needed.

² The default is *not* to use spacy for feature extraction but the large model (which can be downloaded with `python -m spacy download en_core_web_lg`) is the default if spacy is manually chosen. So make sure to download the large model if you want to use spacy for feature extraction.

³ Folder name and location do not matter. Just make sure the scorer and model are in the same directory. The scripts will automatically detect each when given the path to the folder containing them.

3. You can now use the scraper scripts to scrape YouTube and create the dataset needed to train the slide classifier.
9. **Transcript Download w/YouTube API (Not Recommended)** If you want to download video transcripts with the YouTube API⁴, place your `client_secret.json` in the `dataset/scraper-scripts` folder (if you want to download transcripts with the `scraper-scripts`) or in `End-To-End` (if you want to download transcripts in the entire end-to-end process that converts a lecture video to notes).

⁴ The default is to use `youtube-dl` which needs no API key.

TUTORIAL

After you've installed `lecture2notes` using the instructions in [Installation](#), you can follow this guide to perform some common actions.

..note:: Read [the paper](#) for more in-depth explanations regarding the background, methodology, and results of this project.

3.1 Summarizing a lecture video

```
cd End-To-End
python main.py --auto_id --remove_duplicates --deepspeech_model_dir ../deepspeech-models/
↳ <path to video>
```

- `--auto_id` changes the location where files will be saved during processing to a folder in the present working directory named the first 12 characters of the input video's SHA1 hash.
- `--remove_duplicates` will remove duplicate slides before clustering. Once frames have been classified and the slides have been identified, there are likely to be duplicates. Setting this option removes very closely matching slides. Similar slides are detected using perceptual hashing by default (other hashing methods are available).
- `--deepspeech_model_dir` is the path to the folder containing the DeepSpeech model files (the `.pbmm` acoustic model and the scorer).
- `<path to video>` is the path to your video file that you want to process.

3.2 Summarizing a lecture video on YouTube

1. First, download the video from YouTube: `youtube-dl <video url>` (where `<video url>` looks like `https://www.youtube.com/watch?v=dQw4w9WgXcQ`)
2. Then, process the file using a command similar to the one in [Summarizing a lecture video](#):

```
cd End-To-End
python main.py --auto_id --remove_duplicates --deepspeech_model_dir ../deepspeech-models/
↳ --transcription_method youtube --video_id <video id from youtube> <path to video>
```

The only differences from [Summarizing a lecture video](#) are the addition of `--transcription_method youtube` and `--video_id <video id from youtube>`.

- `--transcription_method youtube` makes the script attempt to download the transcript of the video directly from YouTube. This only works when the video has manually attached captions on YouTube. Videos with

automatically generated captions from YouTube will still be processed locally using DeepSpeech by default. This is why the `--deepspeech_model_dir` argument is still specified.

- `--video_id <video id from youtube>` tells the script which video from youtube to download the captions from. `<video id from youtube>` should be set to the id of the video. The id is the part after the `?v=` in the url up to (and not including) the `&` or if no `&` then until the end of the url (if the video link is `https://www.youtube.com/watch?v=dQw4w9WgXcQ` then the id is `dQw4w9WgXcQ`).

3.3 Summarizing a lecture video programmatically

Summarizing a lecture video programmatically is fairly easy due to the `LectureSummarizer` class.

First, create a dictionary or `argparse.Namespace` of the settings you want to use for your `LectureSummarizer` object. The default parameters are stored in a json file at `end_to_end/default_params.json`. This file directly mirrors the defaults set for each `argparse` argument in `end_to_end/main.py`, the script used for CLI usage.

However, the default configuration leaves `video_path` set to null so we need to override this. Instead of modifying or creating a new configuration object for each video, you can pass overrides to the `LectureSummarizer` object upon creation (see below).

Next, create your `LectureSummarizer` object, call the `run_all()` function, and get the summary like so:

```
from lecture2notes.end_to_end.summarizer_class import LectureSummarizer
default_config_path = "lecture2notes/end_to_end/default_params.json"
video_path = "path/to/my/amazing/lecture/video.mp4"
summarizer = LectureSummarizer(default_config_path, video_path=video_path)

summarizer.run_all()

structured_summary = summarizer.final_data["structured_summary"]
lecture_summary = summarizer.final_data["lecture_summary"]
transcript = summarizer.final_data["transcript"]
```

Alternatively, you can iterate over the `all_step_functions` attribute of your `LectureSummarizer` object to run your own code between each step of the process. For example, you can store the current step in a database or to the file system so that if you restart your program the `LectureSummarizer` can automatically resume:

```
last_step_run = int(open("last_step_run.txt", "r").read())

with open("last_step_run.txt", "w") as file:
    for idx, step_func in enumerate(summarizer.all_step_functions):
        if idx + 1 < last_step_run:
            # Skip steps that have already been ran
            continue

        last_step_run = idx + 1
        file.write(last_step_run)

        step_func()
```


DATASET GENERAL INFORMATION

4.1 Directory Structure

- **classifier-data:** Created by [5. Compile Data](#). Contains all extracted slides and extracted sorted frames from the videos directory. This is the folder that should be given to the model for training.
- **scraper-scripts:** Contains all of the scripts needed to obtain and manipulate the data. See [Scraper Scripts](#) for more information.
- **slides:**
 - *images:* The location where slide images extracted from slideshows in *pdfs* subdirectory are saved (used by [3. pdf2image](#)).
 - *pdfs:* The location where downloaded slideshow PDFs are saved (used by [2. Slides Downloader](#)).
- **videos:** Contains the following directory structure for each downloaded video:
 - **video_id:** The parent folder containing all the files related to the specific video.
 - * *frames:* All frames extracted from *video_id* by [3. Frame Extractor](#).
 - * *frames_sorted:* Frames from *video_id* that are grouped into correct classes. [4. Auto Sort](#) can help with this but you must verify correctness. More at [4. Auto Sort](#).
- **slides-dataset.csv:** A list of all the slide presentations used in the dataset. **NOT** automatically updated by [2. Slides Downloader](#). You must manually update this file if you want the dataset to be reproducible.
- **sort_file_map.csv:** A list of filenames and categories. Used exclusively by [4. Sort From File](#) to either make a file mapping of the category to which each frame belongs or to sort each file in *sort_file_map.csv*, moving the respective frame from *video_id/frames* to *video_id/frames_sorted/category*.
- **to-be-sorted.csv:** A list of videos and specific frames that have been sorted by [4. Auto Sort](#) but need to be checked by a human for correctness. When running [4. Auto Sort](#) any frames where the AI model's confidence level is below a threshold are added to this list as most likely incorrect.
- **videos-dataset.csv:** A list of all videos used in the dataset. Automatically updated by [1. YouTube Scraper](#) and [1. Website Scraper](#). The *provider* column is used by [2. Video Downloader](#) to determine how to download the video.

4.2 Walkthrough (Step-by-Step Instructions to Create Dataset)

1. Install Prerequisite Software: `youtube-dl, wget, ffmpeg, poppler-utils` (see [Quick-Install \(Copy & Paste\)](#))
2. **Download Content:**
 1. Download all videos: `python 2-video_downloader.py csv`
 2. Download all slides: `python 2-slides_downloader.py csv`
3. **Data Pre-processing:**
 1. Convert slide PDFs to PNGs: `python 3-pdf2image.py`
 2. Extract frames from all videos: `python 3-frame_extractor.py auto`
 3. Sort the frames: `python 4-sort_from_file.py sort`
4. Compile and merge the data: `python 5-compile_data.py`

4.3 Transcripts WER

Script location: `dataset/transcripts_wer.py`

This script will calculate the Word Error Rate (WER), Match Error Rate (MER), and Word Information Lost (WIL) for all videos in `dataset/videos-dataset.csv` that are YouTube videos with manual transcripts added (see the [YouTube transcription method](#) for more info about transcripts on YouTube).

There are two modes:

1. **transcribe:** Runs speech-to-text with DeepSpeech.
Process: For each transcript in `dataset/transcripts`:
 1. Download the audio for the video
 2. Convert the audio to WAV
 3. Run DeepSpeech speech-to-text
2. **calc:** Calculate the statistics between the YouTube (human, ground-truth) and DeepSpeech (AI, ML transcripts).
Process: For each processed transcript (those with `--suffix`) in `dataset/transcripts`:
 1. Convert the YouTube captions file to a string
 2. Apply pre-processing to the transcripts (to lower case, remove multiple spaces, strip, sentences to list of words, remove empty strings)
 3. Compute the statistics using the [jiwer](#) package
 4. Log the stats
 5. When all files are complete then log the average stats

Note: This script does not automatically download the transcripts for the YouTube videos. It just transcribes the YouTube videos in `dataset/videos-dataset.csv` with DeepSpeech and computes statistics with ground-truth transcripts. This means your ground-truth transcripts can come from a source other than YouTube and this script will still work. To download the transcripts for the videos in `dataset/videos-dataset.csv` use [2. Video Downloader](#).

4.3.1 Directions

Step 0: Make sure how have some videos in dataset/videos-dataset.csv. The *1. YouTube Scraper* script can be used to add videos to the dataset.

1. Run `python 2-video_downloader.py csv --transcript` to download transcripts (in “.vtt” format) for all the YouTube videos in dataset/videos-dataset.csv to the dataset/transcripts folder.
2. Run `python transcripts_wer.py transcribe` to transcribe all the videos with ground-truth transcripts using DeepSpeech.
3. Run `python transcripts_wer.py calc` to calculate the statistics (including WER) between the DeepSpeech and YouTube transcripts.

4.3.2 Transcripts WER Script Help

```
usage: transcripts_wer.py [-h] [--transcripts_dir TRANSCRIPTS_DIR]
                        [--deepspeech_dir DEEPSPEECH_DIR] [--suffix SUFFIX]
                        [--no_chunk]
                        [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                        {transcribe,calc_wer}

Word Error Rate (WER) for Transcripts with DeepSpeech

positional arguments:
{transcribe,calc_wer}
    `transcribe` each video and create a transcript using
    ML models or use `calc_wer` to compute the WER for the
    created transcripts

optional arguments:
-h, --help            show this help message and exit
--transcripts_dir TRANSCRIPTS_DIR
                        path to the directory containing transcripts
                        downloaded with 2-video_downloader.py
--deepspeech_dir DEEPSPEECH_DIR
                        path to the directory containing the DeepSpeech models
--suffix SUFFIX        string added after the video id and before the
                        extension in the transcript output from the ML model
--no_chunk             Disable audio chunking by voice activity.
-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --log {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Set the logging level (default: 'Info').
```


SCRAPER SCRIPTS

All scripts that are needed to obtain and manipulate the data. Located in `dataset/scrapper-scripts`.

Note: The number before the name of each script corresponds to the order the scripts are normally used in. Some scripts may have the same number because they do different tasks that take the same spot in the data processing process. For instance, there may be one script to work with slide presentations (PDFs) and another to work with videos that occupy the same position (for instance *2. Slides Downloader* and *2. Video Downloader*)

5.1 1. Website Scraper

Takes a video page link, video download link, and video published date and then adds that information to `dataset/videos-dataset.csv`.

- Command:

```
python 1-website_scraper.py <date> <page_link> <video_download_link>
↳ <description (optional)>
```

- `<date>` is the date the lecture was published
- `<page_link>` is the link to the webpage where the video can be found
- `<video_download_link>` is the direct link to the video
- `<description (optional)>` is an optional description that gets saved with the rest of the information (currently not used internally)

- Example:

```
python 1-website_scraper.py 1-1-2010 \
https://oyc.yale.edu/astronomy/astr-160/update-1 \
http://openmedia.yale.edu/cgi-bin/open_yale/media_downloader.cgi?file=/
↳ courses/spring07/astr160/mov/astr160_update01_070212.mov
```

5.2 1. YouTube Scraper

Takes a video id or channel id from YouTube, extracts important information using the YouTube Data API, and then adds that information to dataset/videos-dataset.csv.

- Output of `python 1-youtube_scraper.py --help`:

```
usage: 1-youtube_scraper.py [-h] [-n N] [-t] [--transcript-use-yt-api] [-l
↪N]
                                [-f PATH] [-o SEARCH_ORDER] [-p PARAMS]
                                {video,channel,transcript} STR

YouTube Scraper

positional arguments:
{video,channel,transcript}
↪video                        Get metadata for a video or a certain number of
                                videos
                                from a channel. Transcript mode downloads the
                                transcript for a video_id.
STR                           Channel or video id depending on mode

optional arguments:
-h, --help                    show this help message and exit
-n N, --num_pages N          Number of pages of videos to scape if mode is
                                `channel`. 50 videos per page.
-t, --transcript              Download transcript for each video scraped.
--transcript-use-yt-api      Use the YouTube API instead of youtube-dl to
↪download
                                transcripts. `--transcript` must be specified for
↪this
                                option to take effect.
-l N, --min_length_check N    Minimum video length in minutes to be scraped. Only
                                works when `mode` is "channel"
-f PATH, --file PATH         File to add scraped results to.
-o SEARCH_ORDER, --search_order SEARCH_ORDER
                                The order to list videos from a channel when `mode`
↪is
                                'channel'. Acceptable values are in the YouTube API
                                Documentation: https://developers.google.com/
↪youtube/v
                                3/docs/search/list
-p PARAMS, --params PARAMS    A string dictionary of parameters to pass to the
↪call
                                to the YouTube API. If mode=video then the
                                `videos.list` api is used. If mode=channel then the
                                `search.list` api is used.
```

- Examples
 - Add a single lecture video to the dataset:

```
python 1-youtube_scraper.py video 63hAHbkzJG4
```

- Get the transcript for a video file:

```
python 1-youtube_scraper.py transcript 63hAHbkzJG4
```

- Add a video to the dataset/videos-dataset.csv and get the transcript:

```
python 1-youtube_scraper.py video 63hAHbkzJG4 --transcript
```

- Scrape the 50 latest videos from a channel:

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTUrIALZ0w --num_pages ↵  
↵1
```

- Scrape the 50 most viewed videos from a channel:

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTUrIALZ0w --num_pages ↵  
↵1 --search_order viewCount
```

- Scrape the 50 latest videos from a channel that were published before 2020:

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTUrIALZ0w --num_pages ↵  
↵1 --params '{"publishedBefore": "2020-01-01T00:00:00Z"}'
```

- Scrape the 100 latest videos from a channel longer than 20 minutes:

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTUrIALZ0w --num_pages ↵  
↵2 --min_length_check 20
```

- Mass Download 1 (to be used with 2. *Mass Data Collector*):

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTUrIALZ0w --num_pages ↵  
↵2 --min_length_check 20 -f ../mass-download-list.csv
```

- Mass Download 2 (specify certain dates and times):

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTUrIALZ0w --num_pages ↵  
↵2 --min_length_check 20 -f ../mass-download-list.csv --params '{  
↵"publishedBefore": "2015-01-01T00:00:00Z", "publishedAfter": "2014-01-  
↵01T00:00:00Z"}'
```

5.3 2. Mass Data Collector

This script provides a method to collect massive amounts of new data for the slide classifier. These new lecture videos are selected based on what the model struggles with (where its certainty is lowest). This means the collected videos train the model the fastest while exposing it to the most unique situations. However, this method will ignore videos that the model is very confident with but is actually incorrect. These videos are the most beneficial but must be manually found.

The *Mass Data Collector* does the following for each video in dataset/mass-download-list.csv:

1. Download the video to dataset/mass-download-temp/[video_id]

2. Extracts frames
3. Classifies the frames to obtain certainties and the percent incorrect (where certainty is below a threshold)
4. Adds `video_id`, `average_certainty`, `num_incorrect`, `percent_incorrect`, and `certainties` to `dataset/mass-download-results.csv`
5. Deletes video folder (`dataset/mass-download-temp/[video_id]`)

The `--top-k` (or `-k`) argument can be specified to the script add the top `k` most uncertain videos to the `dataset/videos-dataset.csv`. This must be ran after the `dataset/mass-download-results.csv` file has been populated.

Warning: This script will use a lot of bandwidth/data. For instance, the below commands will download 100 videos from YouTube. If each video is 100MB (which is likely on the low end) then this will download at least 10GB of data.

Examples:

1. **Low Disk Space Usage, High Bandwidth, Duplicate Calculations, Large Dataset Filesize Recommended** if you want to build the dataset at full 1080p resolution so that it can be used with a plethora of model architectures. This was how the official dataset was compiled.

The below commands do the following:

1. Scrape the [MIT OpenCourseWare](#) YouTube channel for the latest 100 videos that are longer than 20 minutes and save the data to `../mass-download-list.csv`
 - Optionally, only find videos in a date range. To do this you need to specify the `--params` argument like so: `--params '{"publishedBefore": "2014-07-01T00:00:00Z", "publishedAfter": "2014-01-01T00:00:00Z"}'`. The full list of available parameters can be found in the [YouTube API Documentation for search.list](#) if mode is `channel` and [YouTube API Documentation for videos.list](#) if mode is `video`.
2. Run the *Mass Data Collector* to download each video at 480p and determine how certain the model is with its predictions on that video.
3. Take the top 20 most uncertain videos and add them to the `dataset/videos-dataset.csv`.
4. Download the newly added 20 videos at 480p
5. Extract frames from the new videos
6. Sort the frames from top 20 most uncertain videos
7. Now it is time for you to check the model's predictions, fix them, and then train a better model on the new data.

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTURIALZ0w --num_pages 2 --
min_length_check 20 -f ../mass-download-list.csv
python 2-mass_data_collector.py --resolution 480
python 2-mass_data_collector.py -k 20
python 2-video_downloader.py csv
python 3-frame_extractor.py auto
python 4-auto_sort.py
```

2. **High Disk Space Usage, Higher Bandwidth, No Duplicate Calculations, Large Dataset Filesize Recommended** if you want to build the dataset at full 1080p resolution but do not want to “waste” compute resources on duplicate calculations.

Specifying the `--no_remove` argument to `2-mass_data_collector.py` will make the script keep the processed videos instead of removing them. This means the videos can be copied to the `dataset/videos` folder, manually inspected and fixed, and then 5. *Compile Data* can be used to copy them to the `dataset/classifier-data` folder.

It is recommended to not set the `--resolution` if using this method because some of the downloaded videos will eventually be added to the dataset. **The dataset is compiled at maximum resolution so that different models can be used that accept different resolutions.**

3. **Lower Disk Space Usage, Low Bandwidth, Duplicate Calculations, Small Dataset Filesize Recommended** if you want to build the dataset for a specific model architecture and if you want the dataset to take up a relatively small amount of disk space.

If you want to train a `resnet34`, for example, which expects 224x224 input images, then you can set the resolution to 240p when downloading videos since the frames will be scaled before being used for training anyway. However, if you ever want to train a model that expects larger input images, you will have to download and reprocess the entire dataset.

The modified commands look like this:

```
python 1-youtube_scraper.py channel UCEBb1b_L6zDS3xTUrIALZ0w --num_pages 2 --
    min_length_check 20 -f ../mass-download-list.csv
python 2-mass_data_collector.py --resolution 240
python 2-mass_data_collector.py -k 20
python 2-video_downloader.py csv --resolution 240
python 3-frame_extractor.py auto
python 4-auto_sort.py
```

Notice that the resolution was changed to 240 for the second command and the resolution option was added to the fourth command.

This option can be modified as described in the second method by adding the `--no_remove` argument to `2-mass_data_collector.py`. This will increase disk usage but will prevent duplicate calculations and decrease overall bandwidth since videos will not have to be redownloaded.

5.3.1 Mass Dataset Collector Script Help

Output of `python 2-mass_data_collector.py --help`:

```
usage: 2-mass_data_collector.py [-h] [-k K] [-nr] [-r RESOLUTION] [-p]

Mass Data Collector

optional arguments:
  -h, --help            show this help message and exit
  -k K, --top_k K        Add the top `k` most uncertain videos to the videos-
                        dataset.
  -nr, --no_remove      Don't remove the videos after they have been
                        processed. This makes it faster to manually look
                        through the most uncertain videos since they don't
                        have to be redownloaded, but it will use more disk
                        space.
  -r RESOLUTION, --resolution RESOLUTION
                        The resolution of the videos to download. Default is
                        maximum resolution.
```

(continues on next page)

(continued from previous page)

<code>-p, --pause</code>	Pause after each video has been processed but before deletion.
--------------------------	--

5.4 2. Slides Downloader

Takes a link to a pdf slideshow and downloads it to `dataset/slides/pdfs` or downloads every entry in `dataset/slides-dataset.csv` (csv option).

- Command: `python slides_downloader.py <csv/your_url>`
- Examples:
 - If csv: `python 2-slides_downloader.py csv`
 - If your_url: `python 2-slides_downloader.py https://bit.ly/3dYtUPM`
- Required Software: `wget`

5.5 2. Video Downloader

Uses `youtube-dl` (for youtube videos) and `wget` (for website videos) to download either a youtube video by id or every video that has not been download in `dataset/videos-dataset.csv`.

This script can also download the transcripts from YouTube using `youtube-dl` for each video in `dataset/videos-dataset.csv` with the `--transcript` argument..

- Command: `python 2-video_downloader.py <csv/youtube --video_id your_youtube_video_id>`
- Examples:
 - If csv: `python 2-video_downloader.py csv`
 - If your_youtube_video_id: `python 2-video_downloader.py youtube --video_id 1Qws70XGSq4`
 - Download all transcripts: `python 2-video_downloader.py csv --transcript` (will not download videos or change `dataset/videos-dataset.csv`)
- Required Software: `youtube-dl` ([YT-DL Website](#)/[YT-DL Github](#)), `wget`

5.5.1 Video Downloader Script Help

Output of `python 2-video_downloader.py --help`:

```
usage: 2-video_downloader.py [-h] [--video_id VIDEO_ID] [--transcript]
                             [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                             {csv,youtube}

Video Downloader

positional arguments:
{csv,youtube}          `csv`: Download all videos that have not been marked
                        as downloaded from the `videos-dataset.csv`.
                        `youtube`: Download all youtube videos that have not been marked
                        as downloaded from the `videos-dataset.csv`.
```

(continues on next page)

(continued from previous page)

```

        `youtube`: download the specified video from YouTube
        with id ``--video_id`.

optional arguments:
-h, --help            show this help message and exit
--video_id VIDEO_ID  The YouTube video id to download if `method` is
                    `youtube`.
--transcript          Download the transcript INSTEAD of the video for each
                    entry in `videos-dataset.csv`. This ignores the
                    `downloaded` column in the CSV and will not download
                    videos.
-r RESOLUTION, --resolution RESOLUTION
                    The resolution of the videos to download. Default is
                    maximum resolution.
-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --log {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                    Set the logging level (default: 'Info').

```

5.6 3. Frame Extractor

Extracts either every N frames from a video file (selected by id and must be in *videos* folder) or, in auto mode, every N frames from every video in the dataset that has been downloaded and has not had its frames extracted already. `extract_every_x_seconds` can be set to auto to use the `get_extract_every_x_seconds()` function to automatically determine a good number of frames to extract. auto mode uses this feature and allows for exact reconstruction of the dataset. Extracted frames are saved into `dataset/videos/[video_id]/frames`.

- Command: `python 3-frame_extractor.py <video_id/auto> <extract_every_x_seconds/auto> <quality>`
- Examples:
 - If *video_id*: `python 3-frame_extractor.py VT2o4KCEbes 20 5` or to automatically extract a good number of frames: `python 3-frame_extractor.py 63hAHbkzJG4 auto 5`
 - If *auto*: `python 3-frame_extractor.py auto`
- Required Software: `ffmpeg` ([FFmpeg Website](#)/[FFmpeg Github](#))

5.7 3. pdf2image

Takes every page in all pdf files in `dataset/slides/pdfs`, converts them to png images, and saves them in `dataset/slides/images/pdf_file_name`.

- Command: `python 3-pdf2image.py`
- Required Software: `poppler-utils` (`pdftoppm`) ([Man Page](#)/[Website](#))

5.8 4. Auto Sort

Goes through every extracted frame for all videos in the dataset that don't have sorted frames (based on the presence of the `sorted_frames` directory) and classifies them using `models/slide_classifier`. You need either a trained pytorch model to use this. Creates a list of frames that need to be checked for correctness by humans in `dataset/to-be-sorted.csv`. This script imports certain files from `models/slide_classifier` so the directory structure must not have been changed from installation.

- Command: `python 4-auto_sort.py`

5.9 4. Sort From File

Creates a CSV of the category assigned to each frame of each video in the dataset or organizes extracted frames from a previously created CSV. The purpose of this script is to exactly reconstruct the dataset without downloading the already sorted images.

There are three options: 1. `make`: make a file mapping of the category to which each frame belongs by reading data from the `dataset/videos` directory. 2. `make_compiled` performs the same task as `make` but reads from the `dataset/classifier-data` directory. This is useful if the dataset has been compiled and the `dataset/videos` folder has been cleared. 3. `sort`: sort each file in `dataset/sort_file_map.csv`, moving the respective frame from `video_id/frames` to `video_id/frames_sorted/category`.

Note: This script appends to `dataset/sort_file_map.csv`. It will not overwrite data.

- Command: `python 4-sort_from_file.py <make/make_compiled/sort>`

5.10 5. Compile Data

Merges the sorted frames from all the videos and slides in the dataset to `dataset/classifier-data`.

Note: This script will not erase any data already stored in the `dataset/classifier-data` dataset folder.

- Command: `python 5-compile_data.py <all/videos/slides>`
- **Examples:**
 - If *videos*: `python 5-compile_data.py videos`, processes only sorted frames from videos
 - If *slides*: `python 5-compile_data.py slides`, processes images from slides
 - If *all*: `python 5-compile_data.py all`, processes from both videos and slides

DOWNLOAD DATASET

Note: For more information about the dataset see *Dataset General Information* and for more information about the models trained using this data see *Pre-trained Models*.

Name	Description	Download
classifier-data-cv	Standard dataset split into 3 sets for cross validation	Google Drive Link
classifier-data-three-train-test	3 class dataset split into train and test sets	Google Drive Link
classifier-data-three	3 class dataset (slide, presenter_side, other)	Google Drive Link
classifier-data-train-test	Standard dataset split into train and test sets	Google Drive Link
classifier-data	Standard dataset grouped by the 9 classes	Google Drive Link

Additionally, all of the above versions of the dataset are available from [this Google Drive folder](#)

SLIDE CLASSIFIER

7.1 Overview

The slide classification model is used in the end-to-end process to classify frames of an input video into 9 classes:

- audience
- audience_presenter
- audience_presenter_slide
- demo
- presenter
- presenter_slide
- presenter_whiteboard
- slide
- whiteboard

This allows the system to identify images containing slides for additional processing and means it can ignore useless frames that do not belong in a summary, such as those containing only the presenter or audience. We conduct preliminary tests with 7 architectures, but only report results from [ResNet](#) and [EfficientNet](#) models due to their superior accuracies.

Important: Interactive charts, graphs, raw data, run commands, hyperparameter choices, and more for all experiments are publicly available on the [Lecture2Notes-Slide_Classifier Weights & Biases](#) page.

7.2 Pre-trained Models

Quick Start: Use the `three-category` model trained on the `train-test-three` dataset. It achieves the highest accuracy (about 90%). Run `gdown "https://drive.google.com/uc?id=1eXwWQujo_0HVffuUx0Fa6KydjW8h4gUb" -O lecture2notes/end_to_end/model_best.ckpt` from the project root to download the model and put it in the default expected location.

Model	Dataset	Accuracy	Model Download	Other Checkpoints
Final-general	<code>train-test</code>	82.62	Google Drive Link	Google Drive Folder
Three-category	<code>train-test-three</code>	89.97	Google Drive Link	Google Drive Folder
Squished-image	<code>train-test</code>	83.86	Google Drive Link	Google Drive Folder
	<code>train-test-three</code>	87.21	Google Drive Link	Google Drive Folder

The median models (**recommended**) listed above can also be found on the [Lecture2Notes-Slide_Classifier Weights & Biases](#) page or from [this Google Drive folder](#).

Every model that was trained for the experiments is available from the [Lecture2Notes-Slide_Classifier Weights & Biases](#) page or from [this Google Drive mirror](#).

7.3 Model Architecture

After testing several architectures, we chose ResNet-34 as our final model architecture due to its speed, accuracy, and size. We started training all models from ImageNet pre-trained checkpoints and only perform gradient updates on the last chunk of layers (the pooling and fully connected linear layers shown below). We modified the architectures by changing the last chunk to enable better fine-tuning:

1. **ResNet:** `AdaptiveConcatPool2d(1), Flatten(), BatchNorm1d(1024), Dropout(0.25), Linear(1024, 512), ReLU(inplace=True), BatchNorm1d(512), Dropout(0.5), Linear(512, num_classes)`
2. **EfficientNet:** `AdaptiveConcatPool2d(1), Flatten(), Linear(num_features * 2, 512), MemoryEfficientSwish(), BatchNorm1d(512), Dropout(0.5), Linear(512, num_classes)`

7.4 Model Training Approach

We performed 3-fold cross validation (CV) in order to optimize the slide classifier's hyperparameters. We used the Tree-structured Parzen Estimator algorithm provided by [Optuna](#). All important hyperparameters were optimized, including model selection between a ResNet-34 and an EfficientNet-b0.

We implemented CV by splitting the [dataset](#) into 3 roughly equal sections containing 5502, 5040, and 5057 frames respectively. The hyperparameter optimization algorithm optimizes the average accuracy across the validation sets for each CV split. The accuracy on the validation set is used instead of the training set to make sure the model does not memorize specific characteristics of the images that it is trained on. Accuracy is optimized as opposed to f1-score because f1-score takes into account class imbalance. For this model, the `slide` and `presenter_slide` classes are the two largest and also are the only classes used in the rest of the lecture summarization pipeline. If the model fails to correctly classify frames into the other categories there will be no impact on the final summary. These additional categories were included to allow for future research without having to relabel the data. For instance, the `whiteboard` class can be used in the future to extract handwritten text and drawing from a whiteboard to be added to notes.

Once the best hyperparameters were determined, we split the original dataset into training and testing sets (the `train-test` dataset) and retrained the model with the best hyperparameters. We did not simply copy the model checkpoint that reached the highest accuracy during the hyperparameter optimization process because splitting the dataset again allows us to train the model on 80% of the data instead of 67%. To create the testing set we selected a subset of videos that account for about 20% (by the number of videos not frames) of the complete dataset. This subset minimizes the deviation of the testing set percentage (items in the testing set divided by total items in category) from 20% for each category. We tested 10 million combinations of videos and obtained an average deviation of 0.0929, which equates to the following testing set percentages: `slide`=0.186, `whiteboard`=0.101, `audience_presenter`=0.201, `presenter_whiteboard`=0.205, `presenter_slide`=0.188, `audience_presenter_slide`=0.194, `demo`=0.294, `presenter`=0.208, `audience`=0.304. Selecting a random subset was not used because it could have resulted in class imbalance between the two datasets, thus offsetting the results. Therefore, there are 16 videos (3088 frames) and 62 videos (12511 frames) in the testing and training sets respectively.

Since only the `slide` and `presenter_slide` classes are used in the end-to-end process, we trained a separate model with those classes and an `other` class containing the remaining frames. We created a `train-test-three` dataset with these three categories using the same splitting approach that was used to create the `train-test` dataset. The average deviation was 0.001104 after 156 million combinations, which equates to the following testing set percentages: `slide`=0.200, `presenter_slide`=0.200, `other`=0.197.

Since pre-trained ImageNet CNNs accept square images, but video frames typically have an aspect ratio of 16:9, we center crop the data to a square and then scale to the correct dimensions for the `train-test` and `train-test-three` datasets. However, this may remove important data. Thus, we train a model on both datasets using squished images created by simply rescaling the image to the correct dimensions. Models trained using squished images will learn squished features and thus will produce variable results on images with a proper aspect ratio.

7.5 Results

Table 1: Performance of the 4 model configurations on the testing set.

Model	Dataset	Accuracy	Accuracy (train)	F-score	Precision	Recall
Final-general	train-test	82.62	98.58	87.44	97.73	82.62
Three-category	train-test-three	89.97	99.72	93.82	99.95	89.97
Squished-image	train-test	83.86	97.16	88.16	97.72	83.86
	train-test-three	87.21	100.00	91.57	99.80	87.21

After training 262 models for a total of 94 trials during CV, the highest average accuracy of 85.42% was achieved by a ResNet-34 model after training for 9 epochs with the following hyperparameters: `batch_size=64`, `learning_rate=0.00478`, `momentum=0.952`, `weight_decay=0.00385`, `adamw_alpha=0.994`, `adamw_eps=4.53e-07`, `scheduler=onecycle`.

Table 2: Classification report for median (by accuracy) non-squished **final-general** model

Class Name	Precision	Recall	F1-Score	Support
audience	0.00	0.00	0.00	14
audience_presenter	0.52	0.21	0.30	57
audience_presenter_slide	0.46	0.32	0.38	34
demo	0.15	0.07	0.10	126
presenter	0.91	0.94	0.92	976
presenter_slide	0.78	0.86	0.82	934
presenter_whiteboard	0.89	0.90	0.89	372
slide	0.86	0.85	0.86	557
whiteboard	0.62	0.44	0.52	18
accuracy	–	–	0.83	3088
macro avg	0.58	0.51	0.53	3088
weighted avg	0.81	0.83	0.82	3088

Table 3: Classification report for median (by accuracy) non-squished **three-category** model

Class Name	Precision	Recall	F1-Score	Support
other	0.91	0.98	0.94	1504
presenter_slide	0.93	0.80	0.86	992
slide	0.86	0.90	0.88	600
accuracy	–	–	0.91	3096
macro avg	0.90	0.89	0.89	3096
weighted avg	0.91	0.91	0.90	3096

For each of the 4 model configurations, we trained 11 models and report the average metrics in the tables and figures on this page.

The final-general model (trained on the `train-test` dataset with the best hyperparameters found) achieved an average accuracy of 82.62%. About 15% of the `slide` frames were incorrectly classified as `presenter_slide`. About 14% of the `presenter_slide` (of which 50% were `slide` and 43% were `presenter`) frames were classified incorrectly. Incorrectly classifying `slide` frames as `presenter_slide` will have minimal impact on the final summary. Incorrectly classifying `presenter_slide` frames as `slide` will impact the final summary because they will not receive the correct processing. Incorrectly classifying `presenter_slide` as `presenter` represents a possible loss of information, but this is unlikely due to the same slide appearing in multiple frames.

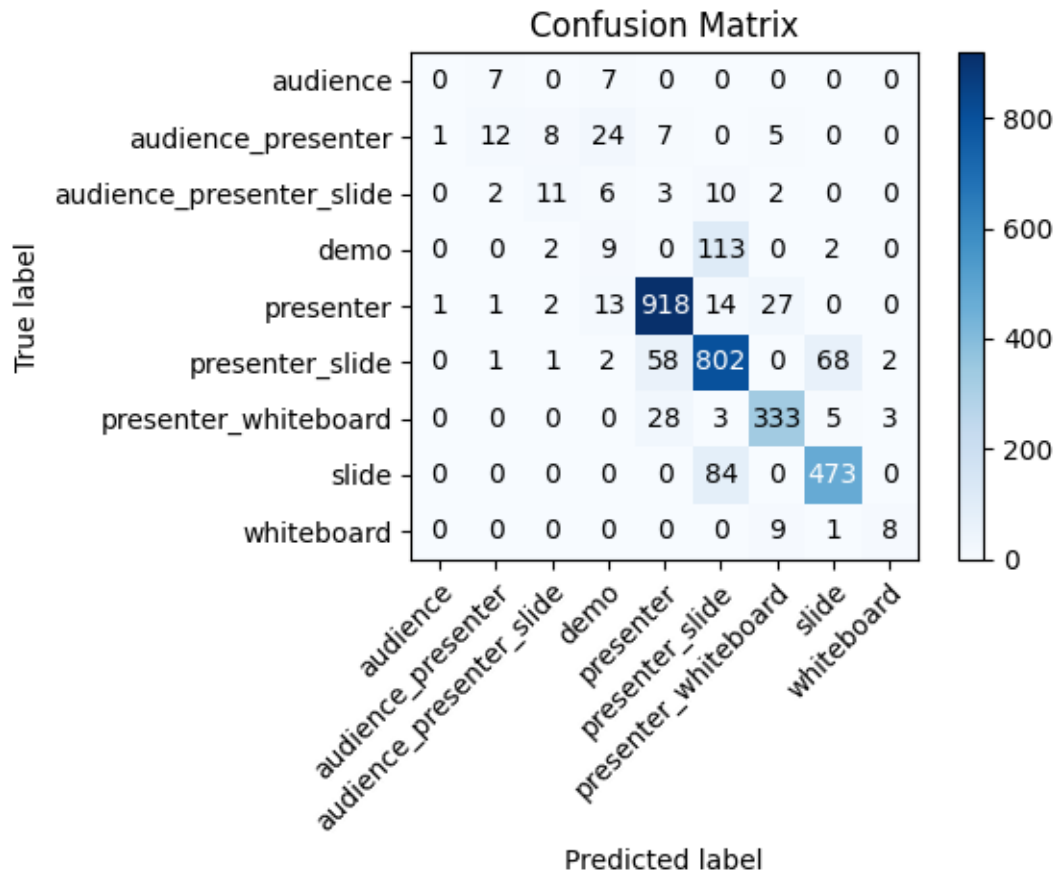


Fig. 1: Final-general slide classification model confusion matrix

The squished-image model (trained on the `train-test` dataset) slightly improves upon the results of the final-general mode by achieving an average accuracy of 83.86%, an increase of 1.24 percentage points. The results of the three-category model (trained on the `train-test-three` dataset) give a better picture of real-world performance with an average accuracy of 89.97%. Squishing the images when training on the `train-test-three` dataset does not appear to improve performance like it did with the `train-test` dataset. Training the squished-image model on the `train-test-three` dataset (squished-image-three model) yields an average accuracy of 87.21%, a decrease of 2.76 percentage points from the three-category model. In the final pipeline, we use the three-category model.

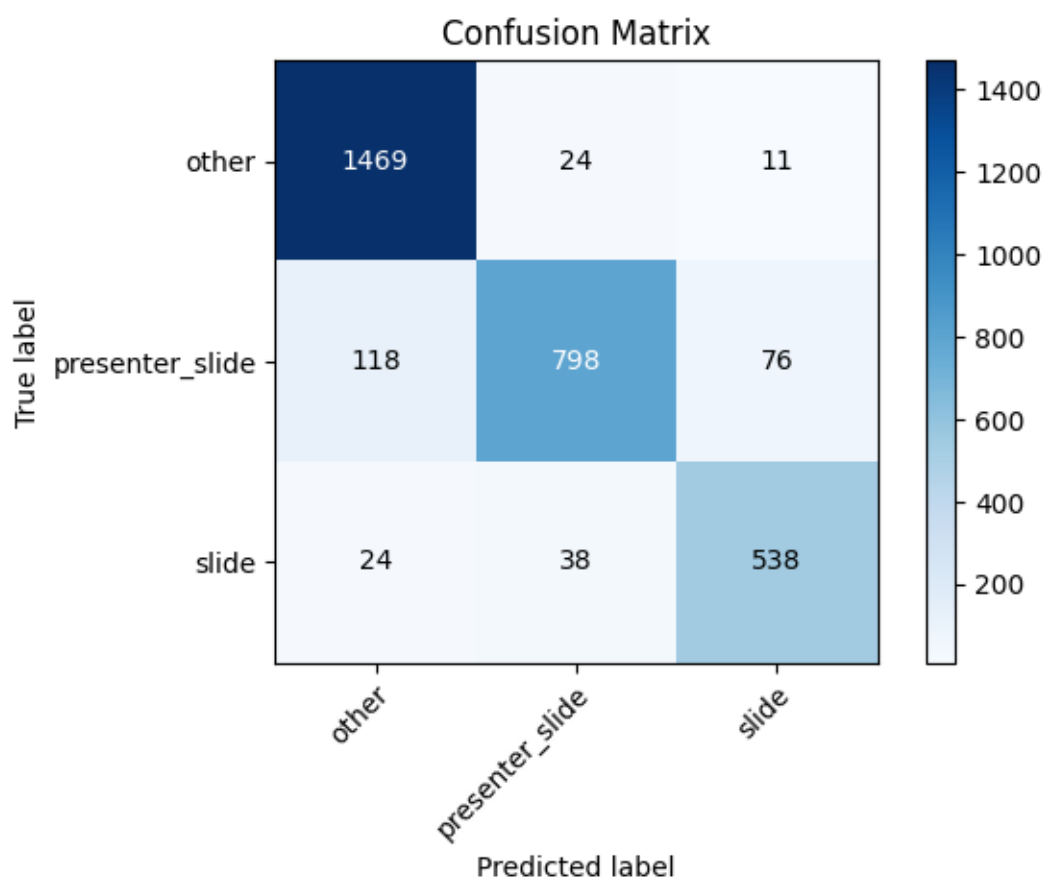


Fig. 2: Three-category slide classification model confusion matrix

7.6 Script Descriptions

Note: Visit the [Slide Classifier API](#) page to see the documentation for each function in more detail.

- **class_cluster_scikit.py:** Implements KMeans and AffinityPropagation from `sklearn.cluster` to provide a `lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster` class. The code is documented in file. The purpose is to add feature vectors using `add()`, then cluster the features, and finally return a list of files and their corresponding cluster centroids with `create_move_list()`. Three important functions and their use cases follow:
 - `create_move_list()` is called in `lecture2notes.end_to_end.cluster.ClusterFilesystem` and returns a list of filenames and their corresponding clusters.
 - `calculate_best_k()` generates a graph (saved to `best_k_value.png` if using Agg matplotlib backend) that graphs the cost (squared error) as a function of the number of centroids (value of `k`) if the algorithm is "kmeans". The point at which the graph becomes essentially linear is the optimal value of `k`.
 - `visualize()` creates a tensorboard projection of the cluster for simplified viewing and understanding.
- **class_cluster_faiss.py:** An outdated version of **class_cluster_scikit** that uses [facebookresearch/faiss](#) (specifically the kmeans implementation [documented here](#)) to provide a `Cluster` class. More details in the `class_cluster_scikit` entry above.
- **custom_nnmodules.py:** Provides a few custom (copied from [fastai](#)) `nn.Modules`.
- **inference.py:** Sets up model and provides `get_prediction()`, which takes an image and returns a prediction and extracted features.
- **lr_finder.py:** Slightly modified (allows usage of matplotlib Agg backend) code from [davidtvs/pytorch-lr-finder](#) to find the best learning rate.
- **mish.py:** Code for the mish activation function.
- **slide-classifier-fastai.ipynb:** Notebook to train simple fastai classifier on the dataset in `dataset/classifier-data`. It is outdated and not supported and only remains in the repository as an example.
- **slide_classifier_helpers.py:** Helper functions for `slide_classifier_pytorch.py`. Includes RELU to Mish activation function conversion and confusion matrix plotting functions among others.
- **slide_classifier_pytorch.py:** The main model code which uses advanced features such as the AdamW optimizer and a modified ResNet that allows for more effective pre-training/feature extracting.
- **slide-classifier-pytorch-old.py:** The old version of the slide classifier model training code. This old version was not organized as well as the current version. The old version was raw PyTorch code since it did not utilize `pytorch_lightning`.

7.7 Slide-Classifier-Pytorch Help

Output of `python slide_classifier_pytorch.py --help`:

```
usage: slide_classifier_pytorch.py [-h] [--default_root_dir DEFAULT_ROOT_DIR]
                                   [--min_epochs MIN_EPOCHS]
                                   [--max_epochs MAX_EPOCHS]
                                   [--min_steps MIN_STEPS]
                                   [--max_steps MAX_STEPS] [--lr LR]
                                   [--check_val_every_n_epoch CHECK_VAL_EVERY_N_EPOCH]
```

(continues on next page)

(continued from previous page)

```

[--gpu GPUS] [--overfit_pct OVERFIT_PCT]
[--train_percent_check TRAIN_PERCENT_CHECK]
[--val_percent_check VAL_PERCENT_CHECK]
[--test_percent_check TEST_PERCENT_CHECK]
[--amp_level AMP_LEVEL]
[--precision PRECISION] [--seed SEED]
[--profiler]
[--progress_bar_refresh_rate PROGRESS_BAR_REFRESH_
→RATE]

[--num_sanity_val_steps NUM_SANITY_VAL_STEPS]
[--use_logger {tensorboard,wandb}]
[--do_train] [--do_test]
[--load_weights LOAD_WEIGHTS]
[--load_from_checkpoint LOAD_FROM_CHECKPOINT]
[-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
[-a ARCH] [-j N]
[--train_batch_size TRAIN_BATCH_SIZE]
[--val_batch_size VAL_BATCH_SIZE]
[--test_batch_size TEST_BATCH_SIZE]
[--momentum M] [--weight_decay W] [-k K]
[--optimizer_alpha N] [--optimizer_eps N]
[--pretrained] [--random_split]
[--relu_to_mish]
[--feature_extract {normal,advanced,none}]
[-o OPTIMIZER]
DIR

```

positional arguments:

DIR path to dataset

optional arguments:

```

-h, --help show this help message and exit
--default_root_dir DEFAULT_ROOT_DIR
    Default path for logs and weights
--min_epochs MIN_EPOCHS
    Limits training to a minimum number of epochs
--max_epochs MAX_EPOCHS
    Limits training to a max number number of epochs
--min_steps MIN_STEPS
    Limits training to a minimum number number of steps
--max_steps MAX_STEPS
    Limits training to a max number number of steps
--lr LR, --learning_rate LR
    initial learning rate
--check_val_every_n_epoch CHECK_VAL_EVERY_N_EPOCH
    Check val every n train epochs.
--gpu GPUS Number of GPUs to train on or Which GPUs to train on.
    (default: -1 (all gpus))
--overfit_pct OVERFIT_PCT
    Uses this much data of all datasets (training,
    validation, test). Useful for quickly debugging or
    trying to overfit on purpose.

```

(continues on next page)

(continued from previous page)

```

--train_percent_check TRAIN_PERCENT_CHECK
    How much of training dataset to check. Useful when
    debugging or testing something that happens at the end
    of an epoch.
--val_percent_check VAL_PERCENT_CHECK
    How much of validation dataset to check. Useful when
    debugging or testing something that happens at the end
    of an epoch.
--test_percent_check TEST_PERCENT_CHECK
    How much of test dataset to check.
--amp_level AMP_LEVEL
    The optimization level to use (O1, O2, etc...) for
    16-bit GPU precision (using NVIDIA apex under the
    hood).
--precision PRECISION
    Full precision (32), half precision (16). Can be used
    on CPU, GPU or TPUs.
--seed SEED
    Seed for reproducible results. Can negatively impact
    performance in some cases.
--profiler
    To profile individual steps during training and assist
    in identifying bottlenecks.
--progress_bar_refresh_rate PROGRESS_BAR_REFRESH_RATE
    How often to refresh progress bar (in steps). In
    notebooks, faster refresh rates (lower number) is
    known to crash them because of their screen refresh
    rates, so raise it to 50 or more.
--num_sanity_val_steps NUM_SANITY_VAL_STEPS
    Sanity check runs n batches of val before starting the
    training routine. This catches any bugs in your
    validation without having to wait for the first
    validation check.
--use_logger {tensorboard,wandb}
    Which program to use for logging.
--do_train
    Run the training procedure.
--do_test
    Run the testing procedure.
--load_weights LOAD_WEIGHTS
    Loads the model weights from a given checkpoint
--load_from_checkpoint LOAD_FROM_CHECKPOINT
    Loads the model weights and hyperparameters from a
    given checkpoint.
-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --log {DEBUG,INFO,WARNING,ERROR,CRITICAL}
    Set the logging level (default: 'Info').
-a ARCH, --arch ARCH
    model architecture: alexnet | densenet121 |
    densenet161 | densenet169 | densenet201 | googlenet |
    inception_v3 | mnasnet0_5 | mnasnet0_75 | mnasnet1_0 |
    mnasnet1_3 | mobilenet_v2 | resnet101 | resnet152 |
    resnet18 | resnet34 | resnet50 | resnext101_32x8d |
    resnext50_32x4d | shufflenet_v2_x0_5 |
    shufflenet_v2_x1_0 | shufflenet_v2_x1_5 |
    shufflenet_v2_x2_0 | squeezenet1_0 | squeezenet1_1 |
    vgg11 | vgg11_bn | vgg13 | vgg13_bn | vgg16 | vgg16_bn
    | vgg19 | vgg19_bn | wide_resnet101_2 |

```

(continues on next page)

(continued from previous page)

```

        wide_resnet50_2 | efficientnet-b0 | efficientnet-b1 |
        efficientnet-b2 | efficientnet-b3 | efficientnet-b4 |
        efficientnet-b5 | efficientnet-b6 (default: resnet34)
-j N, --workers N      number of data loading workers (default: 4)
--train_batch_size TRAIN_BATCH_SIZE
                        Batch size per GPU/CPU for training.
--val_batch_size VAL_BATCH_SIZE
                        Batch size per GPU/CPU for evaluation.
--test_batch_size TEST_BATCH_SIZE
                        Batch size per GPU/CPU for testing.
--momentum M           momentum. Ranger optimizer suggests 0.95.
--weight_decay W       weight decay (default: 1e-2)
-k K, --ranger_k K     Ranger (LookAhead) optimizer k value (default: 6)
--optimizer_alpha N    Optimizer alpha parameter (default: 0.999)
--optimizer_eps N      Optimizer eps parameter (default: 1e-8)
--pretrained           use pre-trained model
--random_split          use random_split to create train and val set instead
                        of train and val folders
--relu_to_mish          convert any relu activations to mish activations
--feature_extract {normal,advanced,none}
                        If `False` or `None`, finetune the whole model. When
                        `normal`, only update the reshaped layer params. When
                        `advanced`, use fastai version of feature extracting
                        (add fancy group of layers and only update this group
                        and BatchNorm)
-o OPTIMIZER, --optimizer OPTIMIZER
                        Optimizer to use (default=AdamW)

```


SLIDE CLASSIFIER API

8.1 Class Cluster Scikit

```
class lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster(algorithm_name='kmeans',  
                                                                    num_centroids=20,  
                                                                    prefer-  
                                                                    ence=None,  
                                                                    damp-  
                                                                    ing=0.5,  
                                                                    max_iter=200)
```

add(*vector, filename*)

Adds a filename and its corresponding feature vector to the cluster object

calculate_best_k(*max_k=50*)

Implements elbow method to graph the cost (squared error) as a function of the number of centroids (value of k) The point at which the graph becomes essentially linear is the optimal value of k. Only works if *algorithm* is “kmeans”.

create_affinity_propagation(*preference, damping, max_iter, store=True*)

Create and fit an affinity propagation cluster

create_algorithm_if_none()

Creates algorithm if it has not been created (if it equals None) based on *algorithm_name* set in `__init__`

create_kmeans(*num_centroids, store=True*)

Create and fit a kmeans cluster

get_closest_sample_filenames_to_centroids()

Return the sample indexes that are closest to each centroid. Ex: If [0,8] is returned then X[0] (X is training data/vectors) is the closest point in X to centroid 0 and X[8] is the closest to centroid 1

get_labels()

get_move_list()

Creates a dictionary of file names and their corresponding centroid numbers

get_num_clusters()

get_vector_array()

Return a numpy array of the list of vectors stored in `self.vectors`

get_vectors()

predict(*array*)

Wrapper function for algorithm.predict. Creates algorithm if it has not been created.

visualize(*tensorboard_dir*)

Creates tensorboard projection of cluster for simplified viewing and understanding

8.2 Custom nn.Modules

class lecture2notes.models.slide_classifier.custom_nnmodules.**AdaptiveConcatPool2d**(*sz=None*)

Layer that concats AdaptiveAvgPool2d and AdaptiveMaxPool2d [https://docs.fast.ai/layers.html#](https://docs.fast.ai/layers.html#AdaptiveConcatPool2d)

[AdaptiveConcatPool2d](https://docs.fast.ai/layers.html#AdaptiveConcatPool2d)

forward(*x*)

Defines the computation performed at every call.

Should be overridden by all subclasses.

Note: Although the recipe for forward pass needs to be defined within this function, one should call the Module instance afterwards instead of this since the former takes care of running the registered hooks while the latter silently ignores them.

training: `bool`

8.3 GradCAM

class lecture2notes.models.slide_classifier.grad_cam.**BackPropagation**(*model*)

forward(*image*)

generate()

class lecture2notes.models.slide_classifier.grad_cam.**Deconvnet**(*model*)

“Striving for Simplicity: the All Convolutional Net” <https://arxiv.org/pdf/1412.6806.pdf> Look at Figure 1 on page 8.

class lecture2notes.models.slide_classifier.grad_cam.**GradCAM**(*model, candidate_layers=None*)

“Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization” <https://arxiv.org/pdf/1610.02391.pdf> Look at Figure 2 on page 4

generate(*target_layer*)

class lecture2notes.models.slide_classifier.grad_cam.**GuidedBackPropagation**(*model*)

“Striving for Simplicity: the All Convolutional Net” <https://arxiv.org/pdf/1412.6806.pdf> Look at Figure 1 on page 8.

lecture2notes.models.slide_classifier.grad_cam.**get_device**(*cuda*)

lecture2notes.models.slide_classifier.grad_cam.**load_images**(*image_paths, input_size*)

lecture2notes.models.slide_classifier.grad_cam.**main**(*args*)

Visualize model responses given multiple images

```
lecture2notes.models.slide_classifier.grad_cam.occlusion_sensitivity(model,
                                                                    images, ids,
                                                                    mean=None,
                                                                    patch=35,
                                                                    stride=1,
                                                                    n_batches=128)

“Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization” https://arxiv.org/pdf/1610.02391.pdf Look at Figure A5 on page 17

Originally proposed in: “Visualizing and Understanding Convolutional Networks” https://arxiv.org/abs/1311.2901

lecture2notes.models.slide_classifier.grad_cam.preprocess(image_path, input_size)

lecture2notes.models.slide_classifier.grad_cam.save_gradcam(filename, gcam,
                                                            raw_image,
                                                            paper_cmap=False)

lecture2notes.models.slide_classifier.grad_cam.save_gradient(filename, gradient)

lecture2notes.models.slide_classifier.grad_cam.save_sensitivity(filename, maps)
```

8.4 Learning Rate Finder

```
class lecture2notes.models.slide_classifier.lr_finder.ExponentialLR(optimizer,
                                                                    end_lr,
                                                                    num_iter,
                                                                    last_epoch=-1)
```

Exponentially increases the learning rate between two boundaries over a number of iterations.

Parameters

- **optimizer** (*torch.optim.Optimizer*) – wrapped optimizer.
- **end_lr** (*float, optional*) – the initial learning rate which is the lower boundary of the test. Default: 10.
- **num_iter** (*int, optional*) – the number of iterations over which the test occurs. Default: 100.
- **last_epoch** (*int*) – the index of last epoch. Default: -1.

get_lr()

```
class lecture2notes.models.slide_classifier.lr_finder.LRFinder(model, optimizer,
                                                                criterion,
                                                                device=None,
                                                                memory_cache=True,
                                                                cache_dir=None)
```

Learning rate range test.

The learning rate range test increases the learning rate in a pre-training run between two boundaries in a linear or exponential manner. It provides valuable information on how well the network can be trained over a range of learning rates and what is the optimal learning rate.

Parameters

- **model** (*torch.nn.Module*) – wrapped model.

- **optimizer** (*torch.optim.Optimizer*) – wrapped optimizer where the defined learning is assumed to be the lower boundary of the range test.
- **criterion** (*torch.nn.Module*) – wrapped loss function.
- **device** (*str or torch.device, optional*) – a string (“cpu” or “cuda”) with an optional ordinal for the device type (e.g. “cuda:X”, where X is the ordinal). Alternatively, can be an object representing the device on which the computation will take place. Default: None, uses the same device as *model*.
- **memory_cache** (*boolean*) – if this flag is set to True, *state_dict* of model and optimizer will be cached in memory. Otherwise, they will be saved to files under the *cache_dir*.
- **cache_dir** (*string*) – path for storing temporary files. If no path is specified, system-wide temporary directory is used. Notice that this parameter will be ignored if *memory_cache* is True.

Example

```
>>> lr_finder = LRFinder(net, optimizer, criterion, device="cuda")
>>> lr_finder.range_test(dataloader, end_lr=100, num_iter=100)
```

Cyclical Learning Rates for Training Neural Networks: <https://arxiv.org/abs/1506.01186> fastai/lr_find: <https://github.com/fastai/fastai>

plot(*skip_start=10, skip_end=5, log_lr=True*)

Plots the learning rate range test.

Parameters

- **skip_start** (*int, optional*) – number of batches to trim from the start. Default: 10.
- **skip_end** (*int, optional*) – number of batches to trim from the start. Default: 5.
- **log_lr** (*bool, optional*) – True to plot the learning rate in a logarithmic scale; otherwise, plotted in a linear scale. Default: True.

range_test(*train_loader, val_loader=None, end_lr=10, num_iter=100, step_mode='exp', smooth_f=0.05, diverge_th=5*)

Performs the learning rate range test.

Parameters

- **train_loader** (*torch.utils.data.DataLoader*) – the training set data loader.
- **val_loader** (*torch.utils.data.DataLoader, optional*) – if *None* the range test will only use the training loss. When given a data loader, the model is evaluated after each iteration on that dataset and the evaluation loss is used. Note that in this mode the test takes significantly longer but generally produces more precise results. Default: None.
- **end_lr** (*float, optional*) – the maximum learning rate to test. Default: 10.
- **num_iter** (*int, optional*) – the number of iterations over which the test occurs. Default: 100.
- **step_mode** (*str, optional*) – one of the available learning rate policies, linear or exponential (“linear”, “exp”). Default: “exp”.
- **smooth_f** (*float, optional*) – the loss smoothing factor within the [0, 1] interval. Disabled if set to 0, otherwise the loss is smoothed using exponential smoothing. Default: 0.05.

- **diverge_th** (*int*, *optional*) – the test is stopped when the loss surpasses the threshold: `diverge_th * best_loss`. Default: 5.

reset()

Restores the model and optimizer to their initial states.

class lecture2notes.models.slide_classifier.lr_finder.**LinearLR**(*optimizer*, *end_lr*,
num_iter,
last_epoch=-1)

Linearly increases the learning rate between two boundaries over a number of iterations.

Parameters

- **optimizer** (*torch.optim.Optimizer*) – wrapped optimizer.
- **end_lr** (*float*, *optional*) – the initial learning rate which is the lower boundary of the test. Default: 10.
- **num_iter** (*int*, *optional*) – the number of iterations over which the test occurs. Default: 100.
- **last_epoch** (*int*) – the index of last epoch. Default: -1.

get_lr()

class lecture2notes.models.slide_classifier.lr_finder.**StateCacher**(*in_memory*,
cache_dir=None)

retrieve(*key*)

store(*key*, *state_dict*)

8.5 Mish

lecture2notes.models.slide_classifier.mish.**f_mish**(*input*, *inplace=False*)

Applies the mish function element-wise: $mish(x) = x * \tanh(\text{softplus}(x)) = x * \tanh(\ln(1 + \exp(x)))$

class lecture2notes.models.slide_classifier.mish.**mish**(*inplace=False*)

Applies the mish function element-wise: $mish(x) = x * \tanh(\text{softplus}(x)) = x * \tanh(\ln(1 + \exp(x)))$

Shape:

- Input: (N, *) where * means, any number of additional dimensions
- Output: (N, *), same shape as the input

Examples

```
>>> m = mish()
>>> input = torch.randn(2)
>>> output = m(input)
```

forward(*input*)

Forward pass of the function.

training: bool

8.6 Inference

```
lecture2notes.models.slide_classifier.inference.get_prediction(model, image,
                                                             percent=False, ex-
                                                             tract_features=True)
lecture2notes.models.slide_classifier.inference.initialize_model(arch, num_classes)
lecture2notes.models.slide_classifier.inference.load_model(model_path='model_best.ckpt')
    Load saved model from model_path.
lecture2notes.models.slide_classifier.inference.load_model_deprecated(model_path='model_best.pth.tar')
    Load saved model trained using old script (“.pth.tar” file extension is old format).
lecture2notes.models.slide_classifier.inference.transform_image(image,
                                                                input_size=224)
```

8.7 Slide Classifier Helpers

```
lecture2notes.models.slide_classifier.slide_classifier_helpers.convert_relu_to_mish(model)
    Find all of the nn.ReLU activation functions in model and replace them with mish.
lecture2notes.models.slide_classifier.slide_classifier_helpers.plot_confusion_matrix(y_pred,
                                                                                     y_true,
                                                                                     classes,
                                                                                     nor-
                                                                                     mal-
                                                                                     ize=False,
                                                                                     ti-
                                                                                     tle='Confusion
                                                                                     Ma-
                                                                                     trix',
                                                                                     cmap=<matplotlib
                                                                                     ob-
                                                                                     ject>,
                                                                                     save_path=None)
```

This function prints and plots the confusion matrix. Normalization can be applied by setting `normalize=True` https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html#sphx-glr-auto-examples-model-selection-plot-confusion-matrix-py.

8.8 Slide Classifier PyTorch

```
class lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier(hparams)
    The main slide classifier model code.

    static add_model_specific_args(parent_parser)

    static calculate_stats(output, target)
        Used for the training, validation, and testing steps to calculate various statistics.

        Parameters
        • output (torch.tensor) – the output from the model
        • target (torch.tensor) – the ground-truth target classes

        Returns a tuple of tensors in the form (accuracy, precision, recall, f_score)
        Return type [tuple]
```

configure_optimizers()

Create the optimizers and schedulers.

forward(*args, **kwargs)

Passes **args* and ***kwargs* to `self.classification_model` since `SlideClassifier` is a wrapper for the classification model.

get_input_size()

Uses the `hparams.arch` to return the image input size to the model.

initialize_model(num_classes)

Create the classification model. Modifies the standard models by adding extra layers to improve performance if `feature_extract` is advanced.

Parameters `num_classes (int)` – the number of classes in the data (number of output features)

Returns the modified pytorch model processed by the configuration options specified

Return type [pytorch model]

prepare_data()

Creates the PyTorch Datasets using `datasets.ImageFolder` and applying appropriate transforms. If `hparams.use_random_split` is True then the dataset will be randomly split 80% for training and 20% for testing. If `hparams.use_random_split` is True then the dataset folder should contain a folder for each class. If it is False then there should be a folder for each split (named “train” and “val”) where each split folder contains a folder for each class. [ImageFolder Documentation](#)

This function will also run `:meth:`~slide_classifier_pytorch.SlideClassifier.initialize_model`` with `len(self.hparams.classes))` as the `num_classes` argument if the classification model as not already been initialized in the `__init__` function.

set_parameter_requires_grad(model)

This helper function sets the `.requires_grad` attribute of the parameters in the model to False when we are feature extracting. By default, when we load a pretrained model all of the parameters have `.requires_grad=True`, which is fine if we are training from scratch or finetuning. However, if we are feature extracting and only want to compute gradients for the newly initialized layer then we want all of the other parameters to not require gradients.

test_dataloader()

Return the validation dataloader. The test process uses the same data as validation but calculates a classification report and displays a confusion matrix.

test_epoch_end(outputs)

Create confusion matrix and calculate a sklearn classification report.

test_step(batch, batch_idx)

Perform a test step. See the [PyTorch Lightning documentation for test_step](#) for more info.

train_dataloader()

Create train dataloader if it has not already been created, otherwise return the stored dataloader.

training: bool**training_step(batch, batch_idx)**

Perform a training step. See the [PyTorch Lightning Docs](#) for more info.

val_dataloader()

Create validation (val) dataloader

static validation_epoch_end(outputs, log_prefix='val')

Compute average statistics after a validation epoch completes.

validation_step(*batch*, *batch_idx*)

Perform a validation step. See the [PyTorch Lightning documentation](#) for `validation_step` for more info.

SUMMARIZATION MODELS

Warning: This page discusses the actual models and algorithms used to summarize text. If you want to compare the methods available for summarizing text as used in the End-To-End process please visit [Combination and Summarization](#). If you are interested in the code behind the models, would like to reproduce results, or want to adapt upon the low-level summarization components, you're in the right place.

Neural: The extractive summarizers are provided by [HHousen/TransformerSum](#) and the abstractive ones from [huggingface/transformers](#) (abstractive was originally accomplished with [HHousen/DocSum](#)). Please see those repositories for details on the exact implementation details of the models. Some of the architectures are HHousen's, some are partly HHousen's, and many are from other research projects.

Non-Neural Algorithms: The [sumy](#) ([Sumy GitHub](#)) package provides some non-neural summarization algorithms, mainly the methods for [generic_extractive_sumy\(\)](#) such as `lsa`, `luhn`, `lex_rank`, `text_rank`, `edmundson`, and `random`.

Note: The [summa](#) ([Summa GitHub](#)) package is used to extract keywords using the TextRank algorithm in [keyword_based_ext\(\)](#).

Note: All other models/algorithms are, to the best of my knowledge, novel and are directly implemented as part of this project. See [Combination and Summarization](#) for details.

E2E GENERAL INFORMATION

The end-to-end approach. One command to take a video file and return summarized notes.

Run `python main.py <path to video>` to get a notes file. See [Summarizing a lecture video](#) for a brief introduction.

10.1 Overall Explanation

First, frames are extracted once every second. Each frame is classified using the slide classifier (see [Overview](#)). Next, frames that were classified as `slide` are processed by a black border removal algorithm, which is a simple program that crop to the largest rectangle in an image if the edge pixel values of the image are all close to zero. Thus, screen-captured slide frames that have black bars on the sides from a presentation created with a 4:3 aspect ratio but recorded at 16:9 can be interpreted correctly.

Frames that were classified as `presenter_slide` are perspective cropped through feature matching and contour/hough lines algorithms. This process removes duplicate slides and crops images from the `presenter_slide` class to only contain the side. However, to clean up any duplicates that may remain and to find the best representation of each slide, the slide frames are clustered using our custom `segment` clustering algorithm.

At this point, the process has identified the set of unique slides presented in the lecture video. The next step of the pipeline is to process these slides by performing an SSA, which is an algorithm that extracts formatted text from each slide. After that, figures are extracted from the slides and attached to the SSA (see [Slide Structure Analysis \(SSA\)](#)) for each slide. A figure is an image, chart, table, or diagram. After the system has extracted the visual content, it begins processing the audio. The audio is transcribed automatically using the Vosk small 36MB model.

After the audio is transcribed, the system has a textual representation of both visual and auditory data, which need to be combined and summarized to create the final output. If the user desires notes then the SSA will be used for formatting, otherwise, there are tens of different ways of combining and summarizing the audio and slide transcripts, which are discussed in [Combination and Summarization](#).

10.2 Script Descriptions

These descriptions are short and concise. For more information about some of the larger, more complicated files visit their respective pages on the left.

- **border_removal:** The black border removal algorithm is a simple instruction set that finds the largest rectangle in the image if the edge pixel values of the image are all $< \gamma$ and then crops to that rectangle.
- **cluster:** Provides `lecture2notes.end_to_end.cluster.ClusterFilesystem` class, which clusters images from a directory and saves them to disk in folders corresponding to each centroid.

- **corner_crop_transform:** Provides functions to detect the bounding box of a slide in a frame and automatically crop to that bounding box. The `lecture2notes.end_to_end.corner_crop_transform.all_in_folder()` method is used by the main script. See *Perspective Cropping & Corner Detection* for more information. This is one of the two components used to remove duplicate slides and crop `presenter_slide` images to only contain the slide. You can learn more about the overall perspective cropping process at *Perspective Cropping*.
- **figure_detection:** The figure extraction algorithm identifies and saves images, charts, tables, and diagrams from slide frames so that they can be shown in the final summary. See *Figure Detection Algorithm* for more information.
- **frames_extractor:** Provides `lecture2notes.end_to_end.frames_extractor.extract_frames()`, which extracts frames from `input_video_path` at quality level `quality` (best quality is 2) every `extract_every_x_seconds` seconds and saves them to `output_path`.
- **helpers:** A small file of helper functions to reduce duplicate code. See *Helpers*.
- **imghash:** Provides functions to detect near duplicate images using image hashing methods from the `imagehash` library. `lecture2notes.end_to_end.imghash.sort_by_duplicates()` will create lists of similar images and `lecture2notes.end_to_end.imghash.remove_duplicates()` will remove those duplicates and keep the last file (when sorted alphanumerically descending).
- **main:** The master file that brings all of the components in this directory together by calling the functions provided by the components. Implements a `skip_to` variable that can be set to skip to a certain step of the process. This is useful if a previous step completed but the overall process failed. The `--help` is *located below*.
- **ocr:** OCR processing uses the `pytesseract` (GitHub) package. “Python-tesseract is an optical character recognition (OCR) tool for python. That is, it will recognize and ‘read’ the text embedded in images. Python-tesseract is a wrapper for Google’s Tesseract-OCR Engine.” This page has good information to improve results from tesseract. See `ocr.all_in_folder()` and `lecture2notes.end_to_end.ocr.write_to_file()`.
- **segment_cluster:** `SegmentCluster` iterates through frames in order and splits based on large visual differences. These differences are measured by the cosine difference between the feature vectors (2nd to last layer or right before the softmax) outputted by the slide classifier. This class behaves similarly to `lecture2notes.end_to_end.cluster.ClusterFilesystem` in that it also provides `extract_and_add_features()` and `transfer_to_filesystem()`.
- **sift_matcher:** One of the components used to remove duplicate slides and crop `presenter_slide` images to only contain the slide. You can learn more about the `sift_matcher` at *SIFT Matcher & Perspective Cropping* and the overall perspective cropping process at *Perspective Cropping*.
- **slide_classifier:** Provides `lecture2notes.end_to_end.slide_classifier.classify_frames()` which automatically sorts images (the extracted frames) using the slide-classifier model. The inference script in `models/slide_classifier` is used.
- **spell_check:** Contains the `SpellChecker` class, which can spell check a string with `check()` or a list of strings with `check_all()`. With both functions, the best correction is returned.
- **summarization_approaches:** Many summarization models and algorithms for use with `end_to_end/main.py`. The `lecture2notes.end_to_end.summarization_approaches.cluster()` is probably the most interesting method from this file.
- **transcript_downloader:** Provides the `lecture2notes.end_to_end.transcript_downloader.TranscriptDownloader` class, which downloads transcripts from YouTube using the YouTube API or `youtube-dl`. `youtube-dl` is the recommended method since it does not require an API key and is significantly more reliable than the YouTube API.
- **youtube_api:** Function to use YouTube API with `key` or `client_secret.json`. See `youtube_api.init_youtube()`.

10.3 Main Script Help

Output of `python -m lecture2notes.end_to_end.main --help`:

```
usage: main.py [-h] [-s N] [-d PATH] [-id] [--custom_id CUSTOM_ID] [-rm]
[--extract_frames_quality EXTRACT_FRAMES_QUALITY]
[--extract_every_x_seconds EXTRACT_EVERY_X_SECONDS]
[--slide_classifier_model_path SLIDE_CLASSIFIER_MODEL_PATH]
[--east_path EAST_PATH] [-c {silence,speech,none}] [-rd]
[-cm {normal,segment}]
[-ca {only_asr,concat,full_sents,keyword_based}]
[-sm {none,full_sents} [{none,full_sents} ...]]
[-sx {none,cluster,lsa,luhn,lex_rank,text_rank,edmundson,random}]
[-sa {none,presumm,sshleifer/distilbart-cnn-12-6,patrickvonplaten/bert2bert-cnn-daily_
→mail,facebook/bart-large-cnn,allenai/led-large-16384-arxiv,patrickvonplaten/led-large-
→16384-pubmed,google/pegasus-billsum,google/pegasus-cnn-dailymail,google/pegasus-pubmed,
→google/pegasus-arxiv,google/pegasus-wikihow,google/pegasus-big_patent}]
[-ss {structured_joined,none}]
[--structured_joined_summarization_method {none,abstractive,extractive}]
[--structured_joined_abs_summarizer {presumm,sshleifer/distilbart-cnn-12-6,
→patrickvonplaten/bert2bert-cnn-daily-mail,facebook/bart-large-cnn,allenai/led-large-
→16384-arxiv,patrickvonplaten/led-large-16384-pubmed,google/pegasus-billsum,google/
→pegasus-cnn-dailymail,google/pegasus-pubmed,google/pegasus-arxiv,google/pegasus-
→wikihow,google/pegasus-big_patent}]
[--structured_joined_ext_summarizer {lsa,luhn,lex_rank,text_rank,edmundson,random}]
[-tm {sphinx,google,youtube,deepspeech,vosk,wav2vec}]
[--transcribe_segment_sentences]
[--custom_transcript_check CUSTOM_TRANSCRIPT_CHECK]
[-sc {ocr,transcript} [{ocr,transcript} ...]] [--video_id ID]
[--transcribe_model_dir DIR] [--abs_hf_api]
[--abs_hf_api_overall] [--tensorboard PATH]
[--bart_checkpoint PATH] [--bart_state_dict_key PATH]
[--bart_fairseq] [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
DIR
```

End-to-End Conversion of Lecture Videos to Notes using ML

positional arguments:

DIR path to video

optional arguments:

```
-h, --help            show this help message and exit
-s N, --skip_to N     set to > 0 to skip specific processing steps
-d PATH, --process_dir PATH
                    path to the processing directory (where extracted
                    frames and other files are saved), set to "automatic"
                    to use the video's folder (default: ./)
-id, --auto_id         automatically create a subdirectory in `process_dir`
                    with a unique id for the video and change
                    `process_dir` to this new directory
--custom_id CUSTOM_ID
                    same as `--auto_id` but will create a subdirectory
                    using this value instead of a random id
```

(continues on next page)

(continued from previous page)

```

--rm, --remove          remove `process_dir` once conversion is complete
--extract_frames_quality EXTRACT_FRAMES_QUALITY
                        ffmpeg quality of extracted frames
--extract_every_x_seconds EXTRACT_EVERY_X_SECONDS
                        how many seconds between extracted frames
--slide_classifier_model_path SLIDE_CLASSIFIER_MODEL_PATH
                        path to the slide classification model checkpoint
--east_path EAST_PATH
                        path to the EAST text detector model
-c {silence,speech,none}, --chunk {silence,speech,none}
                        split the audio into small chunks on `silence` using
                        PyDub or voice activity `speech` using py-webrtcvad.
                        set to 'none' to disable. Recommend 'speech' for
                        DeepSpeech and 'none' for Vosk. (default: 'none').
-rd, --remove_duplicates
                        remove duplicate slides before perspective cropping
                        and before clustering (helpful when `--cluster_method`
                        is `segment`)
-cm {normal,segment}, --cluster_method {normal,segment}
                        which clustering method to use. `normal` uses a
                        clustering algorithm from scikit-learn and `segment`
                        uses the special method that iterates through frames
                        in order and splits based on large visual differences
-ca {only_asr,concat,full_sents,keyword_based}, --combination_algo {only_asr,concat,full_
↪sents,keyword_based}
                        which combination algorithm to use. more information
                        in documentation.
-sm {none,full_sents} [{none,full_sents} ...], --summarization_mods {none,full_sents} [
↪{none,full_sents} ...]
                        modifications to perform during summarization process.
                        each modification is run between the combination and
                        extractive stages. more information in documentation.
-sx {none,cluster,lsa,luhn,lex_rank,text_rank,edmundson,random}, --summarization_ext
↪{none,cluster,lsa,luhn,lex_rank,text_rank,edmundson,random}
                        which extractive summarization approach to use. more
                        information in documentation.
-sa {none,presumm,sshleifer/distilbart-cnn-12-6,patrickvonplaten/bert2bert_cnn_daily_
↪mail,facebook/bart-large-cnn,allenai/led-large-16384-arxiv,patrickvonplaten/led-large-
↪16384-pubmed,google/pegasus-billsum,google/pegasus-cnn_dailymail,google/pegasus-pubmed,
↪google/pegasus-arxiv,google/pegasus-wikihow,google/pegasus-big_patent}, --
↪summarization_abs {none,presumm,sshleifer/distilbart-cnn-12-6,patrickvonplaten/
↪bert2bert_cnn_daily_mail,facebook/bart-large-cnn,allenai/led-large-16384-arxiv,
↪patrickvonplaten/led-large-16384-pubmed,google/pegasus-billsum,google/pegasus-cnn_
↪dailymail,google/pegasus-pubmed,google/pegasus-arxiv,google/pegasus-wikihow,google/
↪pegasus-big_patent}
                        which abstractive summarization approach/model to use.
                        more information in documentation.
-ss {structured_joined,none}, --summarization_structured {structured_joined,none}
                        An additional summarization algorithm that creates a
                        structured summary with figures, slide content (with
                        bolded area), and summarized transcript content from
                        the SSA (Slide Structure Analysis) and transcript JSON

```

(continues on next page)

(continued from previous page)

```

data.
--structured_joined_summarization_method {none,abstractive,extractive}
    The summarization method to use during
    `structured_joined` summarization.
--structured_joined_abs_summarizer {presumm,sshleifer/distilbart-cnn-12-6,
↳patrickvonplaten/bert2bert_cnn_daily_mail,facebook/bart-large-cnn,allenai/led-large-
↳16384-arxiv,patrickvonplaten/led-large-16384-pubmed,google/pegasus-billsum,google/
↳pegasus-cnn_dailymail,google/pegasus-pubmed,google/pegasus-arxiv,google/pegasus-
↳wikihow,google/pegasus-big_patent}
    The abstractive summarizer to use during
    `structured_joined` summarization (to create summaries
    of each slide) if
    `structured_joined_summarization_method` is
    'abstractive'.
--structured_joined_ext_summarizer {lsa,luhn,lex_rank,text_rank,edmundson,random}
    The extractive summarizer to use during
    `structured_joined` summarization (to create summaries
    of each slide) if
    `--structured_joined_summarization_method` is
    'extractive'.
-tm {sphinx,google,youtube,deepspeech,vosk,wav2vec}, --transcription_method {sphinx,
↳google,youtube,deepspeech,vosk,wav2vec}
    specify the program that should be used for
    transcription. CMU Sphinx: use pocketsphinx Google
    Speech Recognition: probably will require chunking
    (online, free, max 1 minute chunks) YouTube: download
    a video transcript from YouTube based on `--video_id`
    DeepSpeech: Use the deepspeech library (fast with good
    GPU) Vosk: Use the vosk library (extremely small low-
    resource model with great accuracy, this is the
    default) Wav2Vec: State-of-the-art speech-to-text
    model through the `huggingface/transformers` library.
--transcribe_segment_sentences
    Disable DeepSegment automatic sentence boundary
    detection. Specifying this option will output
    transcripts without punctuation.
--custom_transcript_check CUSTOM_TRANSCRIPT_CHECK
    Check if a transcript file (followed by an extension of
    vtt, srt, or sbv) with the specified name is in the
    processing folder and use it instead of running
    speech-to-text.
-sc {ocr,transcript} [{ocr,transcript} ...], --spell_check {ocr,transcript} [{ocr,
↳transcript} ...]
    option to perform spell checking on the ocr results of
    the slides or the voice transcript or both
--video_id ID
    id of youtube video to get subtitles from. set
    `--transcription_method` to `youtube` for this
    argument to take effect.
--transcribe_model_dir DIR
    path containing the model files for Vosk/DeepSpeech if
    `--transcription_method` is set to one of those
    models. See the documentation for details.

```

(continues on next page)

(continued from previous page)

```
--abs_hf_api          use the huggingface inference API for abstractive
                        summarization tasks
--abs_hf_api_overall  use the huggingface inference API for final overall
                        abstractive summarization task
--tensorboard PATH    Path to tensorboard logdir. Tensorboard not used if
                        not set. Tensorboard only used to visualize cluster
                        primarily for debugging.
-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --log {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Set the logging level (default: 'Info').
```


COMBINATION AND SUMMARIZATION

Once the system has a plain text representation of the visual and auditory information portrayed in the lecture, it can begin to summarize and create the final notes. We break this into a four-stage process: combination, modification, extractive summarization, and abstractive summarization. Some of these steps can be turned off completely. For example, it is possible to combine the two transcripts and then summarize the result using an abstractive model, thus skipping the modifications and extractive summarization steps. The word “transcripts” in this section refers to the audio transcript and text content extracted from the slides.

11.1 Structured Joined Summarization

The structured joined summarization method is separate from the four-stage process. This method summarizes slides using the SSA (see *Slide Structure Analysis (SSA)*) and audio transcript to create an individual summary for each slide. The words spoken from the beginning of one slide to the start of the next to the nearest sentence are considered the transcript that belongs to that slide. Either DeepSpeech, Vosk, or manual transcription must be used to summarize using `structured_joined` because they are the only models that output the start times of each word spoken. The transcript that belongs to each slide is independently summarized using extractive (see *Extractive Summarization*) or abstractive summarization (see *Abstractive Summarization*). The content from each slide is formatted following the SSA and presented to the user. Only paragraphs on the slide longer than three characters are included. The result contains the following for each unique slide identified: a summary of the words spoken while that slide was displayed, the formatted content on the slide, an image of the slide, and extracted figures from the slide.

You can learn more in the docstring for the `lecture2notes.end_to_end.summarization_approaches.structured_joined_sum()` function.

11.2 Combination

There are five methods of combining the transcripts:

1. `only_asr`: only uses the audio transcript (deletes the slide transcript)
2. `only_slides`: only uses the slides transcript (deletes the audio transcript)
3. `concat`: appends audio transcript to slide transcript
4. `full_sents`: audio transcript is appended to only the complete sentences of the slide transcript
5. `keyword_based`: selects a certain percentage of sentences from the audio transcript based on keywords found in the slides transcript

11.2.1 Full Sentences Algorithm

Complete sentences are detected by tokenizing the input text using a Spacy model and then selecting sentences that end with punctuation and contain at least one subject, predicate, and object (two nouns and one verb). If the number of tokens in the text containing only complete sentences is greater than a percentage of the number of tokens in the original text then the algorithm returns the complete sentences, otherwise, it will return the original text. This check ensures that a large quantity of content is not lost. By default, the cutoff percentage is 70%. You can view the code here: `lecture2notes.end_to_end.summarization_approaches.full_sents()`.

11.2.2 Keyword Based Algorithm

Since the text extracted from slides may contain many incomplete ideas that the presenter clarifies through speech, it may be necessary to completely disregard the slide transcript. However, in the case where the slide transcript contains a loose connection of ideas we view it as an incomplete summary of the lecture. Thus, the information in the slide transcript can be used to select the most important sentences from the audio transcript, thus preventing the loss of significant amounts of information.

First, keywords are extracted from the slide transcript using TextRank. Next, this list of keywords is used as the vocabulary in the creation of a TF-IDF (term frequency-inverse document frequency) vectorizer. The TF-IDF vectorizer is equivalent to converting the voice transcript to a matrix of token counts followed by performing the TF-IDF transformation. After the TF-IDF vectorizer is created, the sentences are extracted from the transcript text using a Spacy model. Finally, the document term matrix is created by fitting the TF-IDF vectorizer on the sentences from the transcript text and then transforming the transcript text sentences using the fitted vectorizer. Next, the algorithm calculates the singular value decomposition (SVD), which is known as latent semantic analysis (LSA) in the context of TF-IDF matrices, of the document term matrix. To compute the ranks of each sentence we pass Σ and V to the rank computation algorithm, which for each row vector in the transposed V matrix finds the sum of $s^2 * v^2$ where s is the row of Σ and v is the column of V . Finally, the algorithm selects the sentences in the top 70% sorted by rank. The sentences are sorted by their original position in the document to ensure the order of the output sentences follows their order in the input document.

11.3 Modifications

Modifications change the output of the combination algorithm before it is sent to the summarization stages. The only modification is a function that will extract the complete sentences from the combined audio and slide transcript. By default, this modification is turned off. Importantly, the modification framework is extendable so that future modifications can be implemented easily.

11.4 Extractive Summarization

There are two high-level extractive summarization methods: `cluster`, an advanced novel algorithm, and `generic`, which is a collection of several standard extractive summarization algorithms such as TextRank, LSA, and Edmundson.

11.4.1 The Cluster Method

The `cluster` algorithm extracts features from the text, clusters based on those features, and summarizes each cluster.

Feature extraction can be done in four ways:

1. `neural_hf`: Uses a large transformer model (RoBERTa by default) to extract features.
2. `neural_sbent`: Uses special BERT and RoBERTa models fine-tuned to extract sentence embeddings. This is the default option.
3. `spacy`: Uses the `en_core_web_lg` (large model is preferred over a smaller model since large models have “read” word vectors) Spacy model to loop through sentences and store their “vector” values, which is an average of the token vectors.
4. `bow`: The name “bow” stands for “bag of words.” This method is fast since it is based on word frequencies throughout the input text. The implementation is similar to the combination keyword based algorithm (see *Keyword Based Algorithm*) but instead of using keywords from another document, the keywords are calculated using the TF-IDF vectorizer. The TF-IDF-weighted document-term matrix contains the features that are clustered.

The feature vectors are clustered using the KMeans algorithm. The user must specify the number of clusters they want, which corresponds to the number of topics discussed in the lecture. Mini batch KMeans is supported if a reduction in computation time is desired and obtaining slightly worse results is acceptable.

Summarization can be done in two ways:

1. `extractive`: Computes the SVD of the document-term matrix and calculates ranks using the sentence rank computation algorithm. This option requires that features were extracted using `bow` because this method needs the document-term matrix produced during `bow` feature extraction in order to compute sentence rankings.
2. `abstractive`: Summarizes text using a seq2seq transformer model trained on abstractive summarization. The default model is a distilled version of BART.

The clusters are summarized sequentially. However, when using the `extractive` summarization method, the ranks of all sentences are only calculated once before clustering. Before summarization, the sentences and corresponding ranks are grouped by cluster centroid. The TF-IDF and ranks are calculated at the document level, not the cluster level.

Automatic Cluster Title Generation

There is an additional optional stage of the `cluster` extractive summarization method that will create titles for each cluster. This is accomplished by summarizing each cluster twice: once for the actual summary and again to create the title. Since titles are much shorter than the content, a seq2seq transformer trained on the XSum dataset is used (BART is the default). XSum contains documents and one-sentence news summaries answering the question “What is the article about?”. To encourage short titles, when decoding the model output of the cluster we set the minimum length to one token and the maximum to ten tokens. This produces subpar titles but, the structured joined summarization method (see *Structured Joined Summarization*) solves this problem.

11.4.2 The Generic Method

There are six generic extractive summarization methods: `lsa`, `luhn`, `lex_rank`, `text_rank`, `edmundson`, and `random`. `Random` selects sentences from the input document at random.

11.5 Abstractive Summarization

The abstractive summarization stage is applied to the result of the extractive summarization stage. If extractive summarization was disabled then abstractive summarization simply transforms the result of the modifications stage.

This stage of the summarization steps passes the text of the previous step through a seq2seq transformer model and returns the result. Multiple seq2seq models can be used to summarize including [TransformerSum](#), [BART](#), [PEGASUS](#), [T5](#), [PreSumm](#) (how [TransformerSum](#) improves [PreSumm](#)).

PERSPECTIVE CROPPING

To improve the SSA (see *Slide Structure Analysis (SSA)*) and the set of slides shown to the user, the frames classified as `presenter_slide` need to be cropped to only contain the slide. Two main algorithms were created to accomplish this: feature matching and corner crop transform.

12.1 SIFT Matcher & Perspective Cropping

The feature matching algorithm iterates through the `slide` and `presenter_slide` frames in chronological order. When the class switches, the algorithm begins matching using Oriented FAST and Rotated BRIEF (ORB) (which performs the same task as Scale Invariant Feature Transform (SIFT) but at two orders of magnitude the speed) for feature detection/description and Fast Library for Approximate Nearest Neighbors (FLANN) for matching. If the number of matched features is above a threshold then the images are considered to contain the same slide. The algorithm then continues detecting duplicates based on the number of feature matches until another `slide` frame is encountered. The `slide` or `presenter_slide` frame with the most content is kept. If the best frame is a `presenter_slide` then the RANSAC transformation will be used to crop the image to only contain the slide.

Note: The main function is `lecture2notes.end_to_end.sift_matcher.match_features()`

12.1.1 Content Detector

The content detector determines if and how much content is added between two images. The algorithm dilates both images and finds contours. It then computes the total area of those contours. If $\gamma\%$ more than the area of the first image's contours is greater than the area of the second image's contours then it is assumed more content is added. The difference in area is the amount of content added.

Note: The main function is `lecture2notes.end_to_end.sift_matcher.is_content_added()`

12.1.2 Camera Motion Detection

The camera motion detection algorithm detects camera movement between two frames by tracking features along the borders of the image. Only the borders are used because the center of the image will contain a slide. Tracking features of the slide is not robust since those features will disappear when the slide changes. Furthermore, features are not found in the bottom border because `presenter_slide` images may have the peoples' heads at the bottom, which will move and do not represent camera motion. Features are identified using ShiTomasi Corner Detection. The Lucas Kanade optical flow is calculated between consecutive frames and the average distance of all features is the total camera movement. If the camera moves more than 10 pixels, then there is assumed to be camera movement. If the camera doesn't move then the feature matching algorithm will automatically crop each `presenter_slide` frame even if it does not have a matching `slide` frame.

Note: The main function is `lecture2notes.end_to_end.sift_matcher.does_camera_move()`

12.2 Corner Crop Transform

Located on its own page here: *Perspective Cropping & Corner Detection*.

Note: The main function is `lecture2notes.end_to_end.corner_crop_transform.crop()`

DUPLICATE IMAGE REMOVAL

The system uses a variety of methods to remove duplicate slides and obtain a set of unique frames containing the one best representation of each slide in the presentation. One method that is applied at various steps of the procedure (during black border removal, perspective crop, and clustering) is image hashing. Standard hashing algorithms will output completely different hashes on images that differ by one-byte but still depict the same content. Image hashing algorithms produce similar output hashes given similar inputs. The system supports 4 hashing methods: average, perception (the default), difference, and wavelet hashing. These algorithms analyze the image structure based on luminance (without color information). This process will only remove extremely similar images and thus can safely be applied without any false-positives. However, since the presenter moving slightly will cause the algorithm to detect two unique images even though they contain the same slide, we employ clustering (see *Slide Clustering*) and feature matching (see *SIFT Matcher & Perspective Cropping*) algorithms.

FIGURE DETECTION ALGORITHM

The figure extraction algorithm identifies and saves images, charts, tables, and diagrams from slide frames so that they can be shown in the final summary. Two sub-algorithms are used during figure extraction, each of which specializes in identifying particular figures. The `large box detection` algorithm identifies images that have a border, such as tables, by performing canny edge detection, applying a small dilation, and searching for rectangular contours that meet a size threshold. The `large dilation detection` algorithm performs a very large dilation on the canny edge map, computes contours, and finally approximates bounding boxes that meet a size and aspect ratio threshold. This algorithm specializes in locating images without borders since `large box detection` will not detect contours that do not closely resemble rectangles.

Text and color checks are applied as part of the `large dilation detection` algorithm. For each potential figure, the text check calculates the area of text within the figure. Before identifying any figures, the bounding boxes of text within the image are determined using the EAST (Efficient and Accurate Scene Text Detector) text detection algorithm. Then, the overlapping area between the potential figure and each text bounding box is calculated and summed. If the area of the text is lower than a percentage of the total potential figure area, then the check passes. The color check simply checks if the image contains color even if it has red, green, and blue color bands by computing the mean of squared errors.

Finally, two checks are applied to all potential figures, regardless of which algorithm proposed them. The first is an overlapping area check. The overlapping area between every combination of two potential figures is calculated. If one figure overlaps another, then the larger figure is kept since it likely contains the smaller one. The second check ensures the complexity of the figure is above a threshold by calculating Shannon Entropy.

Extracted figures are attached to their respective slide in the slide structure analysis.

You can learn more about figure detection in the API documentation. Please see [`lecture2notes.end_to_end.figure_detection.detect_figures\(\)`](#)

SLIDE STRUCTURE ANALYSIS (SSA)

Note: The main function to perform a slide structure analysis is `lecture2notes.end_to_end.slide_structure_analysis.analyze_structure()`

The SSA algorithm extracts formatted text from each slide using OCR, stroke width, and the height of text bounding boxes. The SSA process identifies the title as well as bold, normal, and small/footer text. First, the algorithm performs OCR using Tesseract which outputs text with metadata such as the block, paragraph, line, and word number. This is used to identify individual bullet points that may exist on the slide. Tesseract also provides the location and size of the bounding box for each identified word, which is used to determine the stroke width of each word. Next, the words are grouped into their respective lines and the text classification algorithm is applied. The result is saved line by line with the text and its predicted category. All Tesseract outputs are spell checked with `SymSpell`, a symmetric delete spelling correction algorithm.

The text classification algorithm identifies a line of text as **bold** if it has above-average stroke width or above-average height. A line will be identified as **footer** text if the stroke width is below average and the height is below average. It is worse to misidentify **normal** text as **footer** text than **footer** text as **normal** text because the former causes a loss of content in the created notes. If a line of text does not meet either of the aforementioned checks, it is classified as **normal**. Both checks (height and stroke width) compare against their respective averages times a scaling factor.

The stroke width algorithm takes an input image, applies Ostu's threshold, computes the distance transformation of the image, identifies peaks in intensity, and returns the average of those peaks.

15.1 SSA Title Identification

The SSA title identification algorithm determines the title of an input slide given the Tesseract OCR output and image data. The first paragraph will be classified as a title if it meets the following criteria:

1. The mean top y coordinate of the text bounding boxes is in the upper third of the image.
2. The mean of the x coordinate of the text bounding boxes is less than the 77% of the image width.
3. The number of characters is greater than 3.
4. The mean stroke width of the paragraph is greater than the mean stroke width of all the content on the slide plus one standard deviation.
5. The mean bounding box height of the paragraph is greater than the mean high of all the content on the slide.

If there is only one block and paragraph then the slide might only contain the title. If this situation occurs, the stroke width and height checks are disabled because the averages of all content will only account for the title if these checks were left enabled.

SLIDE CLUSTERING

Clustering is used to group slides that contain the same content. We implement two main methods of clustering: `normal` algorithms (Affinity Propagation and KMeans) and our novel `segment` approach, which is the default. By grouping slides, the system is capable of choosing the frame that best represents each group.

16.1 Normal Algorithms

When the `normal` mode is selected, if the number of slides is specified by the user KMeans will be used, otherwise Affinity Propagation will be used. Features are extracted from the layer before pooling for all model architectures. KMeans and Affinity Propagation select the frame closest to the centroid.

Relevant Classes:

1. `lecture2notes.end_to_end.cluster.ClusterFilesystem`
2. `lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster`

16.2 Segment Method

The `segment` clustering method iterates chronologically through extracted frames that have been filtered and transformed by the perspective cropping (see Section ref{Perspective Cropping}) algorithm. The algorithm marks splitting points based on large visual differences, which are measured by the cosine similarity between the feature vectors extracted from the slide classifier. A split is marked when the difference between two frames is greater than the mean of the differences across all frames.

Relevant Classes:

1. `lecture2notes.end_to_end.segment_cluster.SegmentCluster`

TRANSCRIBE (SPEECH-TO-TEXT)

API Documentation: [Transcribe](#)

The `transcribe` module contains the features necessary to convert a wave file to text.

There are 6 methods, which rely on 4 libraries, of transcribing audio implemented in this module. Additionally, 2 chunking algorithms are supported.

The mixed performance of [DeepSegment](#) combined with the inaccuracies of current STT software results in subpar performance overall in the automatic creation of transcripts. Ideally, the user will provide a transcript. If not, then summarization methods that use the transcript will be negatively impacted since summarization models rely on correct grammatical structures.

17.1 Chunking

Chunking increases speed of voice-to-text by reducing the amount of audio without speech.

- **Voice Activity:** Uses the WebRTC Voice Activity Detector (VAD) which is “reportedly one of the best VADs available, being fast, modern, and free.” The python bindings to the WebRTC VAD API are provided by [wiseman/py-webrtcvad](#). This algorithm is implemented in the `chunk_by_speech()` function. It produces segments that can be transcribed with `process_segments()` using [DeepSpeech](#), [Vosk](#), or [Wav2Vec2](#).
- **Noise Activity:** Detects and removes segments of audio that are significantly below the average loudness of the file. This algorithm is implemented in `chunk_by_silence()` and it writes chunks (wave files that are parts of the original file that contain noise) to disk. `process_chunks()` runs transcription on every file in a directory. It can be used in conjunction with `chunk_by_silence()` to transcribe all the chunks and return a merged transcript.

Note: Chunking is necessary for the `google` method (see [Transcribing Methods](#)) for long audio files since Google Speech Recognition will time out if the file size is too large.

17.2 Transcribing Methods

Note: Throughout this section `google` refers to “Google Speech Recognition” (free) and not the [Google Cloud Speech API](#) (paid). It is my understanding that “Google Speech Recognition” is deprecated and could disappear anytime. “Google Speech Recognition” was not tested but the “Google Cloud Speech API” was tested so in the results “Google STT” refers to the paid “Google Cloud Speech API.”

The recommended method is Vosk (small model) since it works offline (`google` does not), has fantastic accuracy (1st or 2nd best depending on model used), is actively maintained (`sphinx` and `google` are dated), is completely open source (`google` is proprietary), is the most reliable (`google` frequently times out and `sphinx` is difficult to work with), and doesn’t require a GPU (DeepSpeech does utilize a GPU). To download the recommended model run the commands under 4. *Vosk*.

You can specify the transcription method you like using the `--transcription_method` option with the `end_to_end/main.py` script. Some methods require a model to be loaded. For those methods you should set the `--transcribe_model_dir` to the directory containing the model. For example, the default is to use the Vosk small model so the `--transcription_method` defaults to `vosk` and the `--transcribe_model_dir` defaults to `./vosk-models/vosk-model-en-us-small-0.3/`.

17.2.1 1. YouTube

`lecture2notes.transcribe.transcribe_main.get_youtube_transcript()`. This method only works if the lecture to be summarized is a YouTube video that contains manually added captions.

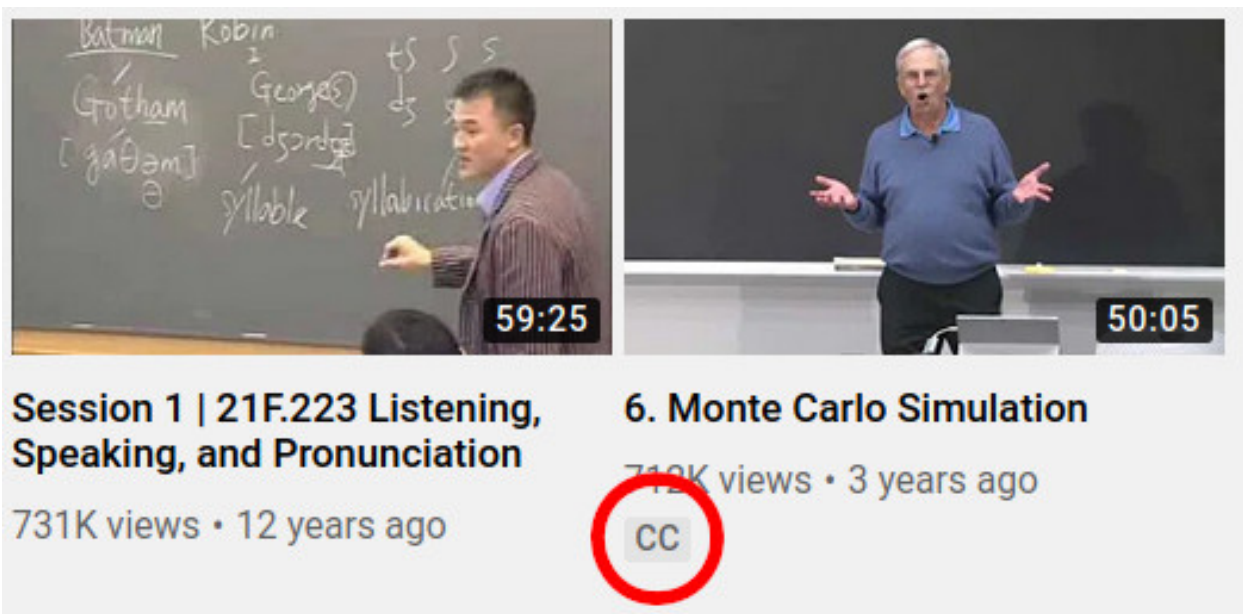


Fig. 1: You can tell if a video contains manual captions if it contains the CC icon as shown above.

This method downloads the transcript for the specified language directly from YouTube using either the YouTube API (`get_transcript_api()`) or `youtube-dl` (`get_transcript_ytdl()`). Both methods are part of the `TranscriptDownloader` class.

The `download()` function provides easy access to both of these download options.

Note: The YouTube API requires an API key. You can find more information about how to obtain a key for free from [Google Developers](#).

Important: Using `youtube-dl` is recommended over the YouTube API because it does not require an API key and is significantly more reliable than the YouTube API.

17.2.2 2. General: Sphinx and Google

The `sphinx` and `google` methods use the [SpeechRecognition library](#) to access `pocketsphinx-python` and Google Speech Recognition, respectively. These methods are grouped together in the `transcribe_audio_generic()` function because the SpeechRecognition library simplifies the differences to one line. The `method` argument allows the switching between both methods.

17.2.3 3. DeepSpeech

The `deepspeech` method uses the [Mozilla DeepSpeech library](#), which achieves very good accuracy on the [LibriSpeech clean test corpus](#) (the current model accuracy can be found on the [latest release page](#)).

The DeepSpeech architecture was created by *Baidu* in 2014. Project DeepSpeech was created by *Mozilla* (the creators of the popular Firefox web browser) to provide the open source community with an updated Speech-To-Text engine.

In order to use this method in the `end_to_end/main.py` script, the latest DeepSpeech model needs to be downloaded (the `.pbmm` acoustic model and the scorer) from the [releases page](#). Mozilla provides code to download and extract the model on the [project's documentation](#). You can rename these files as long as the extensions remain the same. When using the `end_to_end/main.py` script you only have to specify the directory containing both files (the directory name is not important but `deepspeech-models` is descriptive). See [Installation](#) for more details about downloading the deepspeech models.

Example Folder Structure:

```
deepspeech-models/
├── deepspeech-0.7.1-models.pbmm
└── deepspeech-0.7.1-models.scorer
```

17.2.4 4. Vosk

The `vosk` method is implemented in the `transcribe_audio_vosk()` function. It uses the [vosk library](#).

In order to use this method in the `end_to_end/main.py` script, you need to download one of the models from [alphacephei's website](#) at <https://alphacephei.com/vosk/models> ([Google Drive Mirror](#)).

These commands will download the recommended model and put it in the expected location:

```
conda activate lecture2notes
gdown https://drive.google.com/uc?id=1cjgVxc_NJUYapxEJ2xv-t61nKhJhKs-M
unzip vosk-model-en-us-small-0.3.zip
mkdir vosk-models/
mv vosk-model-en-us-small-0.3 ./vosk-models/vosk-model-en-us-small-0.3/
```

17.2.5 5. Wave2Vec2

The Wav2Vec2 model was proposed in [wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations](#) by Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli.

The abstract from the paper is the following:

“We show for the first time that learning powerful representations from speech audio alone followed by fine-tuning on transcribed speech can outperform the best semi-supervised methods while being conceptually simpler. wav2vec 2.0 masks the speech input in the latent space and solves a contrastive task defined over a quantization of the latent representations which are jointly learned. Experiments using all labeled data of Librispeech achieve 1.8/3.3 WER on the clean/other test sets. When lowering the amount of labeled data to one hour, wav2vec 2.0 outperforms the previous state of the art on the 100 hour subset while using 100 times less labeled data. Using just ten minutes of labeled data and pre-training on 53k hours of unlabeled data still achieves 4.8/8.2 WER. This demonstrates the feasibility of speech recognition with limited amounts of labeled data.”

The implementation in lecture2notes uses the [huggingface/transformers](#) package. You can learn more by reading the code for the `transcribe_audio_wav2vec()` function. The wav2vec2 model will automatically be downloading upon usage thanks to huggingface/transformers. The default model key to be downloaded from the [model hub](#) is `facebook/wav2vec2-base-960h`. You can also download the model manually from [this Google Drive mirror](#)

17.3 Experiment Results

We tested the performance of DeepSpeech, Sphinx, several Vosk models, and Google’s paid STT API on 43 lecture videos from the slide classifier dataset. These videos were from MIT OpenCourseWare and had human-written transcripts attached. We used the YouTube API to fetch the transcripts. We then computed the Word Error Rate (WER), Match Error Rate (MER), and Word Information Lost (WIL). The tests were performed on an Intel i7-6700HQ CPU and a Mobile Nvidia GTX-1060-6GB GPU.

Model	WER	MER	WIL	LS TC WER ¹	Processing Time
DeepSpeech (chunking)	43.01	41.82	59.04	5.97	4 hours
DeepSpeech	44.44	42.98	59.99	5.97	20 hours
Google STT (“default” model)	34.43	33.14	49.05	12.23	20 minutes ²
Wav2Vec2	39.38	36.27	54.43	2.60	40 minutes
Sphinx was not tested because it is between 6x-18x slower than other models.					
Vosk small-0.3	35.43	33.64	50.84	15.34	8.5 hours
Vosk daanzu-20200905	33.67	31.87	48.28	7.08	5.5 hours
Vosk aspire-0.2	41.38	38.44	56.35	13.64	19 hours
Vosk aspire-0.2 (chunking)	41.45	38.65	56.56	13.64	19 hours

The WER on our test set of lecture audio recordings is much higher than the LibriSpeech baseline. This is because LibriSpeech contains clean audio recorded by professional microphones while most lecture recordings do not have access to this type of equipment. The Vosk small-0.3 model is the smallest model and the Vosk daanzu-20200905 model is both the fastest and most accurate model. Of the open-source models we tested, DeepSpeech and Wav2Vec2 are only ones that run on the GPU, which restricts the environments in which they can be used. Chunking is a necessity with the DeepSpeech model because it results in a 6.67x improvement in speed and about 1.4 percentage point decrease in WER.

¹ “LS TC WER” stands for LibriSpeech test-clean WER (Word Error Rate).

² Google STT ran in parallel.

Google's STT model was able to perform about 20% better than DeepSpeech (measured by WER) but was still 0.76 percentage points behind Vosk daanzu-20200905. Google's STT model was able to run much faster than any open-source model because multiple files were processed simultaneously. However, Google's services are proprietary and cost money while DeepSpeech, Sphinx, and Vosk are open-source, free, and, in some cases, more accurate. The open-source models can be parallelized, but were not for the sake of simplicity.

Despite its size, the Vosk small-0.3 model is the second most accurate (measured by WER) out of the open-source models. This model ran about 2x faster than both DeepSpeech and the Vosk aspire-0.2` model. The ``Vosk small-0.3 model is only 1.00 and 1.76 percentage points worse than Google's STT service and Vosk daanzu-20200905 respectively. Chunking does not improve the performance or speed of Vosk because it is meant for streaming ASR instead of transcribing an entire audio file.

17.4 Script Descriptions

- **transcribe_main**: Implements transcription using 6 different methods from 4 libraries and other miscellaneous functions related to audio transcription, including audio reading, writing, extraction, and conversion.
- **webrtcvad_utils**: Implements functions to filter out non-voiced sections from audio files. The primary function is `vad_segment_generator()`, which accepts an audio path and returns segments of audio with voice.
- **mic_vad_streaming**: Streams from microphone to DeepSpeech, using Voice Activity Detection (VAD) provided by webrtcvad.

- To select the correct input device, the code below can be used. It will print a list of devices and associated parameters as detected by pyaudio.

```
import pyaudio
p = pyaudio.PyAudio()
for i in range(p.get_device_count()):
    print(p.get_device_info_by_index(i))
```

- Output of `python mic_vad_streaming.py --help`

```
usage: mic_vad_streaming.py [-h] [-v VAD_AGGRESSIVENESS] [--nospinner]
                           [-w SAVEWAV] [-f FILE] -m MODEL [-s SCORER]
                           [-d DEVICE] [-r RATE]

Stream from microphone to DeepSpeech using VAD

optional arguments:
-h, --help            show this help message and exit
-v VAD_AGGRESSIVENESS, --vad_aggressiveness VAD_AGGRESSIVENESS
                        Set aggressiveness of VAD: an integer between 0 and 3,
                        0 being the least aggressive about filtering out non-
                        speech, 3 the most aggressive. Default: 3
--nospinner           Disable spinner
-w SAVEWAV, --savewav SAVEWAV
                        Save .wav files of utterances to given directory
-f FILE, --file FILE  Read from .wav file instead of microphone
-m MODEL, --model MODEL
                        Path to the model (protocol buffer binary file, or
                        entire directory containing all standard-named files
                        for model)
-s SCORER, --scorer SCORER
```

(continues on next page)

(continued from previous page)

```

                                Path to the external scorer file.
-d DEVICE, --device DEVICE      Device input index (Int) as listed by
                                pyaudio.PyAudio.get_device_info_by_index(). If not
                                provided, falls back to PyAudio.get_default_device().
-r RATE, --rate RATE           Input device sample rate. Default: 16000. Your device
                                may require 44100.

```

17.5 DeepSegment

Some STT engines do not add punctuation. To solve this we use the [DeepSegment](#) model to segment sentences. This model restores sentence punctuation by only using the unpunctuated text as the input. A more accurate model may be able to determine punctuation based on the input text and timings of each word. For example, a greater pause between two words may indicate a period. However, since improving STT was not the goal of this research, this was not attempted. DeepSegment achieves a 52.64 absolute accuracy score (number of correctly segmented texts divided by number of examples) on text with no punctuation while Spacy only reaches 11.76 and NLTK Punkt reaches 9.89.

17.6 Other Transcription Options

These are some of my notes on other potential speech-to-text libraries that I never fully integrated into lecture2notes. This information may be outdated, but it is still useful to know about the other options that exist.

17.6.1 ESPnet

[espnet/espnet](#) is extremely promising but is very slow for some reason. The “ASR demo” can be found in the [main README](#).

The ESPnet commands to transcribe a WAV file are:

```

cd egs/librispeech/asr1
. ./path.sh
./../../../../utils/recog_wav.sh --ngpu 1 --models librispeech.transformer.v1 example.wav

```

Installation can be completed with:

```

# OS setup
!cat /etc/os-release
!apt-get install -qq bc tree sox

# espnet setup
!git clone --depth 5 https://github.com/espnet/espnet
!pip install -q torch==1.1
!cd espnet; pip install -q -e .

# download pre-compiled warp-ctc and kaldi tools
!espnet/utils/download_from_google_drive.sh \
    "https://drive.google.com/open?id=13Y4tSygc8WtqzvAVGK_vRV9G1V7TRC0w" espnet/tools_
↪tar.gz > /dev/null

```

(continues on next page)

(continued from previous page)

```
!cd espnet/tools/warp-ctc/pytorch_binding && \  
  pip install -U dist/warpctc_pytorch-0.1.1-cp36-cp36m-linux_x86_64.whl  
  
# make dummy activate  
!mkdir -p espnet/tools/venv/bin && touch espnet/tools/venv/bin/activate  
!echo "setup done."
```

17.6.2 wav2letter

Wav2letter is an “open source speech processing toolkit” written in C++ that is “built to facilitate research in end-to-end models for speech recognition.” It contains pre-trained models, but the state-of-the-art models can not easily be used with the separate inference scripts. They need to be converted. The [inference tutorial](#) is helpful, but it uses a smaller “example model” that does not reach state-of-the-art accuracy.

It is recommended to use wav2letter with docker due to the complex dependency tree.

The [simple_streaming_asr_example](#) script can transcribe a WAV file when it is provided with the models.

The pre-trained SOTA models are [in this folder](#) and are from the “End-to-end ASR: from Supervised to Semi-Supervised Learning with Modern Architectures” paper.

This issue is currently open and discusses the lack of clear instructions about how to use the SOTA models for inference: [Any example code using the new pretrained models](#)

It may be possible to use the [streaming_convnets](#) research models for inference if they are converted using [StreamingTDSModelConverter.cpp](#), which has instruction [in this README](#).

PERSPECTIVE CROPPING & CORNER DETECTION

The corner crop transform algorithm has two steps. First, it will apply Ostu's threshold and extract contours from an edge map of the image. In the edge map, the algorithm attempts to find a large rectangle, which is the slide. This method is ineffective if there are any gaps or obstructions in the outline around the slide. So, if it fails to find the slide, the program will use the Hough Lines algorithm to find horizontal and vertical lines, then find the intersection points, and finally cluster those points using KMeans.

The process this file executes is based on [this jupyter notebook](#) and [this StackOverflow answer](#). The `opencv` package provides most of the low-level functions. See the [Corner Crop Transform API Documentation](#) for details regarding the functions.

Useful guides and tutorials to help understand the script:

- [How To Build a Kick-Ass Mobile Document Scanner in Just 5 Minutes](#)
- [Credit card OCR with OpenCV and Python](#)
- [A Box detection algorithm for any image containing boxes.](#)
- [Sorting Contours using Python and OpenCV](#)
- [Horizontal Line detection with OpenCV](#)
- [Text skew correction with OpenCV and Python](#)

18.1 Debug/Testing Mode

The file can be run as a script in order to try the image processing pipeline on several files, view the results, and tweak the parameters as necessary to improve accuracy.

Recommended Command:

```
python corner_crop_transform.py folder ../presenter_slide/ --debug_mode --debug_gif --  
↪debug_path ../debug_imgs
```

The above command processes all files in the `../presenter_slide/` folder. The `--debug_mode` argument allows the usage of `--debug_gif` and `--debug_path`. `--debug_gif` will save a gif of each step of the pipeline with a 1.4s delay to `--debug_path`. `--debug_imgs` can also be enabled to save each step of the pipeline as its own image (in higher quality than the GIF, the GIF only has 256 colors).

Using `--debug_mode` will output a file in the present working directory named `debug_crop_error_log.txt` with the paths to all of the images that the pipeline thinks it failed with.

Warning: There may be other images that the script failed to process correctly. The `debug_crop_error_log.txt` file only contains the paths to images that did not meet the criteria to be perspective cropped. The reasoning behind why they didn't will likely lead to parameter tuning.

You can also process a single file using the file as a script with the `file` mode:

```
python corner_crop_transform.py file /path/to/image/file/0Q5IimX-AAc-img_067.jpg -d -di -
↪ dg
```

18.2 Example Images

18.2.1 Example of contours mode failing and falling back to `hough_lines`

Command used to create above image:

```
python corner_crop_transform.py file ../dataset/classifier-data/presenter_slide/
↪ IJquEYhiq_U-img_130_debug.gif -d -di -dg -dgo
```

Timeline of images (click image for larger view):

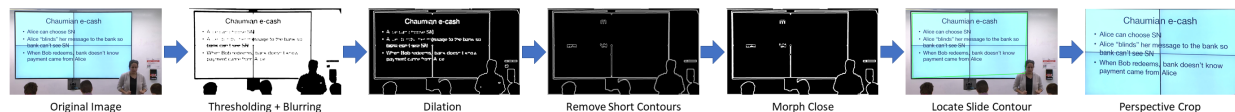


18.2.2 Example of contours mode succeeding

Command used to create above image:

```
python corner_crop_transform.py file ../dataset/classifier-data/presenter_slide/
↪ IJquEYhiq_U-img_068.jpg -d -di -dg -dgo
```

Timeline of images (click image for larger view):



18.3 Script Help

Output of `python corner_crop_transform.py --help`:

```
usage: corner_crop_transform.py [-h] [-d] [-di] [-dg] [-dgo] [-p DEBUG_PATH]
                               [-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}]
                               {file,folder} path

Perspective Crop to Rectangles (Slides)

positional arguments:
{file,folder}          `file` mode will crop a single image and `folder` mode
                        will crop all the images in a given folder
path                   path to file or folder (depending on `mode`) to
                        process

optional arguments:
-h, --help              show this help message and exit
-d, --debug_mode        enable the usage of `--debug_imgs`, `--debug_gif`, and
                        `--debug_path`.
-di, --debug_imgs       Save debug images (JPG of each step of the pipeline).
                        Requires `--debug_mode` to be enabled.
-dg, --debug_gif        Save debug gif (GIF with 1.4s delay between each debug
                        image). Requires `--debug_mode` to be enabled.
-dgo, --debug_gif_optimize
                        Optimize the gif produced by enabling --debug_gif with
                        `gifsicle`.
-p DEBUG_PATH, --debug_path DEBUG_PATH
                        path to folder to store debug images (default:
                        `./debug_imgs`)
-l {DEBUG,INFO,WARNING,ERROR,CRITICAL}, --log {DEBUG,INFO,WARNING,ERROR,CRITICAL}
                        Set the logging level (default: 'Info').
```


19.1 Main Summarizer Class

```
class lecture2notes.end_to_end.summarizer_class.LectureSummarizer(params,
                                                                    **kwargs)

    determine_root_path()
    run_all()
    step_black_border_removal()
    step_classify_slides()
    step_cluster_slides()
    step_extract_figures()
    step_extract_frames()
    step_perspective_crop()
    step_slide_structure_analysis()
    step_summarize()
    step_transcribe_audio()

lecture2notes.end_to_end.summarizer_class.time_this(f)
```

19.2 Transcribe

19.2.1 transcribe_main

```
lecture2notes.end_to_end.transcribe.transcribe_main.caption_file_to_string(transcript_path,
                                                                              re-
                                                                              move_speakers=False)
```

Converts a .srt, .vtt, or .sbv file saved at `transcript_path` to a python string. Optionally removes speaker entries by removing everything before “: ” in each subtitle cell.

```
lecture2notes.end_to_end.transcribe.transcribe_main.check_transcript(generated_transcript,
                                                                      ground_truth_transcript)
```

Compares `generated_transcript` to `ground_truth_transcript` to check for accuracy using spacy similarity measurement. Requires the “en_vectors_web_lg” model to use “real” word vectors.

```
lecture2notes.end_to_end.transcribe.transcribe_main.chunk_by_silence(audio_path,  
                                                                    output_path,  
                                                                    si-  
                                                                    lence_thresh_offset=5,  
                                                                    min_silence_len=2000)
```

Split an audio file into chunks on areas of silence

Parameters

- **audio_path** (*str*) – path to a wave file
- **output_path** (*str*) – path to a folder where wave file chunks will be saved
- **silence_thresh_offset** (*int*, *optional*) – a value subtracted from the mean dB volume of the file. Default is 5.
- **min_silence_len** (*int*, *optional*) – the length in milliseconds in which there must be no sound in order to be marked as a splitting point. Default is 2000.

```
lecture2notes.end_to_end.transcribe.transcribe_main.chunk_by_speech(audio_path,  
                                                                    out-  
                                                                    put_path=None,  
                                                                    aggressive-  
                                                                    ness=1,  
                                                                    de-  
                                                                    sired_sample_rate=None)
```

Uses the python interface to the WebRTC Voice Activity Detector (VAD) API to create chunks of audio that contain voice. The VAD that Google developed for the WebRTC project is reportedly one of the best available, being fast, modern and free.

Parameters

- **audio_path** (*str*) – path to the audio file to process
- **output_path** (*str*, *optional*) – path to save the chunk files. if not specified then no wave files will be written to disk and the raw pcm data will be returned. Defaults to None.
- **aggressiveness** (*int*, *optional*) – determines how aggressive filtering out non-speech is. must be an interger between 0 and 3. Defaults to 1.
- **desired_sample_rate** (*int*, *optional*) – the sample rate of the returned segments. the default is the same rate of the input audio file. Defaults to None.

Returns (segments, sample_rate, audio_length). See [vad_segment_generator\(\)](#).

Return type tuple

```
lecture2notes.end_to_end.transcribe.transcribe_main.convert_deepspeech_json(transcript_json)
```

Convert a deepspeech json transcript from a letter-by-letter format to word-by-word.

Parameters **transcript_json** (*dict* or *str*) – The json format transcript as a dictionary or a json string, which will be loaded using `json.loads()`.

Returns The word-by-word transcript json.

Return type dict

```
lecture2notes.end_to_end.transcribe.transcribe_main.convert_samplerate(audio_path,  
                                                                    de-  
                                                                    sired_sample_rate)
```

Use *SoX* to resample wave files to 16 bits, 1 channel, and `desired_sample_rate` sample rate.

Parameters

- **audio_path** (*str*) – path to wave file to process
- **desired_sample_rate** (*int*) – sample rate in hertz to convert the wave file to

Returns

(**desired_sample_rate**, **output**) where **desired_sample_rate** is the new sample rate and **output** is the newly resampled pcm data

Return type tuple

```
lecture2notes.end_to_end.transcribe.transcribe_main.extract_audio(video_path,  
                                                                    output_path)
```

Extracts audio from video at video_path and saves it to output_path

```
lecture2notes.end_to_end.transcribe.transcribe_main.get_youtube_transcript(video_id,  
                                                                    out-  
                                                                    put_path,  
                                                                    use_youtube_dl=True)
```

Downloads the transcript for video_id and saves it to output_path

```
lecture2notes.end_to_end.transcribe.transcribe_main.load_deepspeech_model(model_dir,  
                                                                    beam_width=500,  
                                                                    lm_alpha=None,  
                                                                    lm_beta=None)
```

Load the deepspeech model from model_dir

Parameters

- **model_dir** (*str*) – path to folder containing the “.pbmm” and optionally “.scorer” files
- **beam_width** (*int*, *optional*) – beam width for decoding. Default is 500.
- **(float (lm_alpha)** – alpha parameter of language model. Default is None.
- **optional}** – alpha parameter of language model. Default is None.
- **lm_beta** (*float*, *optional*) – beta parameter of language model. Default is None.

Returns the loaded deepspeech model

Return type deepspeech.Model

```
lecture2notes.end_to_end.transcribe.transcribe_main.load_model(method, *args,  
                                                                **kwargs)
```

```
lecture2notes.end_to_end.transcribe.transcribe_main.load_vosk_model(model_dir)
```

```
lecture2notes.end_to_end.transcribe.transcribe_main.load_wav2vec_model(model='facebook/wav2vec2-  
                                                                    base-960h',  
                                                                    tokenizer='facebook/wav2vec2-  
                                                                    base-960h',  
                                                                    **kwargs)
```

```
lecture2notes.end_to_end.transcribe.transcribe_main.metadata_to_json(candidate_transcript)
```

Helper function to convert metadata tokens from deepspeech to a dictionary.

```
lecture2notes.end_to_end.transcribe.transcribe_main.metadata_to_list(candidate_transcript)
```

```
lecture2notes.end_to_end.transcribe.transcribe_main.metadata_to_string(metadata)
```

Helper function to convert metadata tokens from deepspeech to a string.

```
lecture2notes.end_to_end.transcribe.transcribe_main.process_chunks(chunk_dir,  
                                                                    method='sphinx',  
                                                                    model_dir=None)
```

Performs transcription on every noise activity chunk (audio file) created by [chunk_by_silence\(\)](#) in a directory.

```
lecture2notes.end_to_end.transcribe.transcribe_main.process_segments(segments,  
                                                                    model, au-  
                                                                    dio_length='unknown',  
                                                                    method='deepspeech',  
                                                                    do_segment_sentences=True)
```

Transcribe a list of byte strings containing pcm data

Parameters

- **segments** (*list*) – list of byte strings containing pcm data (generated by [chunk_by_speech\(\)](#))
- **model** (*deepspeech model*) – a deepspeech model object or a path to a folder containing the model files (see [load_deepspeech_model\(\)](#)).
- **audio_length** (*str, optional*) – the length of the audio file if known (used for logging statements) Default is “unknown”.
- **method** (*str, optional*) – The model to use to perform speech-to-text. Supports ‘deepspeech’ and ‘vosk’. Defaults to “deepspeech”.
- **do_segment_sentences** (*bool, optional*) – Find sentence boundaries using [segment_sentences\(\)](#). Defaults to True.

Returns (*full_transcript, full_transcript_json*) The combined transcript of all the items in *segments* as a string and as dictionary/json.

Return type tuple

```
lecture2notes.end_to_end.transcribe.transcribe_main.read_wave(path, de-  
                                                                    sired_sample_rate=None,  
                                                                    force=False)
```

Reads a “.wav” file and converts to *desired_sample_rate* with one channel.

Parameters

- **path** (*str*) – path to wave file to load
- **desired_sample_rate** (*int, optional*) – resample the loaded pcm data from the wave file to this sample rate. Default is None, no resampling.
- **force** (*bool, optional*) – Force the audio to be converted even if it is detected to meet the necessary criteria.

Returns (PCM audio data, sample rate, duration)

Return type tuple

```
lecture2notes.end_to_end.transcribe.transcribe_main.resolve_deepspeech_models(dir_name)
```

Resolve directory path for deepspeech models and fetch each of them.

Parameters **dir_name** (*str*) – Path to the directory containing pre-trained models

Returns a tuple containing each of the model files (pb, scorer)

Return type tuple

```
lecture2notes.end_to_end.transcribe.transcribe_main.segment_sentences(text,
                                                                    text_json=None,
                                                                    do_capitalization=True)
```

Detect sentence boundaries without punctuation or capitalization.

Parameters

- **text** (*str*) – The string to segment by sentence.
- **text_json** (*str or dict, optional*) – If the detected sentence boundaries should be applied to the JSON format of a transcript. Defaults to None.
- **do_capitalization** (*bool, optional*) – If the first letter of each detected sentence should be capitalized. Defaults to True.

Returns The punctuated (and optionally capitalized) string

Return type `str`

```
lecture2notes.end_to_end.transcribe.transcribe_main.transcribe_audio(audio_path,
                                                                    method='sphinx',
                                                                    **kwargs)
```

Transcribe audio using DeepSpeech, Vosk, or a method offered by [transcribe_audio_generic\(\)](#).

Parameters

- **audio_path** (*str*) – Path to the audio file to transcribe.
- **method** (*str, optional*) – The method to use for transcription. Defaults to “sphinx”.
- ****kwargs** – Passed to the transcription function.

Returns (transcript_text, transcript_json)

Return type `tuple`

```
lecture2notes.end_to_end.transcribe.transcribe_main.transcribe_audio_deepspeech(audio_path_or_data,
                                                                                model,
                                                                                raw_audio_data=False,
                                                                                json_num_transcripts=1,
                                                                                **kwargs)
```

Transcribe an audio file or pcm data with the deepspeech model

Parameters

- **audio_path_or_data** (*str or byte string*) – a path to a wave file or a byte string containing pcm data from a wave file. set `raw_audio_data` to True if pcm data is used.
- **model** (*deepspeech model or str*) – a deepspeech model object or a path to a folder containing the model files (see [load_deepspeech_model\(\)](#))
- **raw_audio_data** (*bool, optional*) – must be True if `audio_path_or_data` is raw pcm data. Defaults to False.
- **json_num_transcripts** (*str, optional*) – Specify this value to generate multiple transcripts in json format.

Returns (transcript_text, transcript_json) the transcribed audio file in string format and the transcript in json

Return type `tuple`

```
lecture2notes.end_to_end.transcribe.transcribe_main.transcribe_audio_generic(audio_path,  
                                                                              method='sphinx',  
                                                                              **kwargs)
```

Transcribe an audio file using CMU Sphinx or Google through the speech_recognition library

Parameters

- **audio_path** (*str*) – audio file path
- **method** (*str*, *optional*) – which service to use for transcription (“google” or “sphinx”). Default is “sphinx”.

Returns the transcript of the audio file

Return type *str*

```
lecture2notes.end_to_end.transcribe.transcribe_main.transcribe_audio_vosk(audio_path_or_chunks,  
                                                                            model='./vosk_models',  
                                                                            chunks=False,  
                                                                            de-  
sired_sample_rate=16000,  
                                                                            chunk_size=2000,  
                                                                            **kwargs)
```

Transcribe audio using a vosk model.

Parameters

- **audio_path_or_chunks** (*str or generator*) – Path to an audio file or a generator of chunks created by [chunk_by_speech\(\)](#)
- **model** (*str or vosk.Model*, *optional*) – Path to the directory containing the vosk models or loaded *vosk.Model*. Defaults to “../vosk_models”.
- **chunks** (*bool*, *optional*) – If the *audio_path_or_chunks* is chunks. Defaults to False.
- **desired_sample_rate** (*int*, *optional*) – The sample rate that the model requires to convert audio to. Defaults to 16000.
- **chunk_size** (*int*, *optional*) – The number of wave frames per loop. Amount of audio data transcribed at a time. Defaults to 2000.

Returns (text_transcript, results_json) The transcript as a string and as JSON.

Return type *tuple*

```
lecture2notes.end_to_end.transcribe.transcribe_main.transcribe_audio_wav2vec(audio_path_or_chunks,  
                                                                                model=None,  
                                                                                chunks=False,  
                                                                                de-  
sired_sample_rate=16000)
```

```
lecture2notes.end_to_end.transcribe.transcribe_main.write_to_file(transcript, tran-  
script_save_file,  
tran-  
script_json=None,  
tran-  
script_json_save_path=None)
```

Write transcript to transcript_save_file and transcript_json to transcript_json_save_path.


```
lecture2notes.end_to_end.transcribe.transcribe_main.write_wave(path, audio,  
                                                             sample_rate)
```

Writes a .wav file.

Takes path, PCM audio data, and sample rate.

19.2.2 webrtcvad_utils

```
class lecture2notes.end_to_end.transcribe.webrtcvad_utils.Frame(bytes, timestamp,  
                                                                duration)
```

Represents a “frame” of audio data.

```
lecture2notes.end_to_end.transcribe.webrtcvad_utils.frame_generator(frame_duration_ms,  
                                                                    audio,  
                                                                    sample_rate)
```

Generates audio frames from PCM audio data.

Takes the desired frame duration in milliseconds, the PCM data, and the sample rate.

Yields Frames of the requested duration.

```
lecture2notes.end_to_end.transcribe.webrtcvad_utils.vad_collector(sample_rate,  
                                                                    frame_duration_ms,  
                                                                    padding_duration_ms,  
                                                                    vad, frames)
```

Filters out non-voiced audio frames.

Given a webrtcvad.Vad and a source of audio frames, yields only the voiced audio.

Uses a padded, sliding window algorithm over the audio frames. When more than 90% of the frames in the window are voiced (as reported by the VAD), the collector triggers and begins yielding audio frames. Then the collector waits until 90% of the frames in the window are unvoiced to detrigger.

The window is padded at the front and back to provide a small amount of silence or the beginnings/endings of speech around the voiced frames.

Parameters

- **sample_rate** – The audio sample rate, in Hz.
- **frame_duration_ms** – The frame duration in milliseconds.
- **padding_duration_ms** – The amount to pad the window, in milliseconds.
- **vad** – An instance of webrtcvad.Vad.
- **frames** – a source of audio frames (sequence or generator).

Returns A generator that yields PCM audio data.

Return type [generator]

```
lecture2notes.end_to_end.transcribe.webrtcvad_utils.vad_segment_generator(wavFile,  
                                                                           aggres-  
                                                                           sive-  
                                                                           ness,  
                                                                           de-  
                                                                           sired_sample_rate=None)
```

Generate VAD segments. Filters out non-voiced audio frames.

Parameters **waveFile** (*str*) – Path to input wav file to run VAD on.

Returns

segments: a bytearray of multiple smaller audio frames (The longer audio split into multiple smaller one's)

sample_rate: Sample rate of the input audio file

audio_length: Duration of the input audio file

Return type [tuple]

19.2.3 mic_vad_streaming

```
class lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio(callback=None,  
                                                                    device=None, in-  
                                                                    put_rate=16000,  
                                                                    file=None)
```

Streams raw audio from microphone. Data is received in a separate thread, and stored in a buffer, to be read from.

BLOCKS_PER_SECOND = 50

CHANNELS = 1

FORMAT = 8

RATE_PROCESS = 16000

destroy()

property frame_duration_ms

read()

Return a block of audio data, blocking if necessary.

read_resampled()

Return a block of audio data resampled to 16000hz, blocking if necessary.

resample(data, input_rate)

Microphone may not support our native processing sampling rate, so resample from input_rate to RATE_PROCESS here for webrtcvad and deepspeech

Parameters

- **data** (*binary*) – Input audio stream
- **input_rate** (*int*) – Input audio rate to resample from

write_wav(filename, data)

```
class lecture2notes.end_to_end.transcribe.mic_vad_streaming.VADAudio(aggresiveness=3,  
                                                                    device=None,  
                                                                    in-  
                                                                    put_rate=None,  
                                                                    file=None)
```

Filter & segment audio with voice activity detection.

frame_generator()

Generator that yields all audio frames from microphone.

vad_collector(padding_ms=300, ratio=0.75, frames=None)

Generator that yields series of consecutive audio frames comprising each utterance, separated by yielding a single None. Determines voice activity by ratio of frames in padding_ms. Uses a buffer to include padding_ms prior to being triggered.

Example: (frame, ..., frame, None, frame, ..., frame, None, ...)
 |---utterance---| |---utterance---|

```
lecture2notes.end_to_end.transcribe.mic_vad_streaming.main(ARGS)
```

19.3 Cluster

```
class lecture2notes.end_to_end.cluster.ClusterFilesystem(slides_dir,
                                                         algorithm_name='kmeans',
                                                         num_centroids=20,
                                                         preference=None,
                                                         damping=0.5, max_iter=200,
                                                         model_path='model_best.ckpt')
```

Clusters images from a directory and saves them to disk in folders corresponding to each centroid.

```
extract_and_add_features(copy=True)
```

Extracts features from the images in *slides_dir* and saves feature vectors with `super().add()`

```
transfer_to_filesystem(copy=True, create_best_samples_folder=True)
```

Uses *move_list* from `super()` to take all images in directory *slides_dir* and save each cluster to a subfolder in *cluster_dir* (directory in parent of *slides_dir*)

19.4 Corner Crop Transform

```
lecture2notes.end_to_end.corner_crop_transform.all_in_folder(path,
                                                            remove_original=False,
                                                            **kwargs)
```

Perform perspective cropping on every file in folder and return new paths. ****kwargs** is passed to `crop()`.

```
lecture2notes.end_to_end.corner_crop_transform.cluster_points(points, nclusters)
```

Perform KMeans clustering (using `cv2.kmeans`) on *points*, creating *nclusters* clusters. Returns the centroids of the clusters.

```
lecture2notes.end_to_end.corner_crop_transform.contour_offset(cnt, offset)
```

Offset contour because of 5px border

```
lecture2notes.end_to_end.corner_crop_transform.crop(img_path, output_path=None,
                                                    mode='automatic',
                                                    debug_output_imgs=False,
                                                    save_debug_imgs=False,
                                                    create_debug_gif=False,
                                                    debug_gif_optimize=True,
                                                    debug_path='debug_imgs')
```

Main method to perspective crop an image to the slide.

Parameters

- **img_path** (*str*) – path to the image to load
- **output_path** (*str*, *optional*) – path to save the image. Defaults to `[filename]_cropped.[ext]`.
- **mode** (*str*, *optional*) – There are three modes available. Defaults to “automatic”.

- **contours**: uses `find_page_contours()` to extract contours from an edge map of the image. is ineffective if there are any gaps or obstructions in the outline around the slide.
- **hough_lines**: uses `hough_lines_corners()` to get corners by looking for horizontal and vertical lines, finding the intersection points, and clustering the intersection points.
- **automatic**: tries to use **contours** and falls back to **hough_lines** if **contours** reports a failure.
- **debug_output_imgs** (*bool or dict, optional*) – if dictionary, modifies the dictionary by adding (image file name, image data) pairs. if boolean and True, creates a dictionary in the same way as if a dictionary was passed. Defaults to False.
- **save_debug_imgs** (*bool, optional*) – uses `write_debug_imgs()` to save the debug_output_imgs to disk. Requires debug_output_imgs to not be False. Defaults to False.
- **create_debug_gif** (*bool, optional*) – create a gif of the debug images. Requires debug_output_imgs to not be False. Defaults to False.
- **debug_gif_optimize** (*bool, optional*) – optimize the gif produced by enabling the create_debug_gif option using pygifsicle. Defaults to True.
- **debug_path** (*str, optional*) – location to save the debug images and debug gif. Defaults to “debug_imgs”.

Returns path to cropped image and failed (True if no slide bounding box found, false otherwise)

Return type [tuple]

`lecture2notes.end_to_end.corner_crop_transform.edges_det(img, min_val, max_val, debug_output_imgs=None)`

Preprocessing (gray, thresh, filter, border) & Canny edge detection

Parameters

- **img** (*image*) – the image loaded using `cv2.imread`.
- **min_val** (*int*) – minimum value for `cv2.Canny`.
- **max_val** (*int*) – maximum value for `cv2.Canny`.
- **debug_output_imgs** (*dict, optional*) – modifies this dictionary by adding (image file name, image data) pairs. Defaults to None.

Returns (dilated, total_border), dialted edges and total border width added

Return type [tuple]

`lecture2notes.end_to_end.corner_crop_transform.find_intersection(line1, line2)`

Find the intersection between line1 and line2.

`lecture2notes.end_to_end.corner_crop_transform.find_page_contours(edges, img, border_size=11, min_area_mult=0.3, debug_output_imgs=None)`

Find corner points of page contour

Parameters

- **edges** (*image*) – edges extracted from *img* by [edges_det\(\)](#).
- **img** (*image*) – the image loaded by `cv2.imread`.
- **border_size** (*int*, *optional*) – the size of the borders added by [edges_det\(\)](#). Defaults to 11.
- **min_area_mult** (*float*, *optional*) – the minimum percentage of the image area that a contour's area must be greater than to be considered as the slide. Defaults to 0.5.

Returns

contour is the set of coordinates of the corners sorted by [four_corners_sort\(\)](#) or returns None when no contour meets the criteria.

Return type [contour or NoneType]

`lecture2notes.end_to_end.corner_crop_transform.four_corners_sort(pts)`

Sort corners: top-left, bot-left, bot-right, top-right

`lecture2notes.end_to_end.corner_crop_transform.horizontal_vertical_edges_det(img, thresh_blurred, debug_output_imgs=None)`

Detects horizontal and vertical edges and merges them together.

Parameters

- **img** (*image*) – the image as provided by `cv2.imread`
- **thresh_blurred** (*image*) – the image processed by thresholding. see [edges_det\(\)](#).
- **debug_output_imgs** (*dict*, *optional*) – modifies this dictionary by adding (image file name, image data) pairs. Defaults to None.

Returns result image with a black background and white edges

Return type [image]

`lecture2notes.end_to_end.corner_crop_transform.hough_lines_corners(img, edges_img, min_line_length, border_size=11, debug_output_imgs=None)`

Uses `cv2.HoughLinesP` to find horizontal and vertical lines, finds the intersection points, and finally clusters those points using KMeans.

Parameters

- **img** (*image*) – the image as loaded by `cv2.imread`.
- **edges_img** (*image*) – edges extracted from *img* by [edges_det\(\)](#).
- **min_line_length** (*int*) – the shortest line length to consider as a valid line
- **border_size** (*int*, *optional*) – the size of the borders added by [edges_det\(\)](#). Defaults to 11.
- **debug_output_imgs** (*dict*, *optional*) – modifies this dictionary by adding (image file name, image data) pairs. Defaults to None.

Returns The corner coordinates as sorted by `four_corners_sort()`.

Return type [list]

`lecture2notes.end_to_end.corner_crop_transform.persp_transform(img, s_points)`

Transform perspective of `img` from start points to target points.

`lecture2notes.end_to_end.corner_crop_transform.remove_contours(edges, contour_removal_threshold)`

Remove contours from an edge map by deleting contours shorter than `contour_removal_threshold`.

`lecture2notes.end_to_end.corner_crop_transform.resize(img, height=800, always=False)`

Resize image to given height.

`lecture2notes.end_to_end.corner_crop_transform.segment_lines(lines, delta)`

Groups lines from `cv2.HoughLinesP` into vertical and horizontal bins.

Parameters

- **lines** (*list*) – the data returned from `cv2.HoughLinesP`
- **delta** (*int*) – how far away the x and y coordinates can differ before they're marked as different lines

Returns (`h_lines`, `v_lines`) the horizontal and vertical lines, respectively. each line in each list is formatted as (`x1`, `y1`, `x2`, `y2`).

Return type [tuple]

`lecture2notes.end_to_end.corner_crop_transform.straight_lines_in_contour(contour, delta=100)`

Returns True if `contour` contains lines that are horizontal or vertical. `delta` allows the lines to tilt by a certain number of pixels. For instance, if a line is vertical, its y values can change by `delta` pixels before it is considered not vertical.

`lecture2notes.end_to_end.corner_crop_transform.write_debug_imgs(debug_output_imgs, base_path='debug_imgs')`

Saves images from `debug_output_imgs` to disk in `base_path`.

Parameters

- **debug_output_imgs** (*dict*) – dictionary in format {image file name: image data}
- **base_path** (*str*, *optional*) – the directory to store the debug images. Defaults to “debug_imgs”.

19.5 Text Detection

`lecture2notes.end_to_end.text_detection.get_text_bounding_boxes(image, net, min_confidence=0.5, resized_width=320, resized_height=320)`

Determine the locations of text in an image.

Parameters

- **image** (*np.array*) – The image to be processed.
- **net** (*cv2.dnn_Net*) – The EAST model loaded with `load_east()`.

- **min_confidence** (*float, optional*) – Minimum probability required to inspect a region. Defaults to 0.5.
- **resized_width** (*int, optional*) – Resized image width (should be multiple of 32). Defaults to 320.
- **resized_height** (*int, optional*) – Resized image height (should be multiple of 32). Defaults to 320.

Returns The coordinates of bounding boxes containing text.

Return type list

`lecture2notes.end_to_end.text_detection.load_east(east_path='frozen_east_text_detection.pb')`
Load the pre-trained EAST model.

Parameters **east_path** (*str, optional*) – Path to the EAST model file. Defaults to “frozen_east_text_detection.pb”.

19.6 Figure Detection

`lecture2notes.end_to_end.figure_detection.add_figures_to_ssa(ssa, figures_path)`
`lecture2notes.end_to_end.figure_detection.all_in_folder(path, remove_original=False, east='frozen_east_text_detection.pb', do_text_check=True, **kwargs)`

Perform figure detection on every file in folder and return new paths. ****kwargs** is passed to `detect_figures()`.

`lecture2notes.end_to_end.figure_detection.area_of_corner_box(box)`

`lecture2notes.end_to_end.figure_detection.area_of_overlapping_rectangles(a, b)`
Find the overlapping area of two rectangles a and b. Inspired by <https://stackoverflow.com/a/27162334>.

`lecture2notes.end_to_end.figure_detection.convert_coords_to_corners(box)`

`lecture2notes.end_to_end.figure_detection.detect_color_image(image, thumb_size=40, MSE_cutoff=22, adjust_color_bias=True)`

Detect if an image contains color, is black and white, or is grayscale. Based on [this StackOverflow answer](#).

Parameters

- **image** (*np.array*) – Input image
- **thumb_size** (*int, optional*) – Resize image to this size to speed up calculation. Defaults to 40.
- **MSE_cutoff** (*int, optional*) – A larger value requires more color for an image to be labeled as “color”. Defaults to 22.
- **adjust_color_bias** (*bool, optional*) – Mean color bias adjustment, which improves the prediction. Defaults to True.

Returns Either “grayscale”, “color”, “b&w” (black and white), or “unknown”.

Return type str

```
lecture2notes.end_to_end.figure_detection.detect_figures(image_path,  
                                                         output_path=None,  
                                                         east='frozen_east_text_detection.pb',  
                                                         text_area_overlap_threshold=0.32,  
                                                         figure_max_area_percentage=0.6,  
                                                         text_max_area_percentage=0.3,  
                                                         large_box_detection=True,  
                                                         do_color_check=True,  
                                                         do_text_check=True,  
                                                         entropy_check=2.5,  
                                                         do_remove_subfigures=True,  
                                                         do_rlsa=False)
```

Detect figures located in a slide.

Parameters

- **image_path** (*str*) – Path to the image to process.
- **output_path** (*str, optional*) – Path to save the figures. Defaults to `[filename]_figure_[index].[ext]`.
- **east** (*str or cv2.dnn_Net, optional*) – Path to the EAST model file or the pre-trained EAST model loaded with `load_east()`. `do_text_check` must be true for this option to take effect. Defaults to “frozen_east_text_detection.pb”.
- **text_area_overlap_threshold** (*float, optional*) – The percentage of the figure that can contain text. If the area of the text in the figure is greater than this value, the figure is discarded. `do_text_check` must be true for this option to take effect. Defaults to 0.10.
- **figure_max_area_percentage** (*float, optional*) – The maximum percentage of the area of the original image that a figure can take up. If the figure uses more area than `original_image_area*figure_max_area_percentage` then the figure will be discarded. Defaults to 0.70.
- **text_max_area_percentage** (*float, optional*) – The maximum percentage of the area of the original image that a block of text (as identified by the EAST model) can take up. If the text block uses more area than `original_image_area*text_max_area_percentage` then that text block will be ignored. `do_text_check` must be true for this option to take effect. Defaults to 0.30.
- **large_box_detection** (*bool, optional*) – Detect edges and classify large rectangles as figures. This will ignore `do_color_check` and `do_text_check`. This is useful for finding tables for example. Defaults to True.
- **do_color_check** (*bool, optional*) – Check that potential figures contain color. This helps to remove large quantities of black and white text from the potential figure list. Defaults to True.
- **do_text_check** (*bool, optional*) – Check that only `text_area_overlap_threshold` of potential figures contains text. This is useful to remove blocks of text that are mistakenly classified as figures. Checking for text increases processing time so be careful if processing a large number of files. Defaults to True.
- **entropy_check** (*float, optional*) – Check that the entropy of all potential figures is above this value. Figures with a `shannon_entropy` lower than this value will

be removed. Set to False to disable this check. The `shannon_entropy` implementation is from `skimage.measure.entropy`. IMPORTANT: This check applies to both the regular tests *and* `large_box_detection`, which most check do not apply to. Defaults to 3.5.

- **do_remove_subfigures** (*bool, optional*) – Check that there are no overlapping figures. If an overlapping figure is detected, the smaller figure will be deleted. This is useful to have enabled when using `large_box_detection` since `large_box_detection` will commonly mistakenly detect subfigures. Defaults to True.
- **do_rlsa** (*bool, optional*) – Use RLSA (Run Length Smoothing Algorithm) instead of dilation. Does not apply to `large_box_detection`. Defaults to False.

Returns (figures, output_paths) A list of figures extracted from the input slide image and a list of paths to those figures on disk.

Return type tuple

19.7 Frames Extractor

```
lecture2notes.end_to_end.frames_extractor.extract_frames(input_video_path, quality,
                                                         output_path,
                                                         extract_every_x_seconds)
```

Extracts frames from *input_video_path* at quality level *quality* (best quality is 2) every *extract_every_x_seconds seconds* and saves them to *output_path*

19.8 Helpers

```
lecture2notes.end_to_end.helpers.copy_all(list_path_files, output_dir, move=False)
```

Copy (or move) every path in *list_path_files* if list or all files in a path if path to *output_dir*

```
lecture2notes.end_to_end.helpers.frame_number_filename_mapping(path, file-
                                                                names_only=True)
```

```
lecture2notes.end_to_end.helpers.frame_number_from_filename(filename)
```

```
lecture2notes.end_to_end.helpers.gen_unique_id(input_data, k)
```

Returns the first *k* characters of the sha1 of *input_data*

```
lecture2notes.end_to_end.helpers.make_dir_if_not_exist(path)
```

Makes directory *path* if it does not exist

19.9 Image Hash

```
lecture2notes.end_to_end.imghash.get_hash_func(hashmethod='phash')
```

Returns a hash function from the `imagehash` library.

Hash Methods:

- ahash: Average hash
- phash: Perceptual hash
- dhash: Difference hash

- `whash-haar`: Haar wavelet hash
- `whash-db4`: Daubechies wavelet hash

`lecture2notes.end_to_end.imghash.remove_duplicates(img_dir, images)`

Remove duplicate frames/slides from disk.

Parameters

- **`img_dir`** (*str*) – path to directory containing image files
- **`images`** (*dict*) – dictionary in format {image hash: image filenames} provided by [`sort_by_duplicates\(\)`](#).

`lecture2notes.end_to_end.imghash.sort_by_duplicates(img_dir, hash_func='phash')`

Find duplicate images in a directory.

Parameters

- **`img_dir`** (*str*) – path to folder containing images to scan for duplicates
- **`hash_func`** (*str*, *optional*) – the hash function to use as given by [`get_hash_func\(\)`](#). Defaults to “`phash`”.

Returns dictionary in format {image hash: image filenames}

Return type [dict]

19.10 OCR

`lecture2notes.end_to_end.ocr.all_in_folder(path)`

Perform OCR using pytesseract on every file in folder and return results

`lecture2notes.end_to_end.ocr.write_to_file(results, save_file)`

Write everything stored in *results* to file at path *save_file*. Used to write results from *all_in_folder()* to *save_file*.

19.11 Segment Cluster

`class lecture2notes.end_to_end.segment_cluster.SegmentCluster(slides_dir,
model_path='model_best.ckpt')`

Iterates through frames in order and splits based on large visual differences (measured by the cosine difference between the feature vectors from the slide classifier)

`extract_and_add_features(gamma=1.3)`

Extracts features from the images in *slides_dir* and saves feature vectors

`transfer_to_filesystem(copy=True, create_best_samples_folder=True)`

Takes all images in directory *slides_dir* and saves each cluster to a subfolder in *cluster_dir* (directory in parent of *slides_dir*)

19.12 SIFT Matcher

```
lecture2notes.end_to_end.sift_matcher.does_camera_move(old_frame, frame, gamma=10,
                                                         border_ratios=(10, 19),
                                                         bottom=False)
```

Detects camera movement between two frames by tracking features in the borders of the image. Only the borders are used because the center of the image probably contains a slide. Thus, tracking features of the slide is not robust since those features will disappear when the slide changes.

Parameters

- **old_frame** (*np.array*) – First frame/image as loaded with `cv2.imread()`
- **frame** (*np.array*) – Second frame/image as loaded with `cv2.imread()`
- **gamma** (*int, optional*) – The threshold pixel movement value. If the camera moves more than this value, then there is assumed to be camera movement between the two frames. Defaults to 10.
- **border_ratios** (*tuple, optional*) – The ratios of the height and width respectively of the first frame to be searched for features. Only the borders are searched for features. these values specify how much of the image should be counted as a border. Defaults to (10, 19).
- **bottom** (*bool, optional*) – Whether to find features in the bottom border. This is not recommended because ‘presenter_slide’ images may have the peoples’ heads at the bottom, which will move and do not represent camera motion. Defaults to False.

Returns (total_movement > gamma, total_movement) If there is camera movement between the two frames and the total movement between the frames.

Return type tuple

```
lecture2notes.end_to_end.sift_matcher.does_camera_move_all_in_folder(folder_path)
```

Runs `does_camera_move()` on all the files in a folder and calculates statistics about camera movement within those files.

Parameters **folder_path** (*str*) – Directory containing the files to be processed.

Returns (movement_detection_percentage, average_move_value, max_move_value) A float representing the percentage of frames where movement was detected from the previous frame. The average of the `total_movement` values returned from `does_camera_move()`. The maximum of the `total_movement` values returned from `does_camera_move()`.

Return type tuple

```
lecture2notes.end_to_end.sift_matcher.is_content_added(first, second,
                                                         first_area_modifier=0.7,
                                                         second_area_modifier=0.4,
                                                         gamma=0.09,
                                                         dilation_amount=22)
```

Detect if `second` contains more content than `first` and how much more content it adds. This algorithm dilates both images and finds contours. It then computes the total area of those contours. If `gamma%` more than the area of the first image’s contours is greater than the area of the second image’s contours then it is assumed more content is added.

Parameters

- **first** (*np.array*) – Image loaded using `cv2.imread()` belonging to the ‘slide’ class

- **second** (*np.array*) – Image loaded using `cv2.imread()` belonging to the ‘`presenter_slide`’ class
- **first_area_modifier** (*float, optional*) – The maximum percent area of the first image that a contour can take up before it is excluded. Defaults to 0.70.
- **second_area_modifier** (*float, optional*) – The maximum percent area of the second image that a contour can take up before it is excluded. Images belonging to the ‘`presenter_slide`’ class are more likely to have mistaken large contours. Defaults to 0.40.
- **gamma** (*float, optional*) – The percentage increase in content area necessary for `second` to be classified as having more content than `first`. Defaults to 0.09.
- **dilation_amount** (*int, optional*) – How much the canny edge maps of each both images `first` and `second` should be dilated. This helps to combine multiple components of one object into a single contour. Defaults to 22.

Returns (`content_is_added`, `amount_of_added_content`) Boolean if `second` contains more content than `first` and float describing the difference in content from `first` to `second`. `amount_of_added_content` can be negative.

Return type tuple

```
lecture2notes.end_to_end.sift_matcher.match_features(slide_path, presenter_slide_path,  
                                                    min_match_count=33,  
                                                    min_area_percent=0.37,  
                                                    do_motion_detection=True)
```

Match features between images in `slide_path` and `presenter_slide_path`. The images in `slide_path` are the queries to the matching algorithm and the images in `presenter_slide_path` are the train/searched images.

Parameters

- **slide_path** (*str*) – Path to the images classified as “slide” or any directory containing query images.
- **presenter_slide_path** (*str*) – Path to the images classified as “presenter_slide” or any directory containing train images.
- **min_match_count** (*int, optional*) – The minimum number of matches returned by `sift_flann_match()` required for the image pair to be considered as containing the same slide. Defaults to 33.
- **min_area_percent** (*float, optional*) – Percentage of the area of the train image (images belonging to the ‘`presenter_slide`’ category) that a matched slide must take up to be counted as a legitimate duplicate slide. This removes incorrect matches that can result in crops to small portions of the train image. Defaults to 0.37.
- **do_motion_detection** (*bool, optional*) – Whether motion detection using `does_camera_move_all_in_folder()` should be performed. If set to False then it is assumed that there is movement since assuming no movement leaves room for a lot of false positives. If no camera motion is detected and this option is enabled then all slides that are unique to the “presenter_slide” category (they have no matches in the “slide” category) will automatically be cropped to contain just the slide. They will be saved to the originating folder but with the string defined by the variable `OUTPUT_PATH_MODIFIER` in their filename. Even if `does_camera_move_all_in_folder()` detects no movement it is still possible that movement is detected while running this function since a check is performed

to make sure all slide bounding boxes found contain 80% overlapping area with all previously found bounding boxes. Defaults to True.

Returns (non_unique_presenter_slides, transformed_image_paths)
 non_unique_presenter_slides: The images in the “presenter_slide” category that are not unique and should be deleted transformed_image_paths: The paths to the cropped images if *do_motion_detection* was enabled and no motion was detected.

Return type tuple

```
lecture2notes.end_to_end.sift_matcher.ransac_transform(sift_matches, kp1, kp2, img1,
                                                    img2, draw_matches=False)
```

Use data from [sift_flann_match\(\)](#) to find the coordinates of *img1* in *img2*. *sift_matches*, *kp1*, *kp2*, *img1*, and *img2* are all the outputs of `meth:~sift_matcher.sift_flann_match`. If *draw_matches* is enabled then the features matches will be drawn and shown on the screen.

Returns The corner coordinates of the quadrilateral representing *img1* within *img2*.

Return type np.array

```
lecture2notes.end_to_end.sift_matcher.sift_flann_match(query_image, train_image,
                                                    algorithm='orb',
                                                    num_features=1000)
```

Locate *query_image* within *train_image* using *algorithm* for feature detection/description and FLANN (Fast Library for Approximate Nearest Neighbors) for matching. You can read more about matching in the [OpenCV “Feature Matching” documentation](#) or about homography on the [OpenCV Python Tutorial “Feature Matching + Homography to find Objects”](#)

Parameters

- **query_image** (*np.array*) – Image to find. Loading using `cv2.imread()`.
- **train_image** (*np.array*) – Image to search. Loading using `cv2.imread()`.
- **algorithm** (*str*, *optional*) – The feature detection/description algorithm. Can be one of [ORB](#), ([ORB Class Reference](#)) [SIFT](#), ([SIFT Class Reference](#)) or [FAST](#). ([FAST Class Reference](#)) Defaults to “orb”.
- **num_features** (*int*, *optional*) – The maximum number of features to retain when using *ORB* and *SIFT*. Does not take effect when using the *FAST* detection algorithm. Setting to 0 for *SIFT* is a good starting point. The default for *ORB* is 500, but it was increased to 1000 to improve accuracy. Defaults to 1000.

Returns (good, kp1, kp2, img1, img2) The good matches as per Lowe’s ratio test, the key points from image 1, the key points from image 2, modified image 1, and modified image 2.

Return type tuple

19.13 Slide Classifier

```
lecture2notes.end_to_end.slide_classifier.classify_frames(frames_dir, do_move=True,  
                                                         incorrect_threshold=0.6,  
                                                         model_path='model_best.ckpt')
```

Classifies images in a directory using the slide classifier model.

Parameters

- **frames_dir** (*str*) – path to directory containing images to classify
- **do_move** (*bool, optional*) – move the images to their sorted folders instead of copying them. Defaults to True.
- **incorrect_threshold** (*float, optional*) – the certainty value that the model must be below for a prediction to be marked “probably incorrect”. Defaults to 0.60.

Returns (frames_sorted_dir, certainties, percent_wrong)

Return type [tuple]

19.14 Slide Structure Analysis

```
lecture2notes.end_to_end.slide_structure_analysis.all_in_folder(path,  
                                                                do_rename=True,  
                                                                **kwargs)
```

Perform structure analysis and OCR on every file in folder using [analyze_structure\(\)](#).

Parameters

- **path** (*str*) – Directory containing images to process.
- **do_rename** (*str, optional*) – Rename files to just their frame number. Defaults to True.
- ****kwargs** (*dict, optional*) – [lecture2notes.end_to_end.slide_structure_analysis.analyze_structure\(\)](#).

Returns (raw_texts, json_texts) A list of the raw text for each slide and a list of the json structure analysis data for each slide.

Return type tuple

```
lecture2notes.end_to_end.slide_structure_analysis.analyze_structure(image,  
                                                                    to_json=None,  
                                                                    re-  
                                                                    turn_unstructured_text=True,  
                                                                    gamma=0.1,  
                                                                    beta=0.2,  
                                                                    orient='index',  
                                                                    ex-  
                                                                    tra_json=None)
```

Perform slide structure analysis.

Parameters

- **image** (*np.array*) – Image to be processed as loaded with `cv2.imread()`.

- **to_json** (*str or bool, optional*) – Path to write json output or a boolean to return json data as a string. The default return value is a `pd.DataFrame`. Defaults to `None`.
- **return_unstructured_text** (*bool, optional*) – If the raw recognized text should be returned in addition to the other return values.
- **gamma** (*float, optional*) – The percentage greater than or less than the average **stroke width** that a text line must meet to be classified as bold/subtitle or small text respectively. Defaults to 0.1.
- **beta** (*float, optional*) – The percentage greater than or less than the average **height** that a text line must meet to be classified as bold/subtitle or small text respectively. This is greater than **gamma** because height is on a larger scale than gamma. Defaults to 0.2.
- **orient** (*str, optional*) – The format of the output json data if **to_json** is set. The acceptable values can be found on the [pandas.DataFrame.to_json documentation](#). Defaults to “index”.
- **extra_json** (*dict, optional*) – Additional keys and values to add to the json output if **to_json** is enabled. Defaults to `None`.

Returns The default is to return a `pd.DataFrame`. However, setting **to_json** to a string will instead write json data to **to_json** and return the path to the data. Setting **to_json** to `True` will return the json data as a string. Setting **return_unstructured_text** returns the previously described data and the raw recognized text as a tuple. Will return `None` if no text is detected.

Return type `pd.DataFrame` or `str` or `tuple` or `None`

```
lecture2notes.end_to_end.slide_structure_analysis.identify_title(tesseract_df,
                                                                image,
                                                                left_start_maximum=0.77,
                                                                character_limit=3,
                                                                enabled_checks=None)
```

```
lecture2notes.end_to_end.slide_structure_analysis.stroke_width(image)
Determine the average stroke length in an image. Inspired by: https://stackoverflow.com/a/61914060.
```

Other Links:

- [cv2.distanceTransform Documentation](#)
- [OpenCV Distance Transform Tutorial](#)
- [Sckit-Image “Finding local maxima”](#)
- [skimage.feature.peak_local_max](#)

```
lecture2notes.end_to_end.slide_structure_analysis.write_to_file(raw_texts,
                                                                json_texts,
                                                                raw_save_file,
                                                                json_save_file)
```

Write the raw text in `raw_texts` to `raw_save_file` and the json data in `json_texts` to `json_save_file`. Used to write results from [all_in_folder\(\)](#) to disk.

Parameters

- **raw_texts** (*list*) – List of raw text outputs from [analyze_structure\(\)](#).
- **json_texts** (*list*) – List of json ssa outputs from [analyze_structure\(\)](#).

- **raw_save_file** (*str*) – The path to save the raw text. A “.txt” file.
- **json_save_file** (*str*) – The path to save the json output. A “.json” file.

19.15 Spell Check

```
class lecture2notes.end_to_end.spell_check.SpellChecker(max_edit_distance_dictionary=2,  
                                                         max_edit_distance_lookup=2,  
                                                         prefix_length=7)
```

A spell checker.

check(*input_term*)

Checks an input string for spelling mistakes

Parameters **input_term** (*str*) – the sequence to check for spelling errors

Returns the best corrected string

Return type [str]

check_all(*input_terms*)

Spell check multiple sequences by calling **check()** for each item in **input_terms**.

Parameters **input_terms** (*list*) – a list of strings to be corrected with spell checking

Returns a list of corrected strings

Return type [list]

19.16 Summarization Approaches

```
lecture2notes.end_to_end.summarization_approaches.cluster(text,  
                                                           coverage_percentage=0.7,  
                                                           final_sort_by=None, cluster_summarizer='extractive',  
                                                           title_generation=False,  
                                                           num_topics=10,  
                                                           minibatch=False,  
                                                           hf_inference_api=False, feature_extraction='neural_sbert',  
                                                           **kwargs)
```

Summarize text to **coverage_percentage** length of the original document by extracting features from the text, clustering based on those features, and finally summarizing each cluster. See the [scikit-learn documentation on clustering text](#) for more information since several sections of this function were borrowed from that example.

Notes

- ****kwargs** is passed to the feature extraction function, which is either [extract_features_bow\(\)](#) or [extract_features_neural\(\)](#) depending on the **feature_extraction** argument.

Parameters

- **text** (*str*) – a string of text to summarize
- **coverage_percentage** (*float, optional*) – The length of the summary as a percentage of the original document. Defaults to 0.70.

- **final_sort_by** (*str*, *optional*) – If *cluster_summarizer* is extractive and *title_generation* is False then this argument is available. If specified, it will sort the final cluster summaries by the specified string. Options are ["order", "rating"]. Defaults to None.
- **cluster_summarizer** (*str*, *optional*) – Which summarization method to use to summarize each individual cluster. “Extractive” uses the same approach as *keyword_based_ext()* but instead of using keywords from another document, the keywords are calculated in the TfidfVectorizer or HashingVectorizer. Each keyword is a feature in the document-term matrix, thus the number of words to use is specified by the *n_features* parameter. Options are ["extractive", "abstractive"]. Defaults to “extractive”.
- **title_generation** (*bool*, *optional*) – Option to generate titles for each cluster. Can not be used if *final_sort_by* is set. Generates titles by summarizing the text using BART finetuned on XSum (a dataset of news articles and one sentence summaries aka headline generation) and forcing results to be from 1 to 10 words long. Defaults to False.
- **num_topics** (*int*, *optional*) – The number of clusters to create. This should be set to the number of topics discussed in the lecture if generating good titles is desired. If separating into groups is not very important and a final summary is desired then this parameter is not incredibly important, it just should not be set super low (3) or super high (50) unless your document is super short or long. Defaults to 10.
- **minibatch** (*bool*, *optional*) – Two clustering algorithms are used: ordinary k-means and its more scalable cousin minibatch k-means. Setting this to True will use minibatch k-means with a batch size set to the number of clusters set in *num_topics*. Defaults to False.
- **hf_inference_api** (*bool*, *optional*) – Use the huggingface inference API for abstractive summarization. Defaults to False.
- **feature_extraction** (*str*, *optional*) – Specify how features should be extracted from the text.
 - **neural_hf**: uses a huggingface/transformers pipeline with the roberta model by default
 - **neural_sbert**: special bert and roberta models fine-tuned to extract sentence embeddings
 - * GitHub: <https://github.com/UKPLab/sentence-transformers>
 - * Paper: <https://arxiv.org/abs/1908.10084>
 - **spacy**: uses spacy model. All other options use the small spacy model to split the text into sentences since sentence detection does not improve with larger models. However, if spacy is specified for *feature_selection* than the *en_core_web_lg* model will be used to extract high-quality embeddings
 - **bow**: bow = “bag of words”. this method is extremely fast since it is based on word frequencies throughout the input text. The *extract_features_bow()* function contains more details on recommended parameters that you can pass to this function because of ***kwargs*.

Options are ["neural_hf", "neural_sbert", "spacy", "bow"] Default is “neural_sbert”.

Raises Exception – If incorrect parameters are passed.

Returns The summarized text as a normal string. Line breaks will be included if `title_generation` is true.

Return type [str]

```
lecture2notes.end_to_end.summarization_approaches.compute_ranks(sigma, v_matrix)
lecture2notes.end_to_end.summarization_approaches.create_sumy_summarizer(algorithm,
                                                                           lan-
                                                                           guage='english')
lecture2notes.end_to_end.summarization_approaches.extract_features_bow(data, re-
                                                                           turn_lsa_svd=False,
                                                                           use_hashing=False,
                                                                           use_idf=True,
                                                                           n_features=10000,
                                                                           lsa_num_components=False)
```

Extract features using a bag of words statistical word-frequency approach.

Parameters

- **data** (*list*) – List of sentences to extract features from
- **return_lsa_svd** (*bool, optional*) – Return the features and `lsa_svd`. See “Returns” section below. Defaults to False.
- **use_hashing** (*bool, optional*) – Use a `HashingVectorizer` instead of a `CountVectorizer`. Defaults to False. A `HashingVectorizer` should only be used with large datasets. Large to the degree that you’ll probably never pass enough data through this function to warrant the usage of a `HashingVectorizer`. `HashingVectorizers` use very little memory and are thus scalable to large datasets because there is no need to store a vocabulary dictionary in memory. More information can be found in the [HashingVectorizer scikit-learn documentation](#).
- **use_idf** (*bool, optional*) – Option to use inverse document-frequency. Defaults to True. In the case of `use_hashing` a `TfidfTransformer` will be appended in a pipeline after the `HashingVectorizer`. If not `use_hashing` then the `use_idf` parameter of the `TfidfVectorizer` will be set to `use_idf`. This step is important because, as explained by the [scikit-learn documentation](#): “In a large text corpus, some words will be very present (e.g. ‘the’, ‘a’, ‘is’ in English) hence carrying very little meaningful information about the actual contents of the document. If we were to feed the direct count data directly to a classifier those very frequent terms would shadow the frequencies of rarer yet more interesting terms. In order to re-weight the count features into floating point values suitable for usage by a classifier it is very common to use the tf-idf transform.”
- **n_features** (*int, optional*) – Specifies the number of features/words to use in the vocabulary (which are the rows of the document-term matrix). In the case of the `TfidfVectorizer` the `n_features` acts as a maximum since the `max_df` and `min_df` parameters choose words to add to the vocabulary (to use as features) that occur within the bounds specified by these parameters. This value should probably be lowered if `use_hashing` is set to True. Defaults to 10000.
- **lsa_num_components** (*int, optional*) – If set then preprocess the data using latent semantic analysis to reduce the dimensionality to `lsa_num_components` components. Defaults to False.

Returns list of features extracted and optionally the `u`, `sigma`, and `v` of the `svd` calculation on the document-term matrix. only returns if `return_lsa_svd` set to True.

Return type [list or tuple]

```
lecture2notes.end_to_end.summarization_approaches.extract_features_neural_hf(sentences,
                                                                              model='roberta-
                                                                              base',
                                                                              tokenizer='roberta-
                                                                              base',
                                                                              n_hidden=768,
                                                                              squeeze=True,
                                                                              **kwargs)
```

Extract features using a transformer model from the huggingface/transformers library

```
lecture2notes.end_to_end.summarization_approaches.extract_features_neural_sb(sentences,
                                                                              model='roberta-
                                                                              base-
                                                                              nli-
                                                                              mean-
                                                                              tokens')
```

Extract features using Sentence-BERT (SBERT) or SROBERTa from the sentence-transformers library

```
lecture2notes.end_to_end.summarization_approaches.extract_features_spacy(sentences)
```

```
lecture2notes.end_to_end.summarization_approaches.full_sents(ocr_text, transcript_text,
                                                                remove_newlines=True,
                                                                cut_off=0.7)
```

```
lecture2notes.end_to_end.summarization_approaches.generic_abstractive(to_summarize,
                                                                          summarizer=None,
                                                                          min_length=None,
                                                                          max_length=None,
                                                                          hf_inference_api=False,
                                                                          *args,
                                                                          **kwargs)
```

```
lecture2notes.end_to_end.summarization_approaches.generic_abstractive_hf_api(to_summarize,
                                                                              summarizer='facebook/bart-
                                                                              large-
                                                                              cnn',
                                                                              *args,
                                                                              **kwargs)
```

```
lecture2notes.end_to_end.summarization_approaches.generic_extractive_sumy(text,
                                                                              cover-
                                                                              age_percentage=0.7,
                                                                              algo-
                                                                              rithm='text_rank',
                                                                              lan-
                                                                              guage='english')
```

```
lecture2notes.end_to_end.summarization_approaches.get_best_sentences(sentences,
                                                                        count, rating,
                                                                        *args,
                                                                        **kwargs)
```

```
lecture2notes.end_to_end.summarization_approaches.get_complete_sentences(text, re-
                                                                              turn_string=False)
```

```
lecture2notes.end_to_end.summarization_approaches.get_sentences(text,
                                                                model='en_core_web_sm')

lecture2notes.end_to_end.summarization_approaches.initialize_abstractive_model(sum_model,
                                                                                use_hf_pipeline=True,
                                                                                *args,
                                                                                **kwargs)

lecture2notes.end_to_end.summarization_approaches.keyword_based_ext(ocr_text,
                                                                    transcript_text,
                                                                    cover-
                                                                    age_percentage=0.7)

lecture2notes.end_to_end.summarization_approaches.structured_joined_sum(ssa_path,
                                                                    tran-
                                                                    script_json_path,
                                                                    frame_every_x=1,
                                                                    end-
                                                                    ing_char='.',
                                                                    first_slide_frame_num=0,
                                                                    to_json=False,
                                                                    summa-
                                                                    riza-
                                                                    tion_method='abstractive',
                                                                    max_summarize_len=50,
                                                                    abs_summarizer='sshleifer/distilbo
                                                                    cnn-12-6',
                                                                    ext_summarizer='text_rank',
                                                                    hf_inference_api=False,
                                                                    *args,
                                                                    **kwargs)
```

Summarize slides by combining the Slide Structure Analysis (SSA) and transcript json to create a per slide summary of the transcript. The content from the beginning of one slide to the start of the next to the nearest **ending_char** is considered the transcript that belongs to that slide. The summarized transcript content is organized in a dictionary where the slide titles are keys. This dictionary can be returned as json or written to a json file.

Parameters

- **ssa_path** (*str*) – Path to the SSA JSON file.
- **transcript_json_path** (*str*) – Path to the transcript JSON file.
- **frame_every_x** (*int*, *optional*) – How often frames were extracted from the video that the SSA was conducted on. This is used to convert frame numbers to time (seconds). Defaults to 1.
- **ending_char** (*str*, *optional*) – The character that the transcript belonging to each slide will be extended to. For instance, if the next slide appears in the middle of a word, the transcript content will continue to be added to the previous slide until the **ending_char** is reached. It is recommended to use periods or a special end of sentence token if present. These can be generated with `lecture2notes.end_to_end.transcribe.transcribe_main.segment_sentences()` Defaults to " " (nearest complete word).
- **first_slide_frame_num** (*int*, *optional*) – The frame number of the first slide. Used to create a ‘preface’ (aka an introduction) if the first slide is not immediately shown. Defaults to 0.

- **to_json** (*bool or str, optional*) – If the output dictionary should be returned as a JSON string. This can also be set to a path as a string and the JSON data will be dumped to the file at that path. Defaults to False.
- **summarization_method** (*str, optional*) – The method to use to summarize each slide’s transcript content. Options include “abstractive”, “extractive”, or “none”. Defaults to “abstractive”.
- **max_summarize_len** (*int, optional*) – Text longer than this many tokens will be summarized. Defaults to 50.
- **abs_summarizer** (*str, optional*) – The abstractive summarization model to use if *summarization_method* is “abstractive”. Defaults to “sshleifer/distilbart-cnn-12-6”.
- **hf_inference_api** (*bool, optional*) – Use the huggingface inference API for abstractive summarization. Defaults to False.
- **function** (**args and **kwargs are passed to the summarization*) – *generic_abstractive()* or *generic_extractive_sumy()* depending on *summarization_method*.
- **either** (*which is*) – *generic_abstractive()* or *generic_extractive_sumy()* depending on *summarization_method*.

Returns A dictionary containing the slide titles as keys and the summarized transcript content for each slide as values. A string will be returned when *to_json* is set. If *to_json* is True (boolean) the JSON data formatted as a string will be returned. If *to_json* is a path (string), then the JSON data will be dumped to the file specified and the path to the file will be returned.

Return type dict or str

19.17 Transcript Downloader

```
class lecture2notes.end_to_end.transcript_downloader.TranscriptDownloader(youtube=None,
                                                                           ytdl=True)
```

Download transcripts from YouTube using the YouTube API or youtube-dl.

static check_suffix(*output_path*)

Gets the file extension from *output_path* and verifies it is either “.srt”, “.vtt”, or it is not present in *output_path*. The default is “.vtt”.

download(*video_id, output_path*)

Convenience function to download transcript with one call. If *self.ytdl* is False, calls *get_caption_id()* and passes result to *get_transcript()*. If *self.ytdl* is True, calls *get_transcript_ytdl()*.

get_caption_id(*video_id, lang='en'*)

Gets the caption id with language *lang* for a video on YouTube with id *video_id*.

get_transcript_api(*caption_id, output_path*)

Downloads a caption track by id directly from the YouTube API.

Parameters

- **caption_id** (*str*) – the id of the caption track to download
- **output_path** (*str*) – path to save the captions. file extensions are parsed by *check_suffix()*

Returns the path where the transcript was saved (may not be the same as the `output_path` parameter)

Return type [str]

get_transcript_ytdl(*video_id*, *output_path*)

Gets the transcript for `video_id` using `youtube-dl` and saves it to `output_path`. The extension from `output_path` will be the `--sub-format` that is passed to the `youtube-dl` command.

19.18 YouTube API

`lecture2notes.end_to_end.youtube_api.init_youtube(oauth=False)`

Initialize the YouTube API. If `oauth` then use the `oauth_client_secret.json` located in the current directory, otherwise use the `YT_API_KEY` environment variable.

INDICES AND TABLES

- `genindex`
- `modindex`
- `search`

PYTHON MODULE INDEX

| lecture2notes.models.slide_classifier.slide_classifier_py
lecture2notes.end_to_end.cluster, 85 40
lecture2notes.end_to_end.corner_crop_transform,
85
lecture2notes.end_to_end.figure_detection, 89
lecture2notes.end_to_end.frames_extractor, 91
lecture2notes.end_to_end.helpers, 91
lecture2notes.end_to_end.imghash, 91
lecture2notes.end_to_end.ocr, 92
lecture2notes.end_to_end.segment_cluster, 92
lecture2notes.end_to_end.sift_matcher, 93
lecture2notes.end_to_end.slide_classifier, 96
lecture2notes.end_to_end.slide_structure_analysis,
96
lecture2notes.end_to_end.spell_check, 98
lecture2notes.end_to_end.summarization_approaches,
98
lecture2notes.end_to_end.summarizer_class, 77
lecture2notes.end_to_end.text_detection, 88
lecture2notes.end_to_end.transcribe.mic_vad_streaming,
84
lecture2notes.end_to_end.transcribe.transcribe_main,
77
lecture2notes.end_to_end.transcribe.webrtcvad_utils,
83
lecture2notes.end_to_end.transcript_downloader,
103
lecture2notes.end_to_end.youtube_api, 104
lecture2notes.models.slide_classifier.class_cluster_scikit,
35
lecture2notes.models.slide_classifier.custom_nnmodules,
36
lecture2notes.models.slide_classifier.grad_cam,
36
lecture2notes.models.slide_classifier.inference,
40
lecture2notes.models.slide_classifier.lr_finder,
37
lecture2notes.models.slide_classifier.mish,
39
lecture2notes.models.slide_classifier.slide_classifier_helpers,
40

INDEX

A

AdaptiveConcatPool2d (class in *lecture2notes.models.slide_classifier.custom_nnmodules*), 36

add() (*lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster* static method), 35

add_figures_to_ssa() (in module *lecture2notes.end_to_end.figure_detection*), 89

add_model_specific_args() (*lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier* static method), 40

all_in_folder() (in module *lecture2notes.end_to_end.corner_crop_transform*), 85

all_in_folder() (in module *lecture2notes.end_to_end.figure_detection*), 89

all_in_folder() (in module *lecture2notes.end_to_end.ocr*), 92

all_in_folder() (in module *lecture2notes.end_to_end.slide_structure_analysis*), 96

analyze_structure() (in module *lecture2notes.end_to_end.slide_structure_analysis*), 96

area_of_corner_box() (in module *lecture2notes.end_to_end.figure_detection*), 89

area_of_overlapping_rectangles() (in module *lecture2notes.end_to_end.figure_detection*), 89

Audio (class in *lecture2notes.end_to_end.transcribe.mic_vad_streaming*), 84

B

BackPropagation (class in *lecture2notes.models.slide_classifier.grad_cam*), 36

BLOCKS_PER_SECOND (*lecture2notes.end_to_end.transcribe.mic_vad_streaming* attribute), 84

C

calculate_best_k() (*lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster* static method), 35

calculate_stats() (*lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier* static method), 40

caption_file_to_string() (in module *lecture2notes.end_to_end.transcribe.transcribe_main*), 77

CHANNELS (*lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio* attribute), 84

check() (*lecture2notes.end_to_end.spell_check.SpellChecker* static method), 98

check_all() (*lecture2notes.end_to_end.spell_check.SpellChecker* static method), 98

check_suffix() (*lecture2notes.end_to_end.transcript_downloader.TranscriptDownloader* static method), 103

check_transcript() (in module *lecture2notes.end_to_end.transcribe.transcribe_main*), 77

chunk_by_silence() (in module *lecture2notes.end_to_end.transcribe.transcribe_main*), 77

chunk_by_speech() (in module *lecture2notes.end_to_end.transcribe.transcribe_main*), 78

classify_frames() (in module *lecture2notes.end_to_end.slide_classifier*), 96

Cluster (class in *lecture2notes.models.slide_classifier.class_cluster_scikit*), 35

cluster() (in module *lecture2notes.end_to_end.summarization_approaches*), 98

cluster_points() (in module *lecture2notes.end_to_end.corner_crop_transform*), 85

ClusterFilesystem (class in *lecture2notes.end_to_end.cluster*), 85

compute_ranks() (in module *lecture2notes.end_to_end.summarization_approaches*), 98

`lecture2notes.end_to_end.summarization_approaches`
`100`
`configure_optimizers()` (lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier method), 41
`contour_offset()` (in module lecture2notes.end_to_end.corner_crop_transform), 85
`convert_coords_to_corners()` (in module lecture2notes.end_to_end.figure_detection), 89
`convert_deepspeech_json()` (in module lecture2notes.end_to_end.transcribe.transcribe_main), 78
`convert_relu_to_mish()` (in module lecture2notes.models.slide_classifier.slide_classifier_helpers), 40
`convert_samplerate()` (in module lecture2notes.end_to_end.transcribe.transcribe_main), 78
`copy_all()` (in module lecture2notes.end_to_end.helpers), 91
`create_affinity_propagation()` (lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster method), 35
`create_algorithm_if_none()` (lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster method), 35
`create_kmeans()` (lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster method), 35
`create_sumy_summarizer()` (in module lecture2notes.end_to_end.summarization_approaches), 100
`crop()` (in module lecture2notes.end_to_end.corner_crop_transform), 85

D

`Deconvnet` (class in lecture2notes.models.slide_classifier.grad_cam), 36
`destroy()` (lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio method), 84
`detect_color_image()` (in module lecture2notes.end_to_end.figure_detection), 89
`detect_figures()` (in module lecture2notes.end_to_end.figure_detection), 89
`determine_root_path()` (lecture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77
`does_camera_move()` (in module lecture2notes.end_to_end.sift_matcher), 93

E

`edges_det()` (in module lecture2notes.end_to_end.corner_crop_transform), 86
`ExponentialLR` (class in lecture2notes.models.slide_classifier.lr_finder), 37
`extract_and_add_features()` (lecture2notes.end_to_end.cluster.ClusterFilesystem method), 85
`extract_and_add_features()` (lecture2notes.end_to_end.segment_cluster.SegmentCluster method), 92
`extract_audio()` (in module lecture2notes.end_to_end.transcribe.transcribe_main), 79
`extract_features_bow()` (in module lecture2notes.end_to_end.summarization_approaches), 100
`extract_features_neural_hf()` (in module lecture2notes.end_to_end.summarization_approaches), 101
`extract_features_neural_sbert()` (in module lecture2notes.end_to_end.summarization_approaches), 101
`extract_features_spacy()` (in module lecture2notes.end_to_end.summarization_approaches), 101
`extract_frames()` (in module lecture2notes.end_to_end.frames_extractor), 91

F

`f_mish()` (in module lecture2notes.models.slide_classifier.mish), 39
`find_intersection()` (in module lecture2notes.end_to_end.corner_crop_transform), 86
`find_page_contours()` (in module lecture2notes.end_to_end.corner_crop_transform), 86
`FORMAT` (lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio attribute), 84
`forward()` (lecture2notes.models.slide_classifier.custom_nnmodules.Adapt method), 36
`forward()` (lecture2notes.models.slide_classifier.grad_cam.BackPropagation method), 36

forward() (lecture2notes.models.slide_classifier.mish.mishget_device() (in module lec-
 method), 39
 forward() (lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
 method), 41
 four_corners_sort() (in module lec-
 ture2notes.end_to_end.corner_crop_transform), 91
 87
 Frame (class in lecture2notes.end_to_end.transcribe.webrtcvad_utils), method), 41
 83
 frame_duration_ms (lec-
 ture2notes.end_to_end.transcribe.mic_vad_streaming.VADAudioInfo (lecture2notes.models.slide_classifier.lr_finder.ExponentialLR
 property), 84
 method), 37
 frame_generator() (in module lec-
 ture2notes.end_to_end.transcribe.webrtcvad_utils), method), 39
 83
 frame_generator() (lec-
 ture2notes.end_to_end.transcribe.mic_vad_streaming.VADAudioInfo (lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster
 method), 84
 method), 35
 frame_number_filename_mapping() (in module lec-
 ture2notes.end_to_end.helpers), 91
 frame_number_from_filename() (in module lec-
 ture2notes.end_to_end.helpers), 91
 full_sents() (in module lec-
 ture2notes.end_to_end.summarization_approaches), 101
 101
G
 gen_unique_id() (in module lec-
 ture2notes.end_to_end.helpers), 91
 generate() (lecture2notes.models.slide_classifier.grad_cam.BackPropagation (lecture2notes.end_to_end.transcript_downloader.TranscriptDownload
 method), 36
 method), 103
 generate() (lecture2notes.models.slide_classifier.grad_cam.GuidedBackPropagation (lecture2notes.end_to_end.transcript_downloader.TranscriptDownload
 method), 36
 method), 104
 generic_abstractive() (in module lec-
 ture2notes.end_to_end.summarization_approaches), 101
 get_vector_array() (lec-
 ture2notes.models.slide_classifier.class_cluster_scikit.Cluster
 method), 35
 generic_abstractive_hf_api() (in module lec-
 ture2notes.end_to_end.summarization_approaches), 101
 get_vectors() (lecture2notes.models.slide_classifier.class_cluster_scikit.
 method), 35
 generic_extractive_sumy() (in module lec-
 ture2notes.end_to_end.summarization_approaches), 101
 get_youtube_transcript() (in module lec-
 ture2notes.end_to_end.transcribe.transcribe_main), 79
 get_best_sentences() (in module lec-
 ture2notes.end_to_end.summarization_approaches), 101
 GradCAM (class in lec-
 ture2notes.models.slide_classifier.grad_cam), 36
 get_caption_id() (lec-
 ture2notes.end_to_end.transcript_downloader.TranscriptDownload (lecture2notes.models.slide_classifier.grad_cam),
 method), 103
 method), 36
 get_closest_sample_filenames_to_centroids() (lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster
 method), 35
H
 get_complete_sentences() (in module lec-
 ture2notes.end_to_end.summarization_approaches), 101
 horizontal_vertical_edges_det() (in module lec-
 ture2notes.end_to_end.corner_crop_transform), 87
 hough_lines_corners() (in module lec-

`lecture2notes.end_to_end.corner_crop_transform`),
 87
 |
`identify_title()` (in module `lecture2notes.end_to_end.slide_structure_analysis`),
 97
`init_youtube()` (in module `lecture2notes.end_to_end.youtube_api`), 104
`initialize_abstractive_model()` (in module `lecture2notes.end_to_end.summarization_approaches`),
 102
`initialize_model()` (in module `lecture2notes.models.slide_classifier.inference`),
 40
`initialize_model()` (`lecture2notes.models.slide_classifier.slide_classifier.pytorch_slide_classifier`
 method), 41
`is_content_added()` (in module `lecture2notes.end_to_end.sift_matcher`), 93
 K
`keyword_based_ext()` (in module `lecture2notes.end_to_end.summarization_approaches`),
 102
 L
`lecture2notes.end_to_end.cluster`
 module, 85
`lecture2notes.end_to_end.corner_crop_transform`
 module, 85
`lecture2notes.end_to_end.figure_detection`
 module, 89
`lecture2notes.end_to_end.frames_extractor`
 module, 91
`lecture2notes.end_to_end.helpers`
 module, 91
`lecture2notes.end_to_end.imghash`
 module, 91
`lecture2notes.end_to_end.ocr`
 module, 92
`lecture2notes.end_to_end.segment_cluster`
 module, 92
`lecture2notes.end_to_end.sift_matcher`
 module, 93
`lecture2notes.end_to_end.slide_classifier`
 module, 96
`lecture2notes.end_to_end.slide_structure_analysis`
 module, 96
`lecture2notes.end_to_end.spell_check`
 module, 98
`lecture2notes.end_to_end.summarization_approaches`
 module, 98
`lecture2notes.end_to_end.summarizer_class`
 module, 77
`lecture2notes.end_to_end.text_detection`
 module, 88
`lecture2notes.end_to_end.transcribe.mic_vad_streaming`
 module, 84
`lecture2notes.end_to_end.transcribe.transcribe_main`
 module, 77
`lecture2notes.end_to_end.transcribe.webrtcvad_utils`
 module, 83
`lecture2notes.end_to_end.transcript_downloader`
 module, 103
`lecture2notes.end_to_end.youtube_api`
 module, 104
`lecture2notes.models.slide_classifier.class_cluster_scikit`
 module, 35
`lecture2notes.models.slide_classifier.custom_nnmodules`
 module, 36
`lecture2notes.models.slide_classifier.grad_cam`
 module, 36
`lecture2notes.models.slide_classifier.inference`
 module, 40
`lecture2notes.models.slide_classifier.lr_finder`
 module, 37
`lecture2notes.models.slide_classifier.mish`
 module, 39
`lecture2notes.models.slide_classifier.slide_classifier_helpers`
 module, 40
`lecture2notes.models.slide_classifier.slide_classifier_pytorch`
 module, 40
`LectureSummarizer` (class in `lecture2notes.end_to_end.summarizer_class`),
 77
`LinearLR` (class in `lecture2notes.models.slide_classifier.lr_finder`),
 39
`load_deepspeech_model()` (in module `lecture2notes.end_to_end.transcribe.transcribe_main`),
 79
`load_east()` (in module `lecture2notes.end_to_end.text_detection`), 89
`load_images()` (in module `lecture2notes.models.slide_classifier.grad_cam`),
 36
`load_model()` (in module `lecture2notes.end_to_end.transcribe.transcribe_main`),
 79
`load_model()` (in module `lecture2notes.models.slide_classifier.inference`),
 40
`load_model_deprecated()` (in module `lecture2notes.models.slide_classifier.inference`),
 40
`load_vosk_model()` (in module `lecture2notes.end_to_end.transcribe.transcribe_main`),
 79

79
load_wav2vec_model() (in module lecture2notes.end_to_end.transcribe.transcribe_main), 79

LRFinder (class in lecture2notes.models.slide_classifier.lr_finder), 37

M

main() (in module lecture2notes.end_to_end.transcribe.mic_vad_streaming), 85

main() (in module lecture2notes.models.slide_classifier.grad_cam), 36

make_dir_if_not_exist() (in module lecture2notes.end_to_end.helpers), 91

match_features() (in module lecture2notes.end_to_end.sift_matcher), 94

metadata_to_json() (in module lecture2notes.end_to_end.transcribe.transcribe_main), 79

metadata_to_list() (in module lecture2notes.end_to_end.transcribe.transcribe_main), 79

metadata_to_string() (in module lecture2notes.end_to_end.transcribe.transcribe_main), 79

mish (class in lecture2notes.models.slide_classifier.mish), 39

module

- lecture2notes.end_to_end.cluster, 85
- lecture2notes.end_to_end.corner_crop_transform, 85
- lecture2notes.end_to_end.figure_detection, 89
- lecture2notes.end_to_end.frames_extractor, 91
- lecture2notes.end_to_end.helpers, 91
- lecture2notes.end_to_end.imghash, 91
- lecture2notes.end_to_end.ocr, 92
- lecture2notes.end_to_end.segment_cluster, 92
- lecture2notes.end_to_end.sift_matcher, 93
- lecture2notes.end_to_end.slide_classifier, 96
- lecture2notes.end_to_end.slide_structure_analysis, 96
- lecture2notes.end_to_end.spell_check, 98
- lecture2notes.end_to_end.summarization_approaches, 98
- lecture2notes.end_to_end.summarizer_class, 77

lecture2notes.end_to_end.text_detection, 88

lecture2notes.end_to_end.transcribe.mic_vad_streaming, 84

lecture2notes.end_to_end.transcribe.transcribe_main, 77

lecture2notes.end_to_end.transcribe.webrtcvad_utils, 83

lecture2notes.end_to_end.transcript_downloader, 103

lecture2notes.end_to_end.youtube_api, 104

lecture2notes.models.slide_classifier.class_cluster_scikit, 35

lecture2notes.models.slide_classifier.custom_nnmodules, 36

lecture2notes.models.slide_classifier.grad_cam, 36

lecture2notes.models.slide_classifier.inference, 40

lecture2notes.models.slide_classifier.lr_finder, 37

lecture2notes.models.slide_classifier.mish, 39

lecture2notes.models.slide_classifier.slide_classifier_pytorch, 40

lecture2notes.models.slide_classifier.slide_classifier_scikit, 40

O

occlusion_sensitivity() (in module lecture2notes.models.slide_classifier.grad_cam), 36

P

persp_transform() (in module lecture2notes.end_to_end.corner_crop_transform), 88

plot() (lecture2notes.models.slide_classifier.lr_finder.LRFinder method), 38

plot_confusion_matrix() (in module lecture2notes.models.slide_classifier.slide_classifier_helpers), 40

predict() (lecture2notes.models.slide_classifier.class_cluster_scikit.Cluster method), 35

prepare_data() (lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier method), 41

preprocess() (in module lecture2notes.models.slide_classifier.grad_cam), 37

process_chunks() (in module lecture2notes.end_to_end.transcribe.transcribe_main), 79

process_segments() (in module lec- 80
ture2notes.end_to_end.transcribe.transcribe_main), 80

R

range_test() (lecture2notes.models.slide_classifier.lr_finder.LRFinder method), 38

ransac_transform() (in module lec- 95
ture2notes.end_to_end.sift_matcher), 95

RATE_PROCESS (lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio method), 84

read() (lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio method), 84

read_resampled() (lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio method), 84

read_wave() (in module lec- 80
ture2notes.end_to_end.transcribe.transcribe_main), 80

remove_contours() (in module lec- 88
ture2notes.end_to_end.corner_crop_transform), 88

remove_duplicates() (in module lec- 92
ture2notes.end_to_end.imghash), 92

resample() (lecture2notes.end_to_end.transcribe.mic_vad_streaming.Audio method), 84

reset() (lecture2notes.models.slide_classifier.lr_finder.LRFinder method), 39

resize() (in module lec- 88
ture2notes.end_to_end.corner_crop_transform), 88

resolve_deepspeech_models() (in module lec- 80
ture2notes.end_to_end.transcribe.transcribe_main), 80

retrieve() (lecture2notes.models.slide_classifier.lr_finder.LRFinder method), 39

run_all() (lecture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

S

save_gradcam() (in module lec- 37
ture2notes.models.slide_classifier.grad_cam), 37

save_gradient() (in module lec- 37
ture2notes.models.slide_classifier.grad_cam), 37

save_sensitivity() (in module lec- 37
ture2notes.models.slide_classifier.grad_cam), 37

segment_lines() (in module lec- 88
ture2notes.end_to_end.corner_crop_transform), 88

segment_sentences() (in module lec- 97
ture2notes.end_to_end.transcribe.transcribe_main), 97

SegmentCluster (class in lec- 92
ture2notes.end_to_end.segment_cluster), 92

set_parameter_requires_grad() (lec- 41
ture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier method), 41

sift_flann_match() (in module lec- 95
ture2notes.end_to_end.sift_matcher), 95

SlideClassifier (class in lec- 95
ture2notes.models.slide_classifier.slide_classifier_pytorch), 95

sort_by_duplicates() (in module lec- 92
ture2notes.end_to_end.imghash), 92

SpellChecker (class in lec- 98
ture2notes.end_to_end.spell_check), 98

StateCacher (class in lec- 39
ture2notes.models.slide_classifier.lr_finder), 39

step_black_border_removal() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_classify_slides() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_cluster_slides() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_extract_figures() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_extract_frames() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_orthographic_crop() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_slide_structure_analysis() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_summarize() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

step_transcribe_audio() (lec- 77
ture2notes.end_to_end.summarizer_class.LectureSummarizer method), 77

store() (lecture2notes.models.slide_classifier.lr_finder.StateCacher method), 39

straight_lines_in_contour() (in module lec- 88
ture2notes.end_to_end.corner_crop_transform), 88

stroke_width() (in module lec- 97
ture2notes.end_to_end.slide_structure_analysis), 97

structured_joined_sum() (in module lec- 40
ture2notes.end_to_end.summarization_approaches)
102

T

test_data_loader() (lec- 83
ture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
method), 41

test_epoch_end() (lec- method), 84
ture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
method), 41

test_step() (lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
method), 41

time_this() (in module lec- ture2notes.end_to_end.transcribe.mic_vad_streaming.VADAudio
ture2notes.end_to_end.summarizer_class), 84
77

train_data_loader() (lec- ture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
ture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
method), 41

training (lecture2notes.models.slide_classifier.custom_nnmodules.AttentionalCustomSlideClassifier
attribute), 36

training (lecture2notes.models.slide_classifier.mish validation_step() (lec-
attribute), 39 ture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
method), 41

training (lecture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
attribute), 41

training_step() (lec- method), 36
ture2notes.models.slide_classifier.slide_classifier_pytorch.SlideClassifier
method), 41

W

transcribe_audio() (in module lec- write_debug_imgs() (in module lec-
ture2notes.end_to_end.transcribe.transcribe_main), ture2notes.end_to_end.corner_crop_transform),
81 88

transcribe_audio_deepspeech() (in module lec- write_to_file() (in module lec-
ture2notes.end_to_end.transcribe.transcribe_main), ture2notes.end_to_end.ocr), 92
81

transcribe_audio_generic() (in module lec- write_to_file() (in module lec-
ture2notes.end_to_end.transcribe.transcribe_main), ture2notes.end_to_end.slide_structure_analysis),
81 97

transcribe_audio_vosk() (in module lec- write_to_file() (in module lec-
ture2notes.end_to_end.transcribe.transcribe_main), ture2notes.end_to_end.transcribe.transcribe_main),
82 82

transcribe_audio_wav2vec() (in module lec- write_wav() (lecture2notes.end_to_end.transcribe.mic_vad_streaming.Au-
ture2notes.end_to_end.transcribe.transcribe_main), write_wave() (in module lec-
82 ture2notes.end_to_end.transcribe.transcribe_main),
82

TranscriptDownloader (class in lec- 82
ture2notes.end_to_end.transcript_downloader),
103

transfer_to_filesystem() (lec-
ture2notes.end_to_end.cluster.ClusterFilesystem
method), 85

transfer_to_filesystem() (lec-
ture2notes.end_to_end.segment_cluster.SegmentCluster
method), 92

transform_image() (in module lec-
ture2notes.models.slide_classifier.inference),