Lecture2Notes: Summarizing Lecture Videos by Classifying Slides and Analyzing Text

Hayden T. Housen
Pawling High School
Pawling, NY 12564

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 beings 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.

Contents

1 Introduction **1**
- 1.1 Note Taking . . . 1
- 1.2 Previewing . . . 1
- 1.3 Online Lectures . . . 1
- 1.4 Our Proposed Solution . . . 1

2 Literature Review & Related Work **2**
- 2.1 Lecture Summarization . . . 2
- 2.2 Multi-Document Summarization (MDS) . . . 3
- 2.3 Automatic Text Summarization . . . 3
- 2.4 Neural-based Long Sequence Summarization . . . 4
  - 2.4.1 Long Extractive Summarization . . . 4
  - 2.4.2 Long Abstractive Summarization . . . 4
- 2.5 Speech-To-Text . . . 5

3 Methodology **5**
- 3.1 Slide Classification Model . . . 5
  - 3.1.1 Dataset . . . 5
  - 3.1.2 Dataset Processing . . . 5
  - 3.1.3 Model Architecture . . . 6
  - 3.1.4 Model Training Approach . . . 6
  - 3.1.5 Model Results . . . 7
- 3.2 End-To-End Process . . . 8
  - 3.2.1 Overall Explanation . . . 8
  - 3.2.2 Speech-To-Text Implementation . . . 10
  - 3.2.3 Duplicate Image Removal . . . 11
  - 3.2.4 Slide Clustering . . . 11
  - 3.2.5 Figure Extraction . . . 12
- 3.3 Slide Structure Analysis . . . 12
  - 3.3.1 SSA Title Identification . . . 13
- 3.4 Perspective Cropping . . . 13
  - 3.4.1 Feature Matching . . . 13
  - 3.4.2 Corner Crop Transform . . . 14
- 3.5 Combination and Summarization . . . 14
  - 3.5.1 Combination . . . 14
  - 3.5.2 Modifications . . . 15
  - 3.5.3 Extractive Summarization . . . 15
  - 3.5.4 Abstractive Summarization . . . 16

4 TransformerSum **16**
4.1 Extractive Models . . . 17
4.1.1 Converting a Dataset to Extractive . . . 17
4.1.2 Architecture . . . 17
4.1.3 Summarization (Classifier) Layers . . . 18
4.2 Extractive Training . . . 19
4.2.1 Loss Function . . . 19
4.2.2 Evaluation/Predicting . . . 19
4.3 Experiments . . . 19
4.3.1 Pooling Modes . . . 19
4.3.2 Classifier . . . 20
4.4 Final Model Results . . . 20
4.5 Abstractive Models . . . 21

5 Website **21**

6 Conclusion **21**

List of Figures

1 Slide classification model confusion matrices . . . 8
2 Slide classification model loss curves . . . 9
3 Grad-CAM visualizations of slide classification models . . . 9
4 Feature matching algorithm example . . . 13
5 Website lecture process example . . . 22

List of Tables

1 ResNet and EfficientNet architecture modifications. . . . 6
2 Slide classification model performance . . . 7
3 Slide classification model classification report . . . 8
4 Speech-to-text models results . . . 10
5 ROUGE scores of extractive summarization models with different pooling modes . . . 20
6 ROUGE scores of extractive summarization models with different classifiers . . . 20
7 Overall ROUGE scores of extractive summarization models . . . 21

1 Introduction

1.1 Note Taking

Whether it's from lectures [39] or reading [65, 18], note-taking has been shown to improve learning by forcing students to actively interpret information. Even though students are advised to be concise in their note-taking, the quantity of notes has a strong positive correlation to exam performance [57]. Furthermore, students' notes are generally incomplete, and thus not adequate for reviewing the material [38].

To solve this problem instructors may provide guided notes, which are teacher-prepared outlines of the material, but with spaces left for students to fill in key information. Guided notes substantially increase the accuracy of notes taken and improve the test scores of students from elementary school through college [32, 27]. Additionally, students generally prefer guided notes over traditional note-taking, and engage more in class when using guided notes [9].

Another solution to incomplete student notes is to have the instructor compile a detailed set of lecture notes. These instructor-provided notes (IPN) are the same as guided notes but with the blank spaces already filled in. Students who use IPNs generally score higher on exams than do learners who review their own notes [38]. Students are more efficient with detailed IPNs than rough outlines of lecture content because with detailed IPNs students take fewer notes to obtain the same information [31].

Problem Teachers may provide guided notes or completed notes, but in both cases the instructor has to spend additional time to create them. Traditional note-taking approaches place this workload on the students, giving the instructor extra time to perfect the lesson. Video/audio lecture recordings are unwieldy and make review difficult. Students with learning disabilities, such as those with attention disorders, may be unable to take notes or may take unclear notes.

1.2 Previewing

Students are often asked to prepare for class by reading lecture materials. Students want summarized materials to preview due to limited attention spans [73]. Summarized lecture slides reduce the amount of previewing time required without impacting knowledge gained [73]. Compared to those who preview the original slide set, students who preview summarized sets of slides finish previewing in a shorter time without experiencing a drop in pre-lecture quiz scores.

1.3 Online Lectures

In the past decade, Massive Open Online Courses (MOOCs) have enabled the dissemination of instructional videos at scale. These online courses are typically organized as sequences of videos and supporting materials such as assessments and interactive demos [28]. Videos are central to the student learning experience in MOOCs since most students spend the majority of their time in MOOCs watching videos [15, 69]. Learners typically skip over assessment problems, online discussions, and other interactive course components [41]. Transcripts from video lectures are available in many MOOCs, but the most valuable information from each lecture can be challenging to locate [55]. The increase in online lecture content and importance of videos to MOOCs has generated significant demand for tools that enable quicker understanding of these lengthy videos. Summaries of videos provide learners with the key points, allowing them to determine where they should focus their attention.

1.4 Our Proposed Solution

In this work, we propose a multimodal automatic summarization approach for lectures. Our approach extracts the voice transcript and formatted text of unique slides. These two sources are combined and summarized to create detailed notes using a plethora of novel methods. This project is broken into three main components: the slide classifier (including the dataset), the summarization models (neural, non-neural, extractive, and abstractive), and the end-to-end-process (one command to convert to notes). The entire workflow is as follows (see Section 3.2.1):

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: Combine, run modifications, extractive summarization, and abstractive summarization.
9. Convert intermediate outputs to a final notes file (HTML, TXT, markdown, etc.)

For the video portion, we solve the tasks of slide classification and clustering using machine learning. We propose novel computer vision algorithms to extract and format important slide content. To ensure accurate transcripts, we tested several speech-to-text (STT) programs. Our work automates the creation of high-quality summaries of presentations and lectures, alleviating the workload from both students and teachers, enabling easy and effective previewing, giving students with learning disabilities increased access to learning opportunities, and allowing users to effortlessly summarize online lecture videos.

2 Literature Review & Related Work

2.1 Lecture Summarization

Multiple methods have been proposed to create summarized representations of blackboard-style lecture videos [74, 44]. [74] proposes a system that matches drawings and writing with sentences from the transcript into a single "visual transcript." Importantly, "visual transcripts" do not condense the content but merely add images to the transcript. Instead, we summarize slide lectures end-to-end, but our figure detection system is based on [74]. [44] extracts handwriting from blackboard-style lecture videos, but it uses two cameras to remove the presenter and does not process the transcript.

Research in lecture video indexing has produced approaches to extract content from lecture videos. TalkMiner [7] is a lecture search engine that can handle build-up material, which is the progressive build-up of a complete final slide over a sequence of partial versions, by thresholding the pixel difference between two temporally adjacent frames. We solve the build-up problem by implementing the method recommended by [7] to detect and align duplicate slides, which enables our keypoint feature matching algorithm to process slides captured from a camera in the back of the room in addition to standard screen-captured slides.

[88] develops a system that finds slide keyframes, detects text using OCR, and finally extracts lecture structure from the OCR transcript by using geometrical information and text stroke width. Their slide-content structure analysis used to extract an outline of the lecture is the basis for our approach that adds structure to our method's automatically generated summaries. We use the same title identification and text classification thresholds. Unlike [88], we combine structured outlines of the slide content with a summary of the transcript for each slide. [73] summarizes a slide presentation lecture by selecting a subset of slides that maximize the importance of content under a given condition, namely, browsing time.

To detect the most significant slides their system searches for sufficient content, unique content, frequent keywords on that slide, and keywords that rarely appear throughout the presentation. The teacher also inputs a recommended amount of time to spend on each slide. This summarization approach is not robust since it requires unique screen-captured slides as input, does not consider the transcript, and does not attempt to build structured notes.

[55] is the most related to our research since it applies deep learning to lecture summarization. Their system uses BERT [21] to create an embedding vector for each sentence and uses K-Means clustering to group similar sentences. The sentences closest to the cluster centroids are selected as the summary. While this bypasses the long-sequence problem, it produces lackluster summaries. Additionally, this approach is limited because it only tests the BERT model, does not consider the information on the slides, does not fine-tune BERT and merely uses it for embeddings, and has no STT component.

[87] identifies visual entities and links them to descriptive speech from the voice transcript using semantic similarity detection. Then it packs these visual entities into a compact notes arrangement. This approach is distinct from ours since we process video captured by a camera, automatically transcribe audio, and combine and summarize slide and audio content. Identifying figures is a small component of our overall end-to-end pipeline.

2.2 Multi-Document Summarization (MDS)

This research approaches lecture summarization from a multimodal perspective (both video and audio content are processed). Since both of these aspects are eventually converted to text, the problem can be framed as a multi-document summarization problem: one document is the voice transcript, the other is the slide transcript.

A common solution to solve MDS is to concatenate all input documents into one long sequence and pass that through a standard Transformer model or LSTM RNN [46]. However, this approach is agnostic of the hierarchical structures and the relations that might exist among documents [48]. Cross-document links are important in isolating salient information, eliminating redundancy, and creating overall coherent summaries [48]. We model these cross-document links and apply novel methods to MDS (see Section 3.5).

2.3 Automatic Text Summarization

Large-scale pre-trained language models ([21, 62, 89]) and sequence-to-sequence (seq2seq) models ([77, 45, 64]) based on the Transformer architecture [84] have achieved state-of-the-art results in many natural language understanding and natural language generation tasks. This is partly due to their self-attention component, which consists of multiple attention heads [11] working in parallel to capture contextual information from the entire sequence, which makes them suitable for complex tasks like summarization.

Small Language Models Operating large language models under constrained computational budgets is challenging. In this work we take a special interest in compressed language models, which make use of knowledge distillation to reduce model size while maintaining the majority of accuracy [68]. The `distil` models started with Distillated-BERT (DistilBERT), a small, fast, and cheap Transformer model based on BERT architecture. It has 40% fewer parameters than `bert-base-uncased`, runs 60% faster while preserving 99% of BERT's performance as measured on the GLUE language understanding benchmark [68]. MobileBERT [80] is similar to DistilBERT but while DistilBERT compresses BERT by reducing its depth, MobileBERT compresses BERT by reducing its width, which is more effective [80]. MobileBERT is 2.64x smaller and 2.45x faster than DistilBERT [80]. We hypothesize that the performance-to-size ratio of the `distil` models can improve summarization in resource-limited situations.

2.4 Neural-based Long Sequence Summarization

The average number of tokens in the transcripts of the lectures used to train the slide classifier is 8536 tokens per lecture. However, state-of-the-art BERT-style pre-trained models typically have a token limit of 512 to 1024 due to their self-attention mechanisms, which scale quadratically with sequence length. Increasing the token limit to the average number of tokens in a lecture would be infeasible with current hardware due to the increased memory requirements.

To work around this problem most existing approaches partition or shorten the long context into smaller sequences before passing it through the language model. However, shortening sequences and partitioning result in the loss of ending and cross-partition information, respectively. Some approaches use task-specific architectures to mitigate this loss. Specifically for the summarization task, the two-stage processes of [46] and [48] are an improvement over truncating since they select the least important information to remove instead of simply removing the ending information. While multi-document summarization models ([46, 48]) still lose information, they typically handle long sequences better than single-document summarization models because the total length of multiple documents is usually longer than a single document.

Longformer The Longformer is a modified version of the Transformer architecture that can handle long input sequences. It contains a self-attention operation that scales linearly with the sequence length, making it versatile for processing long documents [12]. Models similar to the Longformer exist, but they only focus on autoregressive language modeling (Transformer-XL [20], Adaptive Span [79], Compressive [63]), do not explore the pretrain-finetune setting (BP-Transformer [90]), do not test on tasks other than language modeling (Reformer [40], Sparse [19]), or have limited evaluations (BlockBERT [61]). While there are more efficient transformers for long range problems [83], they are either not available in popular software libraries or are not open source.

2.4.1 Long Extractive Summarization

ExtSum-LG [86] is the extractive state-of-the-art on the ArXiv dataset. ExtSum-LG uses pre-trained GloVe embeddings to represent each word and then computes a score for each sentence by combining sentence, document, and topic vectors, which are obtained using multiple GRUs. Instead, we adapt the modern transformer architecture, which has many advantages over RNNs, to this task using the Longformer.

Although the Longformer is an autoencoding BERT-style model and thus can be fine-tuned on a variety of NLP tasks, its application to extractive summarization is not straightforward. Essentially, the Longformer is a version of RoBERTa [50] with modified self-attention layers and an expanded positional embedding matrix created by stacking RoBERTa's pre-trained matrix. The approach used to create PreSumm [49], which was applied to BERT [21], can also be applied to the Longformer. The Longformer is trained as a masked-language model, which means the output vectors are grounded to tokens instead of sentences, while in extractive summarization, most models manipulate sentence-level representations. This research applies the interval segmentation embeddings and "[CLS]" token techniques from [49] with several extensive modifications to the Longformer to summarize long sequences.

2.4.2 Long Abstractive Summarization

Since the Longformer is an autoencoding model, it cannot be directly applied to abstractive summarization, which is a seq2seq task. BART [45] is a seq2seq language model that achieves near state-of-the-art abstractive text summarization results on the CNN/DailyMail [34] (processed by [70]) dataset. However it uses $O(n^2)$ self-attention and thus cannot process long input sequences. [12] solves this limitation by replacing the full-self attention in the encoder with the efficient local+global attention pattern of the Longformer, similar to how RoBERTa was converted to the Longformer, to create a long version of BART called the LongformerEncoderDecoder (LED). This technique could have been applied to PEGASUS [92] to create Longformer-PEGASUS similarly to the techniques from Big Bird used to create BigBird-PEGASUS [91], however such a large model is out of the scope of this research. Any combination of underlying model (implemented in [85]) and efficient attention modification [83] could have been used, but the Longformer has the greatest compatibility with existing software libraries.

2.5 Speech-To-Text

To extract the audio transcript from the input video, our system uses a STT algorithm. The program supports CMU Sphinx [36], Mozilla's implementation of Baidu's DeepSpeech [29], Vosk [4], and Google's Cloud Speech API. We also use the WebRTC Voice Activity Detector (VAD) [3] to identify and only perform STT on audio segments that contain voice. [58] achieves state-of-the-art WER on the LibriSpeech clean test corpus with a score of 1.7% while the DeepSpeech model achieves a 5.97% WER. However, the [58] model is currently not suitable for production. Vosk is an inference abstraction built on Kaldi [60], an open-source speech recognition toolkit. Its best model achieves 7.08% WER on LibriSpeech test-clean.

3 Methodology

3.1 Slide Classification Model

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`, and `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 [33, 82, 81, 35, 37, 75, 42], but only report results from ResNet and EfficientNet models due to their superior accuracies.

3.1.1 Dataset

We compiled a dataset containing 15599 images extracted from 78 videos and manually classified them into 9 classes. The lecture videos were assembled from open-access university websites such as MIT OpenCourseWare and Open Yale Courses. All videos are licensed under Creative Commons, thus allowing us to freely distribute the final dataset.

The most important classes are `slide` and `presenter_slide`, which refer to slides from a screen capture and slides recorded by a video camera respectively. Frames from the `presenter_slide` are automatically perspective cropped while those from `slide` are not. The `slide` class may contain slides recorded by a video camera if the frame containing the slide is filled up nearly completely with the slide. These frames cannot be perspective cropped (see Section 3.4) and thus are classified as `slide` frames.

3.1.2 Dataset Processing

To create the dataset of lecture video frames we started by scraping the video hosting websites for metadata about the lectures. We selected videos at random from our sources and discarded any low quality videos. The model was bootstrapped to aid in the manual classification of extracted frames.

The number of frames extracted from each downloaded depends on the duration of the video to ensure that the dataset can be reconstructed from the original videos. We do not extract every frame because many frames contain very similar content, which is not useful when training the convolutional neural network (CNN). The default is to extract 200 frames. If the video is shorter than 20 minutes, 100 frames are extracted. If the video is longer than 80 minutes, 300 frames are extracted.

Once extracted, the frames are classified either manually, automatically, or from a classification sort mapping file. Automatic classification was used when we had already trained the model on some data: The model classified frames and automatically sorted them. Then, the model's mistakes were manually corrected. Once the final dataset was created, we produced a dictionary file that maps each extracted frame to a class, thus allowing the dataset to be identically sorted given the same input data.

Mass Data Collection To quickly and effectively gather additional data to train the slide classification model, we devised an approach to make the model more robust with minimal effort. We only manually sorted videos that the model was most unsure about. By selected videos that the model struggles with, it can learn fast while being exposed to the most unique situations. We accomplished this by scraping hundreds of videos, downloading them, and classifying extracted frames using a trained slide classification model. The videos with the lowest certainty (calculated by averaging the prediction probabilities for the correct class) were then manually sorted by fixing the model's mistakes and used to retrain the model.

3.1.3 Model Architecture

After testing several architectures, we chose ResNet-34 [33] 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:

ResNet	EfficientNet

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

Table 1: ResNet and EfficientNet architecture modifications.

3.1.4 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 [14, 13] provided by Optuna [8]. 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.

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

Table 2: Performance of the 4 model configurations on the testing set.

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.

Additionally, we investigated the result shown in Figure 3e. 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, as shown in Figure 3e, 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.

3.1.5 Model Results

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.

For each of the 4 model configurations, we trained 11 models and report the average metrics in Table 2. We report results from the median model (by accuracy) in Table 3 and Figures 1, 2, and 3.

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

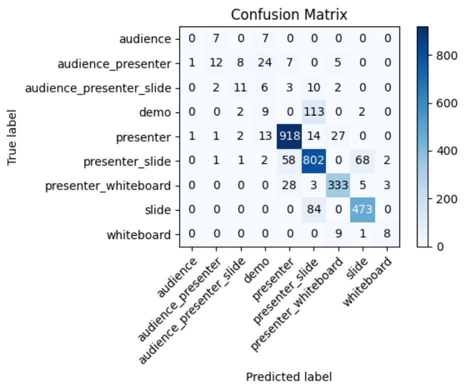
Class Name	Precision	Recall	F1-Score	Support
audience	0.00	0.00	0.00	14
audience_presenter	0.52	0.21	0.30	57
audience_presenter_slide	0.46	0.32	0.38	34
demo	0.15	0.07	0.10	126
presenter	0.91	0.94	0.92	976
presenter_slide	0.78	0.86	0.82	934
presenter_whiteboard	0.89	0.90	0.89	372
slide	0.86	0.85	0.86	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

(a) Final-general model

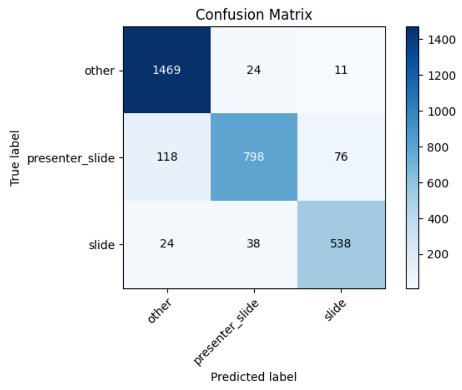
Class Name	Precision	Recall	F1-Score	Support
other	0.91	0.98	0.94	1504
presenter_slide	0.93	0.80	0.86	992
slide	0.86	0.90	0.88	600
accuracy	–	–	0.91	3096
macro avg	0.90	0.89	0.89	3096
weighted avg	0.91	0.91	0.90	3096

(b) Three-category model

Table 3: Classification reports for median (by accuracy) non-squished models.



(a) Final-general



(b) Three-category

Figure 1: The confusion matrices of the median final-general (a) and three-category (b) models.

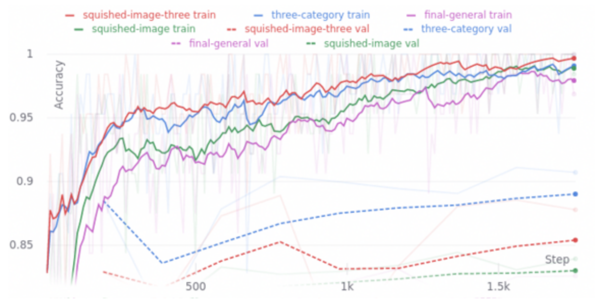
incorrectly. Incorrectly classifying `slide` frames as `presenter_slide` will have minimal impact on the final summary. Incorrectly classifying `presenter_slide` frames as `slide` will impact the final summary because they will not receive the correct processing. Incorrectly classifying `presenter_slide` as `presenter` represents a possible loss of information, but this is unlikely due to the same slide appearing in multiple frames.

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.

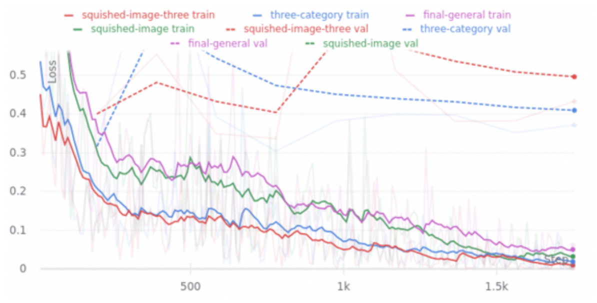
3.2 End-To-End Process

3.2.1 Overall Explanation

First, frames are extracted once every second. Each frame is classified using the slide classifier (see Section 3.1). Next, frames that were classified as `slide` are processed by a black border removal algorithm, which is a simple program that crops to the largest rectangle in an image if the edge pixel values of the image are all close to zero. Thus,

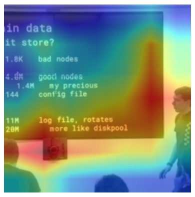


(a) Train & validation accuracy

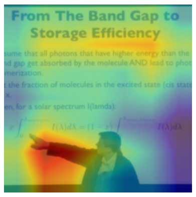


(b) Train & validation loss

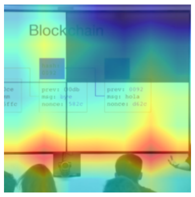
Figure 2: The accuracy (a) and loss (b) graphs of the 4 median (by accuracy) models.



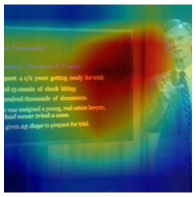
(a) Identified edges of screen displaying slide.



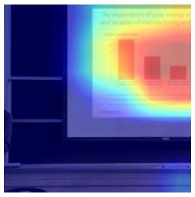
(b) Identified presenter pointing at slide content.



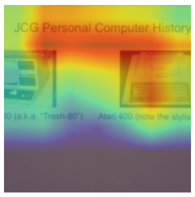
(c) Identified edges of screens with slides



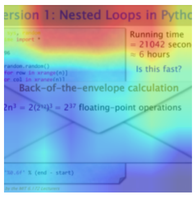
(d) Identified presenter, slide border, and slide content.



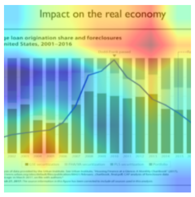
(e) Misclassified as slide because presenter cropped out.



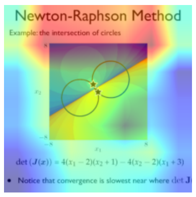
(f) Identifies title and images.



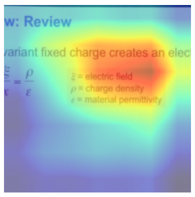
(g) Identifies title.



(h) Identifies part of title and chart.



(i) Identifies title and figure.



(j) Misclassified as presenter_slide.

Figure 3: Gradient-weighted Class Activation Mapping (Grad-CAM) [71] visualization of selected presenter_slide (a-e) and slide (f-j) images. Computed using the median final-general model. Image J is taken from a camera and treating it as a presenter_slide is not an issue since the frame only contains the slide. The problem showcased in image E is solved by the squished-image model.

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 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.

Model	WER	MER	WIL	LS TC WER [1]	Processing Time
DeepSpeech (chunking)	43.01	41.82	59.04	5.97 [1]	≈4 hours
DeepSpeech	44.44	42.98	59.99	5.97 [1]	≈20 hours
Google STT ("default" model)	34.43	33.14	49.05	12.23 [6]	≈20 minutes [2]
Wav2Vec2	39.38	36.27	54.43	2.60 [10]	≈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 [2]	≈8.5 hours
Vosk daanzu-20200905	33.67	31.87	48.28	7.08 [2]	≈5.5 hours
Vosk aspire-0.2	41.38	38.44	56.35	13.64 [2]	≈19 hours
Vosk aspire-0.2 (chunking)	41.45	38.65	56.56	13.64 [2]	≈19 hours

Table 4: Average statistics of various models on 43 lecture video dataset. Google STT is significantly faster since it transcribed the files in parallel as opposed to sequentially. VAD chunking was not used unless indicated. The "Vosk daanzu-20200905" model is from the Kaldi Active Grammar [95] project.

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 Section 3.5.

3.2.2 Speech-To-Text Implementation

In the final pipeline, the STT model takes an audio file and produces a textual representation. To improve the speed of the STT algorithm we implement two chunking strategies: `voice activity`, which uses the WebRTC VAD, and `noise activity`, which detects segments of audio that are significantly below the average loudness. Chunking increases the speed of STT by reducing the amount of audio without speech. Since `voice activity` not only removes silent sections but also removes noisy sections without words, it is the default option.

Some STT engines do not add punctuation. To solve this we use the DeepSegment [5] 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.

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.

Speech-To-Text 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.

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

[1]"LS TC WER" stands for LibriSpeech test-clean WER (Word Error Rate)
[2]Google STT ran in parallel.

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.

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.

3.2.3 Duplicate Image Removal

The system uses a variety of methods to remove duplicate slides and obtain a set of unique frames containing the 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 Section 3.2.4) and feature matching (see Section 3.4.1) algorithms.

3.2.4 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.

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.

Segment Method The `segment` clustering method iterates chronologically through extracted frames that have been filtered and transformed by the perspective cropping (see Section 3.4) 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.

3.2.5 Figure Extraction

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 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) [94] 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 [72].

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

3.3 Slide Structure Analysis

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 [76] 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 [26], 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. The distance transformation is calculated using the algorithm discussed in [25].

3.3.1 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.

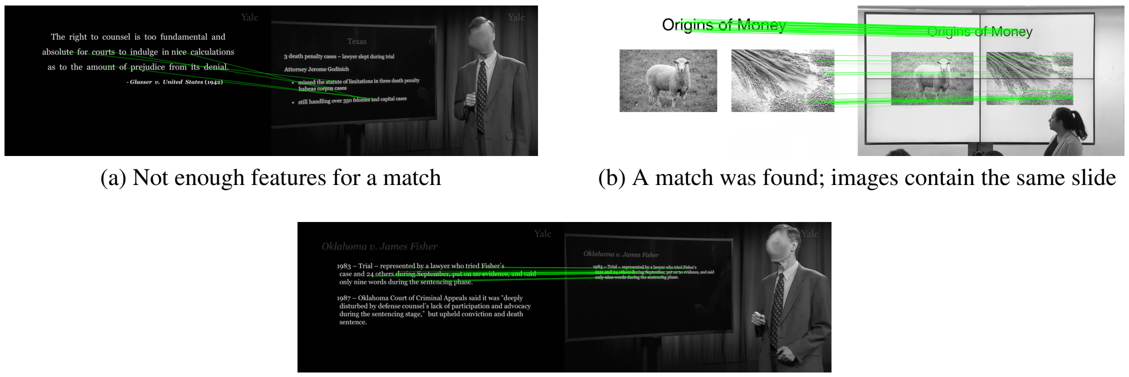
If there is only one block and paragraph then the slide might only contain the title. If this situation occurs, the stroke width and height checks are disabled because the averages of all content will only account for the title if these checks were left enabled.

3.4 Perspective Cropping

To improve the 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.

3.4.1 Feature Matching

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) [67] (which performs the same task as Scale Invariant Feature Transform (SIFT) [51] 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.


The right to counsel is too fundamental and absolute for courts to indulge in nice calculations as to the amount of prejudice from its denial.
- Glasser v. United States (1942)
Texas
Origins of Money
Origins of Money
Oklahoma v. James Fisher
Oklahoma v. James Fisher

(a) Not enough features for a match

(b) A match was found; images contain the same slide

(c) The image on the left has more content than the one on the right. The image with more content is kept.

Figure 4: Examples of feature matching algorithm. A green line is drawn between two identical features. Images on the left are of class slide and those on the right are of class presenter_slide.

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.

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 [30]. The Lucas Kanade optical flow [52] 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.

3.4.2 Corner Crop Transform

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 [22] to find horizontal and vertical lines, then find the intersection points, and finally cluster those points using KMeans.

3.5 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.

Structured Joined Summarization The structured joined summarization method is separate from the four-stage process. This method summarizes slides using the 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 Section 3.5.3) or abstractive summarization (see Section 3.5.4). 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.

3.5.1 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

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%.

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 [54]. 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.

3.5.2 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.

3.5.3 Extractive Summarization

There are three high-level extractive summarization methods: `cluster`, an advanced novel algorithm; `generic`, which is a collection of several standard extractive summarization algorithms such as TextRank, LSA, and Edmundson; and TransformerSum (see Section 4).

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_sbert`: Uses special BERT and RoBERTa models fine-tuned to extract sentence embeddings [66]. 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 Section 3.5.1) 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 [45].

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?" [56]. 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 Section 3.5) solves this problem.

The Generic Method There are six generic extractive summarization methods: `lsa` [78], `luhn` [53], `lex_rank` [24], `text_rank` [54], `edmundson` [23], and `random`. Random selects sentences from the input document at random.

3.5.4 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 BART [45], PEGASUS [92], T5 [64], PreSumm [49], or TransformerSum (see Section 4).

4 TransformerSum

TransformerSum is a summarization model training and inference framework developed as a part of this research. It also is a set of pre-trained abstractive and extractive summarization models.

Since lectures are longer than the 512 to 1024 token limit of state-of-the-art BERT-style pre-trained models, it is necessary to either use an extractive summarization heuristic or adapt neural-based summarization models to long sequences. We successfully perform the latter option using the Longformer [12] (extractive) and LED (abstractive), which is a combination of BART [45] and the Longformer. This research into long-sequence summarization is part of our TransformerSum library. TransformerSum also contains models that can run on resource-limited devices while still maintaining high levels of accuracy. These pre-trained models can be used in the lecture summarization system anytime a neural summarization model can be used. Through this library, we improve the state-of-the-art in long-sequence summarization and summarization on resource-limited devices.

4.1 Extractive Models

4.1.1 Converting a Dataset to Extractive

The "convert to extractive" program converts an abstractive summarization dataset to one that can be used to train extractive models. The program is able to convert manually downloaded and pre-processed datasets but can also automatically download, pre-process, and convert any dataset provided by the `huggingface/datasets` python package. This means `cnn_dailymail`, `scientific_papers`, `newsroom`, `reddit`, `multi_news`, `gigaword`, `billsum`, `wikihow`, `reddit_tifu`, `xsum`, and any future datasets supported by `huggingface/datasets` can easily be converted to the extractive task.

Following [47], we use a greedy algorithm that generates oracle summaries for each document by selecting the sentences which maximize the sum of the ROUGE-1 and ROUGE-2 scores. Sentences selected in the oracle summary are assigned label one and all other sentences are assigned label zero.

4.1.2 Architecture

The architecture of the extractive models is based on BertSum [47], which is a verion of BERT adapted for extractive summarization, and the improvements made in PreSumm [49]. PreSumm builds on top of BertSum by adding abstractive summarization. BertSum is extractive only.

Model Architecture Overview An input document is encoded and passed through an autoencoding transformer model called the word embedding model (WEM), which creates embeddings for each word. The output from the WEM goes through a pooling module, which compresses the token embeddings to sentence embeddings using a pooling strategy (see 4.1.2). Finally, a classifier reduces each sentence embedding to a single number representing the score of that sentence. The word embedding module and classification layers are jointly fine-tuned.

The pooling module is necessary in order to output a representation for each sentence and perform sentence-level summarization using an autoencoding transformer model. Autoencoding transformer models correspond to the encoder of the original transformer model [84] and are usually trained with the masked language modeling objective. This means the output vectors are grounded to tokens instead of sentences. While the separator token (`[SEP]` in BERT and `</s>` in RoBERTa) provides a representation of an individual sentence, it only applies to sentence-pair inputs (such as in question-answering tasks), while summarization models must encode multi-sentential inputs. We insert the separator token after each sentence to train the WEM to understand sentence boundaries when there are more than two sentences and use various pooling methods to obtain sentence embeddings.

Pooling Methods To obtain a representation for each sentence from the word vectors outputted by the WEM we use three pooling methods: `mean_tokens`, `sentence_rep_tokens`, and `max_tokens`.

The `sentence_rep_tokens` method is an abbreviation for "sentence representation tokens" and is nearly identical to the method used in PreSumm [49]. In this method, the classification token (`[CLS]` in BERT and `<s>` in RoBERTa) is inserted before each sentence (in addition to the separator token at the end). Each classification token collects features about the sentence after it and the separator tokens help to indicate to the model when the classification token should stop collecting features.

The `mean_tokens` pooling method uses the average of the token vectors for each sentence as the sentence embedding. [66] found that the mean of the token vectors produces slightly higher scores in the Semantic Textual Similarity (STS) benchmark [17]. An evaluation of sentence embeddings using the SentEval toolkit found a 0.28 percentage point difference between using the `[CLS]` token from BERT and taking the mean of the token vectors [66].

The `max_tokens` method uses the maximum of the token vectors for each sentence in the input as the sentence embedding. [66] found that this approach produces worse scores than the `sentence_rep_tokens` method on the development set of the STS benchmark, but we apply it to summarization since we hypothesize it may be better suited for summarization.

Segment Embeddings For WEMs that support token type IDs, which are also called segment embeddings, we use interval or sequential segment embeddings to distinguish multiple sentences in a document. These inputs are in addition to the separator token. For instance, RoBERTa does not use token type IDs and thus relies solely on the separator token. Interval sentence embeddings alternate between two values for each sentence in the input document. For $sentence_i$ we assign a segment embedding E_A or E_B depending on if i is even or odd. For example, for document $[sent_1, sent_2, sent_3, sent_4, sent_5]$, the model would assign embeddings $[E_A, E_B, E_A, E_B, E_A]$. Sequential segment embeddings are similar to interval embeddings but instead of alternating they sequentially increase. For example, for the same document, the model would assign embeddings $[E_A, E_B, E_C, E_D, E_F]$. However, the segment embedding vocabulary size of most pre-trained transformer models is two, so additional segment embeddings would need to be learned during fine-tuning or the model would need to be pre-trained with a larger vocabulary size.

We improve upon BertSum and the extractive component of PreSumm in several ways:

1. **Multiple encoder models supported**: Any transformer encoder model implemented in the `huggingface/transformers` python package [85] can be used as the encoder of the summarization model. This allows us to test BERT [21], RoBERTa [50], ALBERT [43], DistilBERT [68], DistilRoBERTa [68], MobileBERT [80], Longformer [12], and any future models implemented.
2. **Converting a dataset to extractive**: We designed a separate program capable of converting any abstractive summarization dataset to the extractive summarization task whereas PreSumm only provides code for converting CNN/DailyMail.
3. **Extractive pooling methods**: We created the `mean_tokens` and `max_tokens` pooling methods and support the `sentence_rep_tokens` method while BertSum only uses `sentence_rep_tokens`.
4. **Extractive classifiers**: BertSum supports three classifiers: a linear layer, a transformer, and an LSTM network. We support four classifiers as discussed in Section 4.1.3.

4.1.3 Summarization (Classifier) Layers

Summarization layers, which are referred to as classifier modules or simply classifiers, reduce sentence embeddings to a list that indicates the scores of the sentences. These layers are stacked at the end of the WEM and capture document-level features. We support four different classifiers. `linear` is a sequence consisting of a linear layer with a shape of the WEM hidden size by 1536, an activation function (GELU by default), a dropout layer with probability 0.1, another linear layer with 1536 by 1 parameters, and finally a sigmoid. `simple_linear` is a single linear layer and a sigmoid. `transformer` is a small transformer encoder followed by a linear layer and the sigmoid function. By default, the transformer has 8 attention heads, the feedforward dimension is 2048, the dropout probability is 0.1, and there are two encoder layers stacked on top of each other. `transformer_linear` is the `transformer` classifier followed by the `linear` classifier. Unlike BertSum, we do not test an LSTM classifier since [47] found it had little impact on ROUGE scores.

4.2 Extractive Training

All extractive models were trained for three epochs with gradient accumulation every two steps. We use the AdamW optimizer with $\beta_1 = 0.9$, $\beta_2 = 0.999$, and $\varepsilon = 1\text{e}{-8}$. Models were trained on one NVIDIA Tesla P100 GPU. Unless otherwise specified, the learning rate was $2\text{e}{-5}$ and a linear warmup with decay learning rate scheduler was used with 1400 steps of warmup. Except during their respective experiments, the `simple_linear` classifier and `sent_rep_tokens` pooling method are used. Gradients are clipped at 1.0 during training for all models. Model checkpoints were saved and evaluated on the validation set at the end of each epoch. We report the ROUGE scores on the test set of the model checkpoint saved after the final epoch.

4.2.1 Loss Function

The loss of the model is the binary classification entropy (BCE) of prediction $\hat{y}_i$ against ground-truth label y_i. The sum and mean reduction methods cannot be used because batches of input sequences vary in length and are padded to the length of the shortest sequence. Thus, the mean method would be lower simply based on the number of padding tokens, which are zeros, and sum method would be lower based on the length of the documents in a batch. To solve this, we implement five reduction methods:

1. `total`: Sum of the BCE of all sentences across all sequences. Larger batch sizes cause larger loss values.
2. `total_norm_batch`: `total` loss divded by the batch size.
3. `avg_seq_sum`: Sum of the average loss for each input sequence. Larger batch sizes cause larger loss values.
4. `avg_seq_mean`: `avg_seq_sum` divided by the batch size (average of averages).
5. `avg`: The `total` loss divided by the total number of non-padding loss values.

These custom reduction methods ignore padded values when performing calculations. The `avg_seq_mean` loss reduction method is used by default since it is not impacted by batch size and performed slightly better during testing.

4.2.2 Evaluation/Predicting

When predicting summaries for a document, we first use the model to obtain the score for each sentence. Then, these sentences are ranked by their scores from highest to lowest, and the top three sentences are selected as the summary. Alternatively, the sentences that have a rank above a threshold can be used to create the summary. During sentence selection we use trigram blocking to reduce redundancy [59, 49]. While selecting sentences to form a summary, trigram blocking will skip a sentence if it has a trigram that overlaps with the previously selected sentences [93]. The intuition is similar to Maximal Marginal Relevance [16]. In both cases, the goal is to minimize the similarity between the sentence being considered and sentences that have been already selected as part of the summary [49].

4.3 Experiments

4.3.1 Pooling Modes

We test all three pooling modes (`mean_tokens`, `sentence_rep_tokens`, and `max_tokens`) using DistilBERT and DistilRoBERTa, which are warm started from the `distilbert-base-uncased` and `distilroberta-base` checkpoints, respectively. We only test the `distil` models since they reach at least 95% of the performance of the original model while being significantly faster to train. The models were trained and tested on CNN/DailyMail, WikiHow, and ArXiv/PubMed to determine if certain methods worked better with certain topics. All models were trained with a batch size of 32 and the hyperparameters discussed in Section 4.2.

Across all datasets and models, the pooling mode has no significant impact on the final ROUGE scores. However, the `sent_rep` method usually performs slightly better. Additionally, the `mean` and `max` methods are about 30% slower than the `sent_rep` pooling method due to inefficiencies of nested loops.

Model	Pooling Method	CNN/DM	WikiHow	ArXiv/PubMed
distilbert-base-uncased	sent_rep	42.71/19.91/39.18	**30.69**/08.65/28.58	**34.93/12.21/31.00**
	mean	42.70/19.88/39.16	30.48/08.56/28.42	34.48/11.89/30.61
	max	42.74/19.90/39.17	30.51/08.62/28.43	34.50/11.91/30.62
distilroberta-base	sent_rep	42.87/20.02/39.31	31.07/08.96/28.95	**34.70/12.16/30.82**
	mean	43.00/20.08/39.42	30.96/08.93/28.86	34.24/11.82/30.42
	max	42.91/20.04/39.33	30.93/08.92/28.82	34.28/11.82/30.44

Table 5: Test set results using ROUGE F_1 of extractive summarization models with different pooling methods. The data in each cell is in the R1/R2/RL-Sum format. Sentence representation model results are identical to Table 7. Best ROUGE scores for each dataset and model greater than 0.15 more than the second best scoring pooling mode are bolded.

Model	Classifier	CNN/DM	WikiHow	ArXiv/PubMed
distilbert-base-uncased	simple_linear	42.71/19.91/39.18	30.69/08.65/28.58	34.93/12.21/31.00
	linear	42.70/19.84/39.14	30.67/08.62/28.56	34.87/12.20/30.96
	transformer	42.78/19.93/39.22	30.66/08.69/28.57	34.94/12.22/31.03
	transformer_linear	42.78/19.93/39.22	30.64/08.64/28.54	34.97/12.22/31.02
distilroberta-base	simple_linear	42.87/20.02/39.31	31.07/08.96/28.95	34.70/12.16/30.82
	linear	43.18/20.26/39.61	31.08/08.98/28.96	34.77/12.17/30.88
	transformer	42.94/20.03/39.37	31.05/08.97/28.93	34.77/12.17/30.87
	transformer_linear	42.90/20.00/39.34	31.13/09.01/29.02	34.77/12.18/30.88

Table 6: Test set results using ROUGE F_1 of extractive summarization models with different classifiers after the WEM. The data in each cell is in the R1/R2/RL-Sum format. The "simple_linear" results are identical to Table 7.

4.3.2 Classifier

We test all four summarization layers, including two variations of the `transformer` method for a total of five configurations, using the same models and datasets from the pooling modes experiment. For this experiment, a batch size of 32 was used.

There is no significant difference between the classifier used. Thus, we use the linear classifier by default since it contains fewer parameters.

4.4 Final Model Results

Table 7 shows the results of our trained summarization models on the CNN/DailyMail, WikiHow, and ArXiv/PubMed datasets. Note that in table 7 BertSum model results are average ROUGE scores on the test set of the top-3 checkpoints based on the validation loss.

Importantly, we do not test the large variants of BERT or RoBERTa due to computing resource limitations. Additionally, using them in production is infeasible. However, they can easily be trained for future research.

The `mobilebert-uncased-ext-sum` model achieves 96.59% of the performance of BertSum [47] while containing 4.45x fewer parameters. It achieves 94.06% of the performance of MatchSum [93], the current state-of-the-art in extractive summarization on CNN/DailyMail. The `distilroberta-base-ext-sum` model trains in about 6.5 hours on 1 NVIDIA Tesla P100 GPU, while MatchSum takes 30 hours on 8 Tesla V100 GPUs to train. If a V100 is about twice as powerful as a P100, then it would take 480 hours to train MatchSum on one P100. This simplistic approximation suggests that it takes about 74x more time to train MatchSum than `distilroberta-base-ext-sum`. The `distilroberta` model matches 92.7% of the performance of `roberta-base` on CNN/DailyMail.

R1/R2/RL-Sum	CNN/DM	WikiHow	ArXiv/PubMed
distilbert-base-uncased-ext-sum	42.71/19.91/39.18	30.69/08.65/28.58	34.93/12.21/31.00
distilroberta-base-ext-sum	42.87/20.02/39.31	31.07/08.96/28.95	34.70/12.16/30.82
bert-base-uncased-ext-sum	42.78/19.83/39.18	30.68/08.67/28.59	34.80/12.26/30.92
roberta-base-ext-sum	43.24/20.36/39.65	31.26/09.09/29.14	34.81/12.26/30.91
mobilebert-uncased-ext-sum	42.01/19.31/38.53	30.72/08.78/28.59	33.97/11.74/30.19
BertSumExt [49]	43.25/20.24/39.63	None	None
BertSumExt-large [49]	43.85/20.34/39.90	None	None
MatchSum bert-base [93]	44.22/20.62/40.38	31.85/08.98/29.58	None
MatchSum roberta-base [93]	**44.41**/20.86/40.55	None	None
PEGASUS-base [92]	41.79/18.81/38.93	36.58/15.64/30.01	37.39/12.66/23.87
PEGASUS-large HN [92]	44.17/**21.47**/**41.11**	41.35/18.51/33.42	44.88/18.37/26.58
PEGASUS-large C4 [92]	43.90/21.20/40.76	**43.06/19.71/34.80**	**45.10/18.59/26.75**

Table 7: Test set results using ROUGE F_1 of extractive summarization models. PEGASUS models are included to provide a comparison between extractive and abstractive summarization. All PEGASUS models are abstractive. "*-ext-sum" models are ours.

4.5 Abstractive Models

When summarizing a lecture, many pre-trained abstractive summarization models can be used (see Section 3.5.4). Here, we discuss the LED architecture, which can summarize long sequences. TransformerSum can be used to fine-tune any seq2seq included in the `huggingface/trasnformers` library [85].

Effect of Pre-training [12] computed the masked language modeling bits per character (BPC) on their training corpus and compared results across several model configurations. RoBERTa-base achieved a 1.846 BPC. The BPC of Longformer before pre-training with randomly initialized position embeddings was 10.299, but with position embeddings copied from RoBERTa, the BPC droped to 1.957. The relatively small difference between the RoBERTa BPC and the Longformer initialized BPC indicates that, without fine-tuning, the Longformer is capable of expanding a model's input sequence length without a substantial drop in performance. However, pre-training does have an impact: 2,000 gradient updates yielded 1.753 BPC and 65K 1.705 BPC. The large variants of RoBERTa and the Longformer show the same trends but with smaller scores. However, this is not true for summarization. BART fine-tuned for summarization and then expanded using the Longformer's attention layers without any gradient updates performs no better than random.

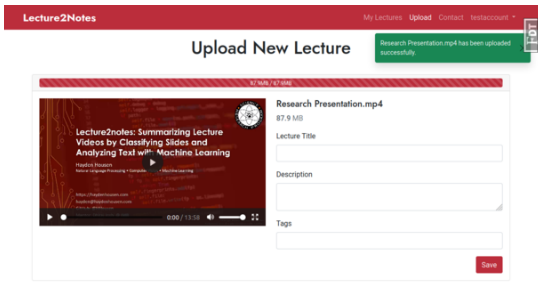
Future Research We do not fine-tune any abstractive summarization models and instead use the `led-large-16384-arxiv` checkpoint when abstractive summarization is requested by the user in the end-to-end process. We leave experimentation with various long sequence transformer architectures for abstractive summarization to future reserach. We released TransformerSum as open-souce to simplify and encourage research in this area.

5 Website

We created a website to allow educators and students to convert their own lectures to notes. The backend is powered by Flask, a micro web framework written in Python, and the frontend uses Bootstrap, a popular CSS framework. Celery, an asynchronous task queue, is used to control lecture summarization jobs launched by users. A tour of the website is displayed in 5.

6 Conclusion

We created lecture2notes, a state-of-the-art program that converts lecture videos to notes using machine learning. We compiled a large dataset of images from lecture videos to accelerate research in lecture video data mining. We trained a model, which can be used in future research to extract specific information, that identifies important frames


Lecture2Notes

My Lectures Upload Contact testaccount

Upload New Lecture

Research Presentation.mp4 has been uploaded successfully.

Research Presentation.mp4

87.9 MB

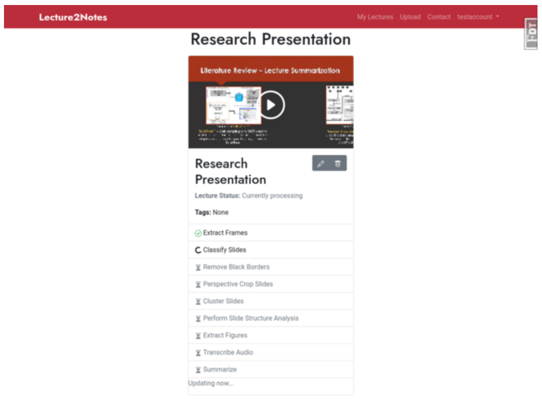
Lecture Title

Description

Tags

Save

(a) A user successfully uploads their lecture video.


Lecture2Notes

My Lectures Upload Contact testaccount

Research Presentation

Research Presentation

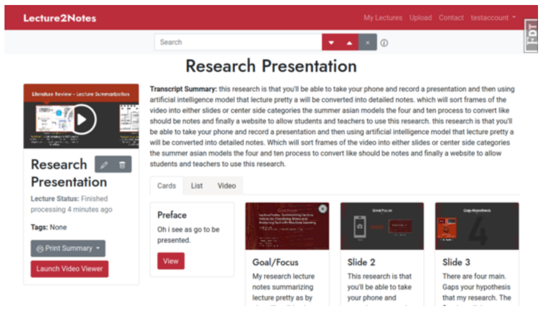
Lecture Status: Currently processing

Tags: None

- Extract Frames
- Classify Slides
- Remove Black Borders
- Perspective Crop Slides
- Cluster Slides
- Perform Slide Structure Analysis
- Extract Figures
- Transcribe Audio
- Summarize

Updating now...

(b) The lecture video is processed in a series of steps.


Lecture2Notes

My Lectures Upload Contact testaccount

Research Presentation

Transcript Summary: this research is that you'll be able to take your phone and record a presentation and then using artificial intelligence model that lecture pretty a will be converted into detailed notes. which will sort frames of the video into either slides or center side categories the summer asian models the four and ten process to convert like should be notes and finally a website to allow students and teachers to use this research. this research is that you'll be able to take your phone and record a presentation and then using artificial intelligence model that lecture pretty a will be converted into detailed notes. Which will sort frames of the video into either slides or center side categories the summer asian models the four and ten process to convert like should be notes and finally a website to allow students and teachers to use this research.

Research Presentation

Lecture Status: Finished processing 4 minutes ago

Tags: None

Print Summary

Launch Video Viewer

Cards List Video

Preface

Oh i see as go to be presented.

View

Goal/Focus

My research lecture notes summarizing lecture pretty as by

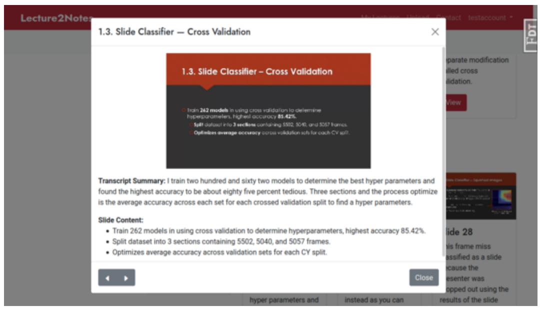
Slide 2

This research is that you'll be able to take your phone and

Slide 3

There are four main. Gaps your hypothesis that my research. The

(c) The lecture summary is shown in many views.


1.3. Slide Classifier — Cross Validation

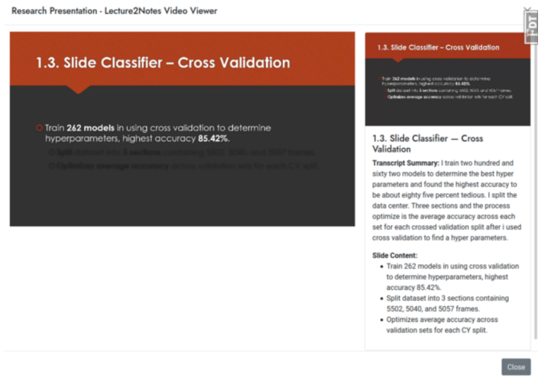
Transcript Summary: I train two hundred and sixty two models to determine the best hyper parameters and found the highest accuracy to be about eighty five percent tedious. Three sections and the process optimize is the average accuracy across each set for each crossed validation split to find a hyper parameters.

Slide Content:

- Train 262 models in using cross validation to determine hyperparameters, highest accuracy 85.42%.
- Split dataset into 3 sections containing 5502, 5040, and 5057 frames.
- Optimizes average accuracy across validation sets for each CY split.

Close

(d) An individual slide can be clicked for a summary of that slide.


Research Presentation - Lecture2Notes Video Viewer

1.3. Slide Classifier – Cross Validation

Train **262 models** in using cross validation to determine hyperparameters, highest accuracy **85.42%**.

1.3. Slide Classifier — Cross Validation

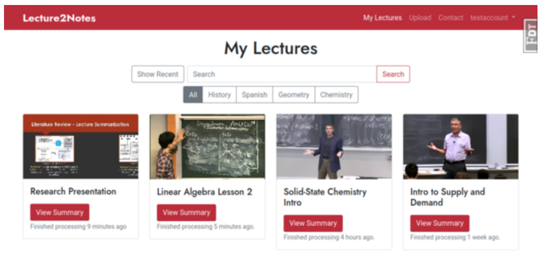
Transcript Summary: I train two hundred and sixty two models to determine the best hyper parameters and found the highest accuracy to be about eighty five percent tedious. I split the data center. Three sections and the process optimize is the average accuracy across each set for each crossed validation split after i used cross validation to find a hyper parameters.

Slide Content:

- Train 262 models in using cross validation to determine hyperparameters, highest accuracy 85.42%.
- Split dataset into 3 sections containing 5502, 5040, and 5057 frames.
- Optimizes average accuracy across validation sets for each CY split.

Close

(e) The video can be watched and the slide summary relevant to the portion being watched will be displayed.


Lecture2Notes

My Lectures Upload Contact testaccount

My Lectures

Show Recent Search Search

All History Spanish Geometry Chemistry

Research Presentation	Linear Algebra Lesson 2	Solid-State Chemistry Intro	Intro to Supply and Demand
View Summary	View Summary	View Summary	View Summary
Finished processing 9 minutes ago	Finished processing 5 minutes ago	Finished processing 4 hours ago	Finished processing 1 week ago

(f) The list of lectures uploaded.

Figure 5: An example form the user's perspective of the process of converting a lecture to notes using the website.

in lecture videos. This novel model is robust to various environments and achieves state-of-the-art accuracy. We also advanced the state-of-the-art in automatic text summarization by proposing an architecture capable of summarizing long sequences and by designing models that require little computational power both to train and perform inference. Long sequence summarization models will achieve great results in many real-world tasks since many texts are too long to be processed by previously existing methods. To enable easier access to neural summarization research, we have released TransformerSum, which includes our novel architectures and processing scripts, to the community. Our final end-to-end process that summarizes lecture videos will increase content knowledge, enable faster learning by allowing students to preview materials, and present the material differently for those who learn best by reading instead of listening, such as special needs learners. For future research, we will train the proposed large summarization models to add another summarization method to lecture2notes. We also will publish the website so teachers and students can benefit from our software.

References

[1] Release DeepSpeech 0.7.1 · mozilla/DeepSpeech, . URL /mozilla/DeepSpeech/releases/tag/v0.7.1.

[2] VOSK Models, . URL https://alphacephei.com/vosk/models.

[3] WebRTC, . URL https://webrtc.org/.

[4] alphacep/vosk-api, September 2020. URL https://github.com/alphacep/vosk-api. original-date: 2019-09-03T17:48:42Z.

[5] notAI-tech/deepsegment, September 2020. URL https://github.com/notAI-tech/deepsegment. original-date: 2018-11-15T12:06:14Z.

[6] Picovoice/speech-to-text-benchmark, September 2020. URL https://github.com/Picovoice/speech-to-text-benchmark. original-date: 2018-08-04T02:52:01Z.

[7] John Adcock, Matthew Cooper, Laurent Denoue, Hamed Pirsiavash, and Lawrence A. Rowe. TalkMiner: a lecture webcast search engine. In *Proceedings of the international conference on Multimedia - MM '10*, page 241, Firenze, Italy, 2010. ACM Press. ISBN 978-1-60558-933-6. doi: 10.1145/1873951.1873986. URL http://dl.acm.org/citation.cfm?doid=1873951.1873986.

[8] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A Next-generation Hyperparameter Optimization Framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining - KDD '19*, pages 2623–2631, Anchorage, AK, USA, 2019. ACM Press. ISBN 978-1-4503-6201-6. doi: 10.1145/3292500.3330701. URL http://dl.acm.org/citation.cfm?doid=3292500.3330701.

[9] Jennifer L Austin, Matthew D Thibeault, and Jon S Bailey. Effects of Guided Notes on University Students' Responding and Recall of Information. *Journal of Behavioral Education*, page 12, 2002.

[10] Alexei Baevski, Henry Zhou, Abdelrahman Mohamed, and Michael Auli. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations. *arXiv:2006.11477 [cs, eess]*, October 2020. URL http://arxiv.org/abs/2006.11477. arXiv: 2006.11477.

[11] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural Machine Translation by Jointly Learning to Align and Translate. *arXiv:1409.0473 [cs, stat]*, May 2016. URL http://arxiv.org/abs/1409.0473. arXiv: 1409.0473.

[12] Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The Long-Document Transformer. *arXiv:2004.05150 [cs]*, April 2020. URL http://arxiv.org/abs/2004.05150. arXiv: 2004.05150.

[13] J Bergstra, D Yamins, and D D Cox. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. page 9, .

[14] James S Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for Hyper-Parameter Optimization. page 9, .

[15] Lori Breslow, David Pritchard, Jennifer DeBoer, Glenda Stump, Andrew Ho, and Daniel Seaton. Studying Learning in the Worldwide Classroom: Research into edX's First MOOC. *Research in Practice and Assessment*, 2013.

[16] Jaime Carbonell and Jade Goldstein. The use of MMR, diversity-based reranking for reordering documents and producing summaries. In *Proceedings of the 21st annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '98, pages 335–336, New York, NY, USA, August 1998. Association for Computing Machinery. ISBN 978-1-58113-015-7. doi: 10.1145/290941.291025. URL `https://doi.org/10.1145/290941.291025`.

[17] Daniel Cer, Mona Diab, Eneko Agirre, Iñigo Lopez-Gazpio, and Lucia Specia. SemEval-2017 Task 1: Semantic Textual Similarity Multilingual and Crosslingual Focused Evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, pages 1–14, Vancouver, Canada, August 2017. Association for Computational Linguistics. doi: 10.18653/v1/S17-2001. URL `https://www.aclweb.org/anthology/S17-2001`.

[18] Wan-Chen Chang and Yu-Min Ku. The Effects of Note-Taking Skills Instruction on Elementary Students' Reading. *The Journal of Educational Research*, 108(4):278–291, July 2015. ISSN 0022-0671, 1940-0675. doi: 10.1080/00220671.2014.886175. URL `http://www.tandfonline.com/doi/full/10.1080/00220671.2014.886175`.

[19] Rewon Child, Scott Gray, Alec Radford, and Ilya Sutskever. Generating Long Sequences with Sparse Transformers. *arXiv:1904.10509 [cs, stat]*, April 2019. URL `http://arxiv.org/abs/1904.10509`. arXiv: 1904.10509.

[20] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-XL: Attentive Language Models Beyond a Fixed-Length Context. *arXiv:1901.02860 [cs, stat]*, June 2019. URL `http://arxiv.org/abs/1901.02860`. arXiv: 1901.02860.

[21] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL `http://arxiv.org/abs/1810.04805`. arXiv: 1810.04805.

[22] Richard O. Duda and Peter E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, January 1972. ISSN 0001-0782. doi: 10.1145/361237.361242. URL `https://doi.org/10.1145/361237.361242`.

[23] H. P. Edmundson. New Methods in Automatic Extracting. *Journal of the ACM*, 16(2):264–285, April 1969. ISSN 0004-5411. doi: 10.1145/321510.321519. URL `https://doi.org/10.1145/321510.321519`.

[24] Gunes Erkan and Dragomir R. Radev. LexRank: Graph-based Lexical Centrality as Salience in Text Summarization. *Journal of Artificial Intelligence Research*, 22:457–479, December 2004. ISSN 1076-9757. doi: 10.1613/jair.1523. URL `http://arxiv.org/abs/1109.2128`. arXiv: 1109.2128.

[25] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Distance Transforms of Sampled Functions. *Theory of Computing*, 8(1): 415–428, 2012. ISSN 1557-2862. doi: 10.4086/toc.2012.v008a019. URL `http://www.theoryofcomputing.org/articles/v008a019`.

[26] Wolf Garbe. wolfgarbe/SymSpell, July 2020. URL `https://github.com/wolfgarbe/SymSpell`. original-date: 2014-03-25T11:01:35Z.

[27] Anneh Mohammad Gharravi. Impact of instructor-provided notes on the learning and exam performance of medical students in an organ system-based medical curriculum. *Advances in Medical Education and Practice*, Volume 9:665–672, September 2018. ISSN 1179-7258. doi: 10.2147/AMEP.S172345. URL `https://www.dovepress.com/impact-of-instructor-provided-notes-on-the-learning-and-exam-performan-peer-reviewed-article-AMEP`.

[28] Philip J. Guo, Juho Kim, and Rob Rubin. How video production affects student engagement: an empirical study of MOOC videos. In *Proceedings of the first ACM conference on Learning @ scale conference - L@S '14*, pages 41–50, Atlanta, Georgia, USA, 2014. ACM Press. ISBN 978-1-4503-2669-8. doi: 10.1145/2556325.2566239. URL `http://dl.acm.org/citation.cfm?doid=2556325.2566239`.

[29] Awni Hannun, Carl Case, Jared Casper, Bryan Catanzaro, Greg Diamos, Erich Elsen, Ryan Prenger, Sanjeev Satheesh, Shubho Sengupta, Adam Coates, and Andrew Y. Ng. Deep Speech: Scaling up end-to-end speech recognition. *arXiv:1412.5567 [cs]*, December 2014. URL `http://arxiv.org/abs/1412.5567`. arXiv: 1412.5567.

[30] C. Harris and M. Stephens. A Combined Corner and Edge Detector. In *Procedings of the Alvey Vision Conference 1988*, pages 23.1–23.6, Manchester, 1988. Alvey Vision Club. doi: 10.5244/C.2.23. URL `http://www.bmva.org/bmvc/1988/avc-88-023.html`.

[31] James Hartley. Lecture Handouts and Student Note-taking. *Programmed Learning and Educational Technology*, 13(2):58–64, May 1976. ISSN 0033-0396. doi: 10.1080/1355800760130208. URL `https://www.tandfonline.com/doi/full/10.1080/1355800760130208`.

[32] Todd Haydon, G. Richmond Mancil, Stephen D. Kroeger, James McLeskey, and Wan-Yu Jenny Lin. A Review of the Effectiveness of Guided Notes for Students who Struggle Learning Academic Content. *Preventing School Failure: Alternative Education for Children and Youth*, 55(4):226–231, August 2011. ISSN 1045-988X, 1940-4387. doi: 10.1080/1045988X.2010.548415. URL `http://www.tandfonline.com/doi/abs/10.1080/1045988X.2010.548415`.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. URL `http://arxiv.org/abs/1512.03385`. arXiv: 1512.03385.

[34] Karl Moritz Hermann, Tomáš Kočiský, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching Machines to Read and Comprehend. *arXiv:1506.03340 [cs]*, November 2015. URL `http://arxiv.org/abs/1506.03340`. arXiv: 1506.03340.

[35] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, January 2018. URL `http://arxiv.org/abs/1608.06993`. arXiv: 1608.06993.

[36] D. Huggins-Daines, M. Kumar, A. Chan, A.W. Black, M. Ravishankar, and A.I. Rudnicky. Pocketsphinx: A Free, Real-Time Continuous Speech Recognition System for Hand-Held Devices. In *2006 IEEE International Conference on Acoustics Speed and Signal Processing Proceedings*, volume 1, pages I–185–I–188, Toulouse, France, 2006. IEEE. ISBN 978-1-4244-0469-8. doi: 10.1109/ICASSP.2006.1659988. URL `http://ieeexplore.ieee.org/document/1659988/`.

[37] Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5MB model size. *arXiv:1602.07360 [cs]*, November 2016. URL `http://arxiv.org/abs/1602.07360`. arXiv: 1602.07360.

[38] Kenneth A. Kiewra. Providing the Instructor's Notes: An Effective Addition to Student Notetaking. *Educational Psychologist*, 20(1):33–39, January 1985. ISSN 0046-1520, 1532-6985. doi: 10.1207/s15326985ep2001_5. URL `https://www.tandfonline.com/doi/full/10.1207/s15326985ep2001_5`.

[39] Kenneth A. Kiewra. How Classroom Teachers Can Help Students Learn and Teach Them How to Learn. *Theory Into Practice*, 41(2):71–80, May 2002. ISSN 0040-5841, 1543-0421. doi: 10.1207/s15430421tip4102_3. URL `http://www.tandfonline.com/doi/abs/10.1207/s15430421tip4102_3`.

[40] Nikita Kitaev, Łukasz Kaiser, and Anselm Levskaya. Reformer: The Efficient Transformer. *arXiv:2001.04451 [cs, stat]*, January 2020. URL `http://arxiv.org/abs/2001.04451`. arXiv: 2001.04451.

[41] René F. Kizilcec, Chris Piech, and Emily Schneider. Deconstructing disengagement: analyzing learner subpopulations in massive open online courses. In *Proceedings of the Third International Conference on Learning Analytics and Knowledge - LAK '13*, page 170, Leuven, Belgium, 2013. ACM Press. ISBN 978-1-4503-1785-6. doi: 10.1145/2460296.2460330. URL `http://dl.acm.org/citation.cfm?doid=2460296.2460330`.

[42] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6):84–90, May 2017. ISSN 0001-0782, 1557-7317. doi: 10.1145/3065386. URL `https://dl.acm.org/doi/10.1145/3065386`.

[43] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. *arXiv:1909.11942 [cs]*, February 2020. URL `http://arxiv.org/abs/1909.11942`. arXiv: 1909.11942.

[44] Greg C. Lee, Fu-Hao Yeh, Ying-Ju Chen, and Tao-Ku Chang. Robust handwriting extraction and lecture video summarization. *Multimedia Tools and Applications*, 76(5):7067–7085, March 2017. ISSN 1380-7501, 1573-7721. doi: 10.1007/s11042-016-3353-y. URL `http://link.springer.com/10.1007/s11042-016-3353-y`.

[45] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv:1910.13461 [cs, stat]*, October 2019. URL `http://arxiv.org/abs/1910.13461`. arXiv: 1910.13461.

[46] Peter J. Liu, Mohammad Saleh, Etienne Pot, Ben Goodrich, Ryan Sepassi, Lukasz Kaiser, and Noam Shazeer. Generating Wikipedia by Summarizing Long Sequences. *arXiv:1801.10198 [cs]*, January 2018. URL `http://arxiv.org/abs/1801.10198`. arXiv: 1801.10198 version: 1.

[47] Yang Liu. Fine-tune BERT for Extractive Summarization. *arXiv:1903.10318 [cs]*, September 2019. URL `http://arxiv.org/abs/1903.10318`. arXiv: 1903.10318.

[48] Yang Liu and Mirella Lapata. Hierarchical Transformers for Multi-Document Summarization. *arXiv:1905.13164 [cs]*, May 2019. URL `http://arxiv.org/abs/1905.13164`. arXiv: 1905.13164.

[49] Yang Liu and Mirella Lapata. Text Summarization with Pretrained Encoders. *arXiv:1908.08345 [cs]*, September 2019. URL `http://arxiv.org/abs/1908.08345`. arXiv: 1908.08345.

[50] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *arXiv:1907.11692 [cs]*, July 2019. URL `http://arxiv.org/abs/1907.11692`. arXiv: 1907.11692.

[51] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2): 91–110, November 2004. ISSN 0920-5691. doi: 10.1023/B:VISI.0000029664.99615.94. URL `http://link.springer.com/10.1023/B:VISI.0000029664.99615.94`.

[52] Bruce D Lucas and Takeo Kanade. An Iterative Image Registration Technique with an Application to Stereo Vision. page 10, 1981.

[53] H. P. Luhn. The Automatic Creation of Literature Abstracts. *IBM Journal of Research and Development*, 2(2):159–165, April 1958. ISSN 0018-8646. doi: 10.1147/rd.22.0159. Conference Name: IBM Journal of Research and Development.

[54] Rada Mihalcea and Paul Tarau. TextRank: Bringing Order into Texts. page 8.

[55] Derek Miller. Leveraging BERT for Extractive Text Summarization on Lectures. page 7.

[56] Shashi Narayan, Shay B. Cohen, and Mirella Lapata. Don't Give Me the Details, Just the Summary! Topic-Aware Convolutional Neural Networks for Extreme Summarization. *arXiv:1808.08745 [cs]*, August 2018. URL `http://arxiv.org/abs/1808.08745`. arXiv: 1808.08745.

[57] Pauline A. Nye, Terence J. Crooks, Melanie Powley, and Gail Tripp. Student note-taking related to university examination performance. *Higher Education*, 13(1):85–97, February 1984. ISSN 0018-1560, 1573-174X. doi: 10.1007/BF00136532. URL `http://link.springer.com/10.1007/BF00136532`.

[58] Daniel S. Park, Yu Zhang, Ye Jia, Wei Han, Chung-Cheng Chiu, Bo Li, Yonghui Wu, and Quoc V. Le. Improved Noisy Student Training for Automatic Speech Recognition. *arXiv:2005.09629 [cs, eess]*, May 2020. URL `http://arxiv.org/abs/2005.09629`. arXiv: 2005.09629.

[59] Romain Paulus, Caiming Xiong, and Richard Socher. A Deep Reinforced Model for Abstractive Summarization. *arXiv:1705.04304 [cs]*, November 2017. URL `http://arxiv.org/abs/1705.04304`. arXiv: 1705.04304.

[60] Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukasˇ Burget, Ondˇrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlıˇcek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The Kaldi Speech Recognition Toolkit. page 4.

[61] Jiezhong Qiu, Hao Ma, Omer Levy, Scott Wen-tau Yih, Sinong Wang, and Jie Tang. Blockwise Self-Attention for Long Document Understanding. *arXiv:1911.02972 [cs]*, November 2019. URL `http://arxiv.org/abs/1911.02972`. arXiv: 1911.02972.

[62] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. page 24.

[63] Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive Transformers for Long-Range Sequence Modelling. *arXiv:1911.05507 [cs, stat]*, November 2019. URL `http://arxiv.org/abs/1911.05507`. arXiv: 1911.05507.

[64] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *arXiv:1910.10683 [cs, stat]*, October 2019. URL `http://arxiv.org/abs/1910.10683`. arXiv: 1910.10683.

[65] Mina Rahmani and Karim Sadeghi. Effects of Note-Taking Training on Reading Comprehension and Recall. page 13.

[66] Nils Reimers and Iryna Gurevych. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. *arXiv:1908.10084 [cs]*, August 2019. URL `http://arxiv.org/abs/1908.10084`. arXiv: 1908.10084.

[67] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571, Barcelona, Spain, November 2011. IEEE. ISBN 978-1-4577-1102-2 978-1-4577-1101-5 978-1-4577-1100-8. doi: 10.1109/ICCV.2011.6126544. URL `http://ieeexplore.ieee.org/document/6126544/`.

[68] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv:1910.01108 [cs]*, February 2020. URL `http://arxiv.org/abs/1910.01108`. arXiv: 1910.01108.

[69] Daniel T. Seaton, Yoav Bergner, Isaac Chuang, Piotr Mitros, and David E. Pritchard. Who does what in a massive open online course? *Communications of the ACM*, 57(4):58–65, April 2014. ISSN 00010782. doi: 10.1145/2500876. URL `http://dl.acm.org/citation.cfm?doid=2580723.2500876`.

[70] Abigail See, Peter J. Liu, and Christopher D. Manning. Get To The Point: Summarization with Pointer-Generator Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1073–1083, Vancouver, Canada, July 2017. Association for Computational Linguistics. doi: 10.18653/v1/P17-1099. URL `https://www.aclweb.org/anthology/P17-1099`.

[71] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, 128(2):336–359, February 2020. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-019-01228-7. URL `http://arxiv.org/abs/1610.02391`. arXiv: 1610.02391.

[72] C E Shannon. A Mathematical Theory of Communication. *The Bell System Technical Journal*, 27:379–423, 623–656, 1948.

[73] Atsushi Shimada, Fumiya Okubo, Chengjiu Yin, and Hiroaki Ogata. Automatic Summarization of Lecture Slides for Enhanced Student PreviewTechnical Report and User Study. *IEEE Transactions on Learning Technologies*, 11(2):165–178, April 2018. ISSN 1939-1382, 2372-0050. doi: 10.1109/TLT.2017.2682086. URL `https://ieeexplore.ieee.org/document/7879355/`.

[74] Hijung Valentina Shin, Floraine Berthouzoz, Wilmot Li, and Frédo Durand. Visual transcripts: lecture notes from blackboard-style lecture videos. *ACM Transactions on Graphics*, 34(6):1–10, October 2015. ISSN 07300301. doi: 10.1145/2816795.2818123. URL `http://dl.acm.org/citation.cfm?doid=2816795.2818123`.

[75] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, April 2015. URL `http://arxiv.org/abs/1409.1556`. arXiv: 1409.1556.

[76] R. Smith. An Overview of the Tesseract OCR Engine. In *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007) Vol 2*, pages 629–633, Curitiba, Parana, Brazil, September 2007. IEEE. ISBN 978-0-7695-2822-9. doi: 10.1109/ICDAR.2007.4376991. URL `http://ieeexplore.ieee.org/document/4376991/`.

[77] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. MASS: Masked Sequence to Sequence Pre-training for Language Generation. *arXiv:1905.02450 [cs]*, June 2019. URL `http://arxiv.org/abs/1905.02450`. arXiv: 1905.02450.

[78] Josef Steinberger and Karel Ježek. Using Latent Semantic Analysis in Text Summarization and Summary Evaluation. page 8, 2004.

[79] Sainbayar Sukhbaatar, Edouard Grave, Piotr Bojanowski, and Armand Joulin. Adaptive Attention Span in Transformers. *arXiv:1905.07799 [cs, stat]*, August 2019. URL `http://arxiv.org/abs/1905.07799`. arXiv: 1905.07799.

[80] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. MobileBERT: a Compact Task-Agnostic BERT for Resource-Limited Devices. *arXiv:2004.02984 [cs]*, April 2020. URL `http://arxiv.org/abs/2004.02984`. arXiv: 2004.02984.

[81] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper with Convolutions. *arXiv:1409.4842 [cs]*, September 2014. URL `http://arxiv.org/abs/1409.4842`. arXiv: 1409.4842.

[82] Mingxing Tan and Quoc V. Le. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. *arXiv:1905.11946 [cs, stat]*, November 2019. URL `http://arxiv.org/abs/1905.11946`. arXiv: 1905.11946.

[83] Yi Tay, Mostafa Dehghani, Samira Abnar, Yikang Shen, Dara Bahri, Philip Pham, Jinfeng Rao, Liu Yang, Sebastian Ruder, and Donald Metzler. Long Range Arena: A Benchmark for Efficient Transformers. November 2020. URL `https://arxiv.org/abs/2011.04006v1`.

[84] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL `http://arxiv.org/abs/1706.03762`. arXiv: 1706.03762.

[85] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.emnlp-demos.6. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[86] Wen Xiao and Giuseppe Carenini. Extractive Summarization of Long Documents by Combining Global and Local Context. *arXiv:1909.08089 [cs]*, September 2019. URL `http://arxiv.org/abs/1909.08089`. arXiv: 1909.08089.

[87] C. Xu, R. Wang, S. Lin, X. Luo, B. Zhao, L. Shao, and M. Hu. Lecture2Note: Automatic Generation of Lecture Notes from Slide-Based Educational Videos. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 898–903, July 2019. doi: 10.1109/ICME.2019.00159. ISSN: 1945-788X.

[88] Haojin Yang, Maria Siebert, Patrick Luhne, Harald Sack, and Christoph Meinel. Lecture Video Indexing and Analysis Using Video OCR Technology. In *2011 Seventh International Conference on Signal Image Technology & Internet-Based Systems*, pages 54–61, Dijon, France, November 2011. IEEE. ISBN 978-1-4673-0431-3 978-0-7695-4635-3. doi: 10.1109/SITIS.2011.20. URL `http://ieeexplore.ieee.org/document/6120629/`.

[89] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. XLNet: Generalized Autoregressive Pretraining for Language Understanding. *arXiv:1906.08237 [cs]*, January 2020. URL `http://arxiv.org/abs/1906.08237`. arXiv: 1906.08237.

[90] Zihao Ye, Qipeng Guo, Quan Gan, Xipeng Qiu, and Zheng Zhang. BP-Transformer: Modelling Long-Range Context via Binary Partitioning. *arXiv:1911.04070 [cs]*, November 2019. URL `http://arxiv.org/abs/1911.04070`. arXiv: 1911.04070.

[91] Manzil Zaheer, Guru Guruganesh, Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, and Amr Ahmed. Big Bird: Transformers for Longer Sequences. *arXiv:2007.14062 [cs, stat]*, July 2020. URL `http://arxiv.org/abs/2007.14062`. arXiv: 2007.14062.

[92] Jingqing Zhang, Yao Zhao, Mohammad Saleh, and Peter J. Liu. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. *arXiv:1912.08777 [cs]*, December 2019. URL `http://arxiv.org/abs/1912.08777`. arXiv: 1912.08777.

[93] Ming Zhong, Pengfei Liu, Yiran Chen, Danqing Wang, Xipeng Qiu, and Xuanjing Huang. Extractive Summarization as Text Matching. *arXiv:2004.08795 [cs]*, April 2020. URL `http://arxiv.org/abs/2004.08795`. arXiv: 2004.08795.

[94] Xinyu Zhou, Cong Yao, He Wen, Yuzhi Wang, Shuchang Zhou, Weiran He, and Jiajun Liang. EAST: An Efficient and Accurate Scene Text Detector. *arXiv:1704.03155 [cs]*, July 2017. URL `http://arxiv.org/abs/1704.03155`. arXiv: 1704.03155.

[95] David Zurow. daanzu/kaldi-active-grammar, September 2020. URL `https://github.com/daanzu/kaldi-active-grammar`. original-date: 2019-03-24T03:30:06Z.