Financial Audio Analysis:
Leveraging Speech for Sentiment Classification and Predictability.

John Arancio

Springboard DSC

Capstone Project 3

August 1, 2021

Table of Contents

- Abstract
- (1) Introduction
- (2) Approach
  - (2.1) Dataset Ensembling and Wrangling
  - (2.2) Exploratory Data Analysis [EDA]
  - (2.3) Data Augmentation
  - (2.4) Feature Engineering and Baseline Modeling
    - (2.4.1) Logistic Regression
    - (2.4.2) Support Vector Classifier [SVC]
    - (2.4.3) XGBoost
  - (2.5) Extended Modeling
    - (2.5.1) Convolutional Neural Network (CNN)
- (3) Findings
- (4) Conclusions
  - (4.1) Future Work
- (5) Client Recommendations
- (6) Consulted Resources

Abstract

Sentiment analysis and natural language processing is a growing practice within the financial industry. Using NLP, many analysts generate their reports from textual data (ex. News articles, Twitter tweets etc) and now with "transformers," these analyses can be powerful in understanding market volatility and momentum. However, one can argue that traditional sentiment analysis can sometimes lack in depth, as we can misconstrue written language, not being able to capture the many facets that contribute to an individual's genuine expression that which we mark as sentiment. It is our thought that by utilizing NLP with audio feature extraction, classifying received sentiment into categories should be more realistic for "ground truth". The acoustic attributes of one's speech contain useful information that is not only unique to the speaker but also comparable to others within the constraints of the frequency spectrum. By exploring audio within the specific industry of finance, it could be possible to separate the realistic sentiment from unrealistic sentiment, such as false claims targeting various assets, to sway the consensus.

We can separate our entire analysis into 4 main stages, each contributing to their ultimate conclusions provided in section (3) Findings. In order to conduct this experiment, we gained original data and ensemble our entire dataset from scratch. After constructing our dataset, we could then build 8 separate models. 4 of them being our baseline models (Logistic Regression, Support Vector Classifiers, and XGBoost) and 6 developed using deep learning (CNNx6). Out of the 4 baselines, we identified our Support Vector Classifier (SVC) as the best performing, with Logistic Regression being the worst performing. As for the convolutional neural networks, they all performed the same, as we faced issues that call for a reevaluation of approach. Each achieving an accuracy of around 60% and 0.85 loss.

Classification Reports (Baseline Models)

LOG. REGRESSION	0	1	2	W.AVG	SVC (RAPIDS)	0	1	2	W.AVG	SVC (SKLEARN)	0	1	2	W.AVG	XGBOOST	0	1	2	W.AVG
PRECISION	0.4	0.38	0.48	0.43	PRECISION	0.94	0.9	0.92	0.92	PRECISION	0.92	0.9	0.92	0.91	PRECISION	0.7	0.56	0.44	0.55
RECALL	0.27	0.45	0.51	0.43	RECALL	0.95	0.93	0.89	0.92	RECALL	0.95	0.93	0.88	0.91	RECALL	0.16	0.2	0.89	0.47
F1-SCORE	0.32	0.41	0.5	0.42	F1-SCORE	0.94	0.91	0.9	0.92	F1-SCORE	0.93	0.92	0.9	0.91	F1-SCORE	0.26	0.29	0.58	0.4
ACCURACY	0.43				ACCURACY	0.92				ACCURACY	0.91				ACCURACY	0.47			

Figure 1

Keywords: *audio-analysis, speech-sentiment, logistic-regression, convolutional-neural-network, xgboost, support-vector-classifier, nlp, transformer*

Loss Vs. Accuracy (CNN)

CNN VERSION	TEST LOSS	TEST ACCURACY
v1 (Shallow + tuned)	0.9341	0.5358
v2 (Larger filters + depth)	0.8658	0.5871
v3 (Depth + dropout)	0.8323	0.6192
v4 (Depth + dropout - FC dropout)	0.8134	0.6345

Figure 2

(1) Introduction

We gathered the research and discoveries realized from our analysis with prospects to eliminate noise within the financial sector, where market sentiment is a rising sign of price action. We are intending to produce more exact sentiment using audio, where NLP is employed as a supplementary addition to our audio through annotations. This combination of NLP and audio should be deemed preliminary because our ground truth (labels) are collected utilizing a neural network. However, as you'll discover from our investigation, this approach yields a classification framework that is both automated and accurate. Our results will develop future models and establish a fluid sentiment classifier end to end.

The details you'll find below in our approach are contained in four separate areas and five separate jupyter notebooks. It should be acknowledged that much of the dataset assembling process is not included within those five notebooks. This is because most of the processing required to be performed on our original audio files was conducted within many notebooks and operating the command line only. Each of the notebooks provides a brief introduction and inference, offering the reader proper context to the code. All notebooks can be located in this GitHub repository.

We compiled 10,548 unique audios to start our project and ended up with 176,433 useful sentences or phrases moving into the modeling portion of our inquiry. Those audio segments being where we obtained our features from. All audios used in this project contain matters of finance, ranging from GDP estimates, debt outlooks to equity, options, commodity trading. The specialized financial instrument was not relevant to us as long as the audio contained examples of potential outcomes or assessments. Each audio was less than 5 minutes and all communicated in the English language. We segmented our audios based on sentence uttered, and all segments were 16-bit, mono, and consisted of a 16000 Hz sample rate that of which were .wav files.

(2) Approach

Each step of the approach used to solve the classification problem at hand is divided in 4 sections below, with each section containing the detailed process involved. The process by which each section is performed is only the first rendition of many more due to having limited time and limited computational resources. The aim overall was to acquire empirical evidence of potentiality in regards to a possible use case and lay the groundwork moving forward long-term.

(2.1) Dataset Ensembling and Wrangling

In order to assemble our dataset we downloaded 11,525 videos from Youtube via "web scraping." To properly scrape and acquire the videos in an efficient manner we wrote a basic python scraping application based on the selenium library. The script ultimately searched for Youtube videos with a

provided search term and filters. Once at the dynamic search page it then automated its scrolling while collecting each video on the page along with its title and link. From there all the data scraped from YouTube was then written to a .csv file which was then fed to another library or tool called “youtube-dl” and each video collected was then downloaded and converted to audio as a .mp3. We then used the “sox” library to batch convert our audios to .wav and the same sample rate (16000 Hz). Once we completed our audio collection, we then moved to the transcription process. Figure 2 below visualizes the entire assembling process.

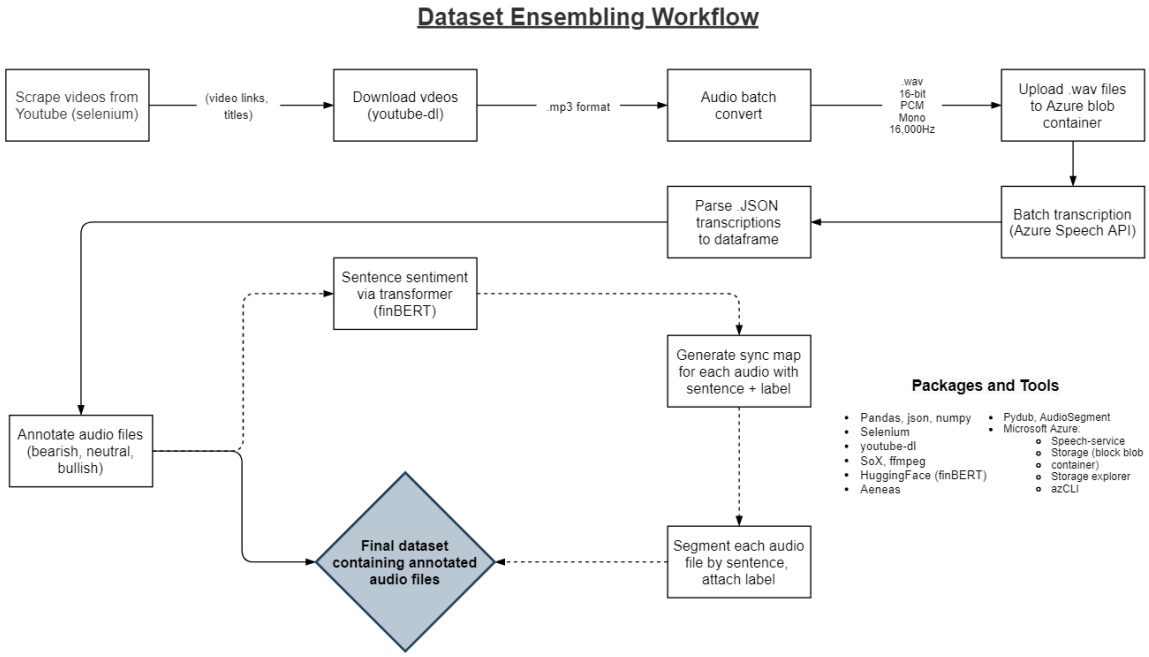


Figure 3

First, we needed to transcribe the audios to properly annotate our eventual segments that were contained in a single audio. We used Microsoft Azure “Speech” API for speech to batch transcribe each of the 11,525 audios.[1] All transcriptions were provided in JSON format, which was then parsed through and cleaned for a proper NLP analysis.

Second, in order to speed up and automate the annotation process, we incorporated the use of a transformer provided by “HuggingFace” and Google. The transformer model we used was called “FinBERT” which is a modified version of Google's “BERT” model built specifically for the financial domain.[2] FinBERT was trained on the dataset “FinancialPhraseBank” which contains 4500 sentences labeled by 16 experts in the financial industry and was able to achieve 97% accuracy. Being that it was fine-tuned using finance specific terms and the versatile nature of transformers while maintaining a very high accuracy it seemed like a valid fit in regards to being able to automate the annotations of our audios. Using FinBERT we tokenized and segmented our text transcriptions, allowing the model to provide us with a sentiment based annotation per sentence.

[1] wolfma61 (n.d.). How to use batch transcription - Speech service - Azure Cognitive Services. [online] docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/batch-transcription [Accessed 10 Aug. 2021]
[2] Araci, D. (2019). FinBERT: Financial Sentiment Analysis with Pre-trained Language Models. arXiv:1908.10063 [cs]. [online] Available at: https://arxiv.org/abs/1908.10063

Next, we gathered our annotated sentences to be fed into the package, “aeneas", which allowed us to synchronize our sentence fragments with our audio files.[3] Since our audio files were a different length (longer) than the labeled transcription sentences, we had aeneas take our file containing all the sentences in a single audio file as the input paired with the actual audio file. The output would then be a new .wav file labeled with the tagged sentence generated by FinBERT as its filename. This process was performed over the course of a few days, periodically checking to see if the file's outputs matched the inputs manually. All new audio files successfully matched correctly along with its respective annotation, resulting in our 176,433 sentence audios.

Then finally, once we compiled our final dataset containing all audio segments, we then trimmed each segment based on the average length of segments, which was roughly 6.0 seconds with a standard deviation of about 2.0 seconds. We also converted all audio files to mono and a sample rate of 16000 Hz. The objective of this step was to make sure all the files contained the same properties as each other, making sure our signals are consistent through and through. From this point, we were then able to perform our Exploratory Data Analysis (EDA) and visualize the assembled dataset in detail.

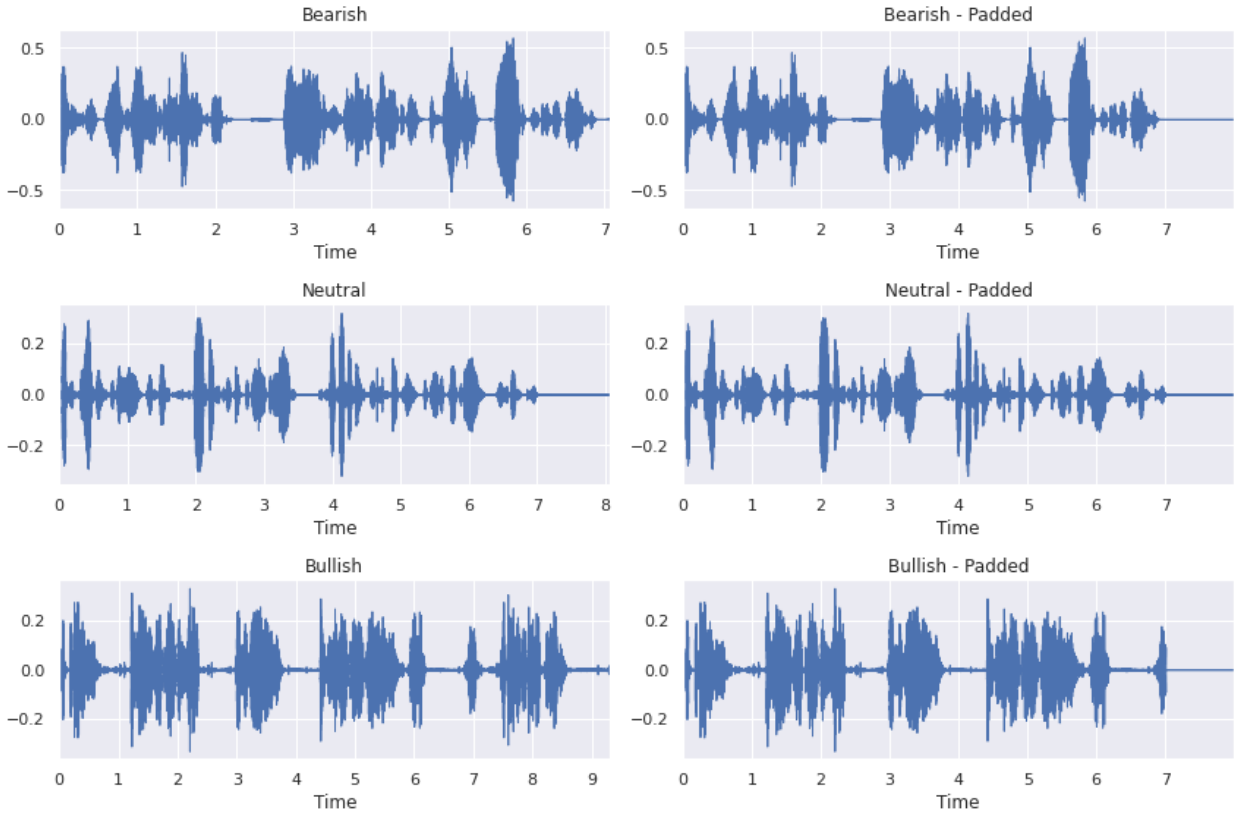
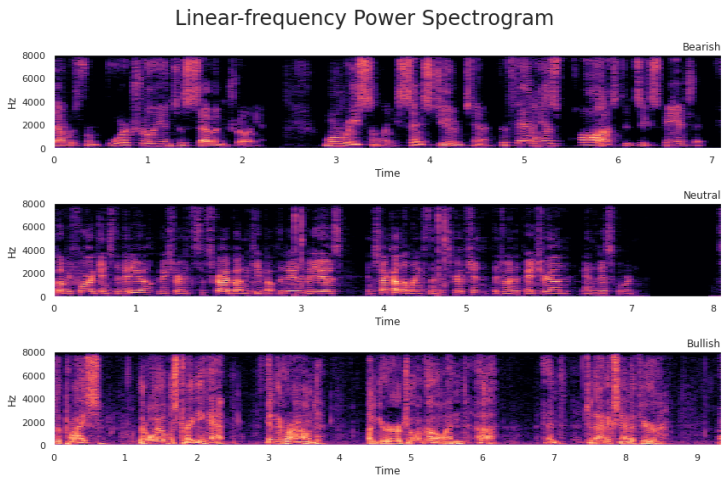


Figure 4

[3] www.readbeyond.it. (n.d.). aeneas: automagically synchronize audio and text. [online] Available at: https://www.readbeyond.it/aeneas/ [Accessed 10 Aug. 2021].

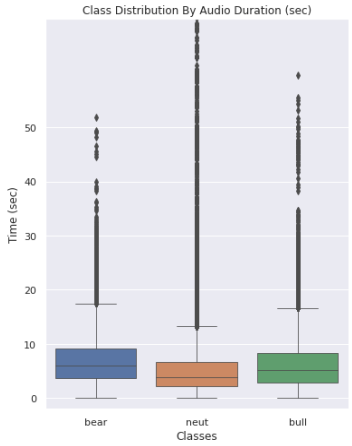
(2.2) Exploratory Data Analysis (EDA)

During this stage, we visualized each of our classes audio signal waveform, spectrogram and mel-frequency cepstral coefficients (MFCCs) in order to get an idea of what our audios look like across its different feature representations. Along with those visuals we also plot the distribution of our classes (bearish, bullish, neutral) while computing the percentage difference between each. It was also useful to use box plots to view the entire dataset distribution and view class outliers in respect to certain attributes (i.e. duration, sample rate, channels).



As for the feature plots, there was no obvious visual indication of difference between audio classes within the frequency spectrum. Although, while examining the plots visually we did notice that the bullish audios seemingly contained more frequent and longer drawn out pauses of silence, more so than the other classes. This was hypothesized as a potentially useful inference that might aid in feature extraction or when building models. (See Figure above)

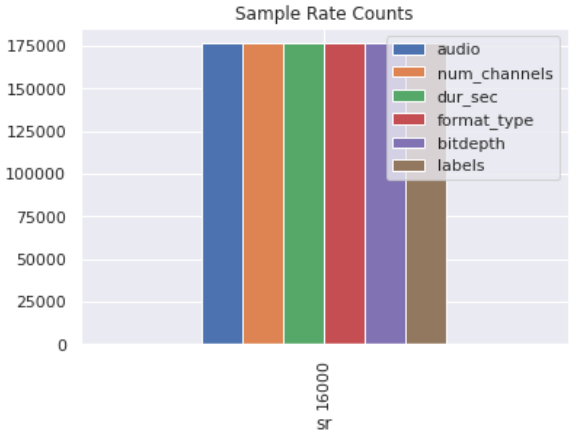
We explored further by viewing the distributions of the dataset as a whole. From that we noticed that our dataset was incredibly unbalanced. The amount of neutral audios was 74.204% greater than the combined amount of bearish and bullish audios. Our bearish audios made up 11.627% of our dataset, and bullish audios made up 14.169% of our dataset, giving us a combined difference of both bullish and bearish audios to neutral audios of about 48.408%. We used a bar plot like the figure to the right to visualize the unbalance, where each bar represents a single class.



Shown in the figure to the left, we then used box plots to visualize the outliers contained within each class individually by counting the amount of audio lengths. By visualizing the length variance we were able to understand why reformatting our audios was a necessary procedure. As mentioned previously we remedied the gap in audio lengths by cutting all audios to a maximum of 8 seconds, if the segment was shorter than that length we simply zero padded the signal. It is also interesting to mention that from viewing the distribution of the audio durations specifically using a histogram we can see it forms a

left-skewed distribution where the dataset contained more shorter length audios than longer ultimately revealing the outliers visually. The same thing can be seen using a boxplot as well.

Our EDA was extremely useful in figuring out what steps needed to be taken in order to move forward. Since we found a major unbalance within our data, it meant that it was then necessary to oversample the two classes that were lacking in depth (bearish and bullish). To perform the oversampling we used a few different augmentation techniques, explained in section below. Moreover as a result of this strong unbalance in our dataset we decided that the best approach to building, training and testing our models from the initial split was to create a copy of our original dataset and then oversample the copy, leaving the original untouched for evaluation.



The initial train/test split was performed using random stratification with 80% as our train-set and 20% as our test-set. We then split the train-set once more while maintaining the original distribution using the same 80/20 ratio, however, for this split we used a stratified K-Fold method. The new train-set and test-set is used to train and cross validate on, and then the evaluation was then performed on the un-augmented test-set containing 20% of original dataset. This approach allowed us to perform an unbiased evaluation and gather a model's performance on truly unseen data that does not contain “synthetic” or augmented audio signals as input. This is visualized below.

Augmenting to Train/Test Models Workflow

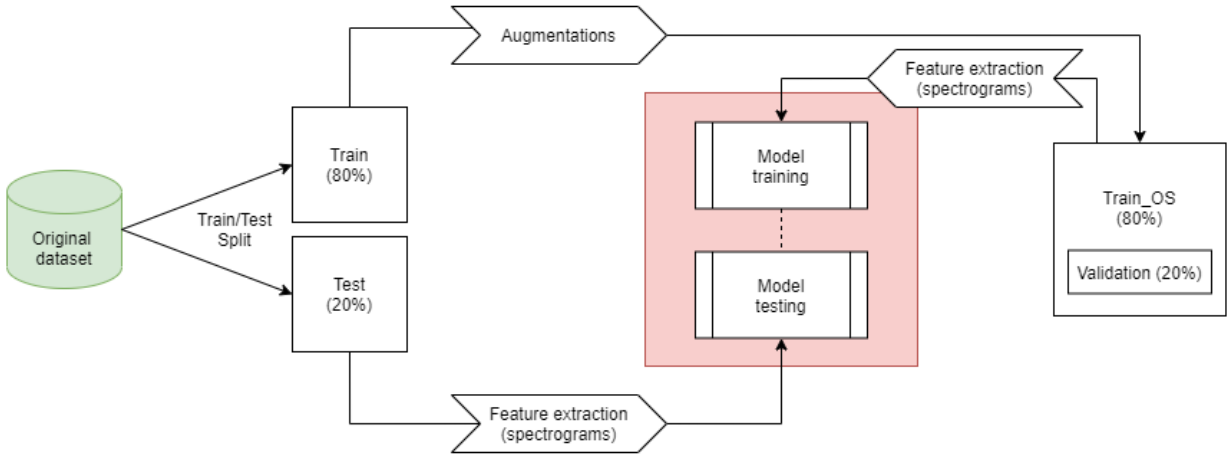
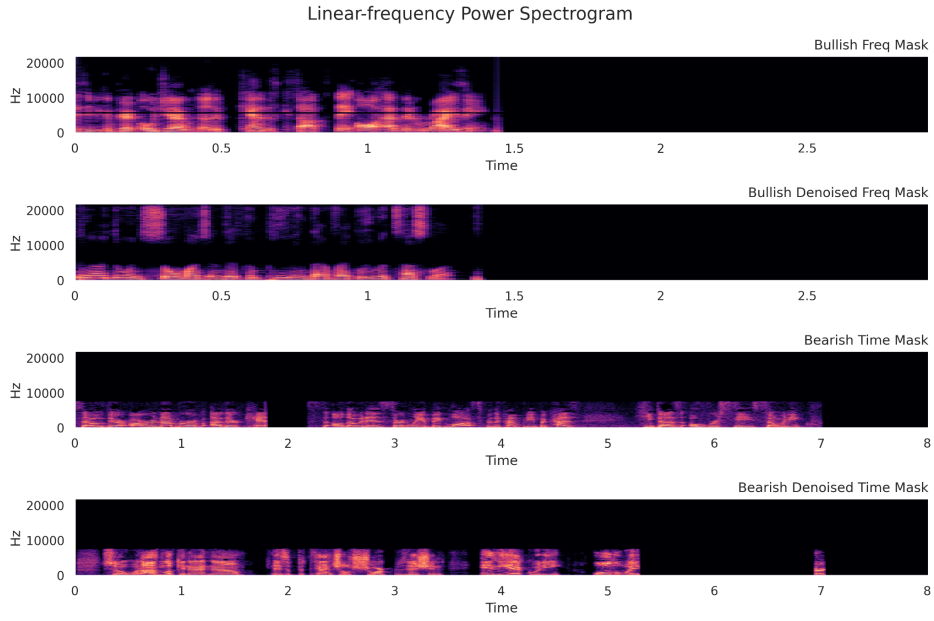


Figure 10

(2.3) Data Augmentation

We used 4 separate augmentation techniques (visualization in figure below):

- Sample denoising
- Sample shifting
- Sample frequency mask
- Sample time mask



Sample denoising is usually used to clean an audio's background noise in order for the vocals/speech to become more audible. From listening to many of the audios manually, background noise was somewhat an issue in respect to clarity. Since the denoising was assumed to produce a marginally different alteration from the original audios, we felt it would contribute positively to the oversample. To denoise the audio we used a deep learning neural network called, "DTLN" which is a Dual-Signal Transformation LSTM Network for Real-Time Noise Suppression.[4] The model instance used was trained on 500 hours of speech and noise data and took roughly 1 hour to complete the process of generating new "denoised" audios. We augmented 100% of both bearish and bullish audios, giving us double the amount of audios for each class, totaling 41028 bearish audios and 49998 bullish audios. We also denoised a random sample of neutral audios, approx. 17801, making up for the difference of denoised vs original audios across all 3 classes. We determined that augmenting the neutral class as well would benefit our models eventual training.

After completing the denoising, we then used sample shifting, sample frequency mask, and sample time mask on both our clean and unclean audios. For each of the 3 methods we augmented 50% of denoised and 50% of original. This gives us a 100% augmented audio for both bullish and bearish classes, further minimizing the unbalance across the entire dataset. After performing all augmentations we eventually arrived at a less unbalanced dataset than previously where it contained 65471 bearish audios, 80257 bullish audios and 122515 neutral audios. See Figure 11 below.

[4] Westhausen, N.L. and Meyer, B.T. (2020). Dual-Signal Transformation LSTM Network for Real-Time Noise Suppression. Interspeech 2020. [online] Available at: https://www.isca-speech.org/archive/Interspeech_2020/pdfs/2631.pdf.

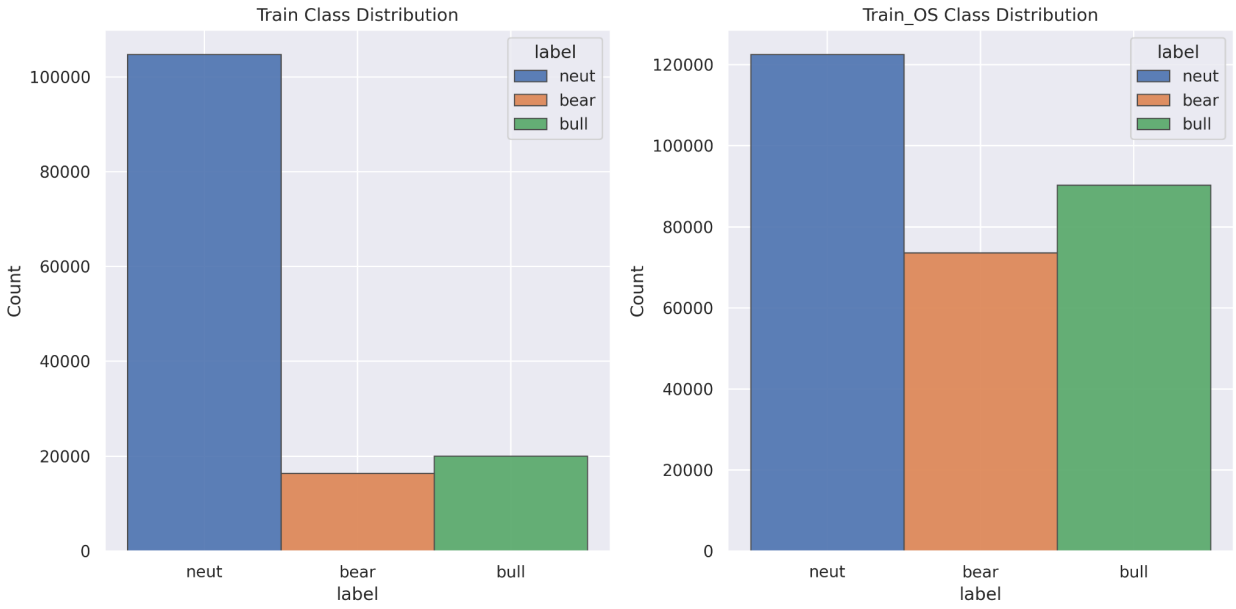


Figure 11

(2.4) Feature Engineering and Baseline Modeling

For feature extraction we chose to use the mel-spectrograms. The spectrograms on the mel-scale are considered a non-linear transformation of an audio signal ultimately as result of fourier transformations. A Fourier Transform decomposes the signal into its constituent frequencies and displays the amplitude of each frequency present in the signal. Because humans perceive sound in only a narrow range of frequencies, the logarithmic mel-scale is more useful because it allows us to extract frequencies that are not seen within a normal spectrogram. Allowing us to use data that humans would not be able to hear themselves. We first get our spectrograms on the decibel scale and then convert to amplitude which is how we perceive loudness, revealing the entire spectrogram frequencies for each speech audio signal. Time is on the x-axis and amplitude is on the y-axis of our spectrogram plots.

That being said, we then extracted each mel-spectrogram from each of our audios and then proceeded to scale and transform our features data. In order to do that, we first use MinMaxScaler with a feature range of 0,1. MinMaxScaler from the Scikit-learn package was used instead of the common “StandardScaler” because it preserves the shape of the data (the audio spectrograms) resulting in zero distortion.

After scaling the spectrograms we then reduce the datasets dimensionality using a Principal Component Analysis (PCA), retrieving the components that produced 95% of the variance contained within the entire dataset. This greatly reduced the amount of columns, thus reducing the data size while maintaining the features that explain the variance or details of each spectrogram. Now that we have our most important features scaled and transformed, we can then construct our baseline models.

We constructed 4 unique models. Two of which use the same algorithm but contain a different “approach.” We used one Logistic Regression algorithm, one XGBoost and then one we used twice was a Support Vector Classifier (SVC). Each of our models produced different results and were built using a

K-Fold cross validation via GridSearch. Each GridSearch used 10 folds in cross validating while searching in order to tune for maximum performance. GridSearch cross validation was used to find the best parameters for each respective model. The difference between the two SVC models is that one used the Scikit-learns package and the other utilized the RAPIDS[5] library from Nvidia in order to accelerate the computations. With the aid of acceleration we were actually able to achieve slightly better scores when evaluating performances. Our XGBoost model was also accelerated using a GPU. Aside from the two SVC models, our test performance results differed greatly. The Logistic Regression model performed the worst out of the four, and the accelerated version the SVC performed the best.

Training times:

- Logistic Regression: 6 hours
- SVC w/o GPU: 10 hours
- SVC w/ GPU: 3 hours
- XGBoost: 45 min

(2.4.1) Logistic Regression

```
{C: 0.09, solver: lbfgs, max_iter: 5000, penalty: l2, tol: 0.0001}
```

The logistic regression model was designed as our original model and consisted of the parameters above.

	precision	recall	f1-score	support
0	0.40	0.27	0.32	8184
1	0.38	0.45	0.41	10032
2	0.48	0.51	0.50	12251
accuracy			0.43	30467
macro avg	0.42	0.41	0.41	30467
weighted avg	0.43	0.43	0.42	30467

These parameters were found to be the best fit using a GridSearch, however, three of the parameters were not tuned. Those being solver (lbfgs), max iterations (5000) and penalty (l2). The solver used was necessary to correctly fit and predict our data, and is in the family of quasi-Newton methods that approximates the Broyden–Fletcher–Goldfarb–Shanno algorithm (BFGS) using a limited amount of computer memory. Based on GridSearches results, algorithms ‘sag’ and ‘saga’ provided no performance improvement even though they are mainly used with large datasets like ours for multi classification.

We also ran into a problem with the “max_iter” parameter, where line search failed during the earlier number of iterations so we had to keep increasing them to actually produce tangible results. Also, in regards to the penalty parameter, a penalty of l2 was necessary to add regularization to our model in order to prevent overfitting.

[5] RAPIDS. (n.d.). Open GPU Data Science. [online] Available at: https://rapids.ai/about.html [Accessed 10 Aug. 2021].

(2.4.2) Support Vector Classifiers (SVC)

```
{'C': 100, 'gamma': 0.001, 'kernel': 'rbf', 'multiclass_strategy': 'ovo'}
```

Both SVC renditions, accelerated and not, utilized GridSearch which selected the same parameters for both models. Our multiclass strategy showed the greatest improvement in performance in comparison to the other parameters. Trials were run using "OVR" or one-verse-rest and results were extremely poor in comparison to "OVO." The kernel selection was interesting, as RBF provided us with the best results, showing us that our data works well with a space of Gaussian distributions. Where each point becomes a probability density function of a normal distribution within the means of scaling. Since dot products are integrals. It works well for our case (many features, unbalanced dataset/classes) by providing us with flexibility to separate everything gathered by the support vector machine.

That being said, the accelerated version performed slightly better overall looking at the classification reports weighted averages for both.

RAPIDS - cuML

	precision	recall	f1-score	support
0	0.94	0.95	0.94	8184
1	0.90	0.93	0.91	10032
2	0.92	0.89	0.90	12251
accuracy			0.92	30467
macro avg	0.92	0.92	0.92	30467
weighted avg	0.92	0.92	0.92	30467

Scikit-learn

	precision	recall	f1-score	support
0	0.92	0.95	0.93	8184
1	0.90	0.93	0.92	10032
2	0.92	0.88	0.90	12251
accuracy			0.91	30467
macro avg	0.91	0.92	0.92	30467
weighted avg	0.91	0.91	0.91	30467

(2.4.3) XGBoost

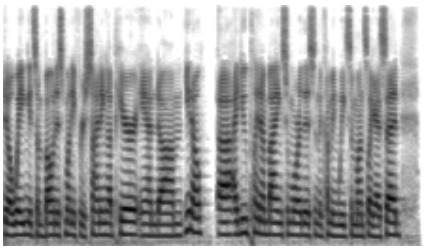
```
{'subsample': '0.',  'max_depth': '12', 'n_estimators': '100',  'learning_rate':
'0.006', 'num_classes': '3', 'colsample_bytree': '0.3',  'gamma': '5', 'objective':
'multi:softprob', 'num_classes': '3', 'tree_method': 'gpu_hist', 'sampling_method':
'gradient_based', 'eval_metric': 'mlogloss'}
```

	precision	recall	f1-score	support
0	0.70	0.16	0.26	8184
1	0.56	0.20	0.29	10032
2	0.44	0.89	0.58	12251
accuracy			0.47	30467
macro avg	0.56	0.42	0.38	30467
weighted avg	0.55	0.47	0.40	30467

The XGBoost model surprisingly performed almost as badly as logistic regression, although its individual class metrics scored considerably high in comparison. The weighted averages, scored similarly. We also used GridSearch on the classifier, and after a large amount of trial and error we were able to increase accuracy performance during training to around 70% but when evaluated it performed no better than flipping a coin. This showed us that the model itself has a hard time generalizing to unseen data and could possibly be overfit given a "high" training score.

After gaining insight from our baseline models, we moved on to our extended modeling phase containing deep learning using convolutional neural networks (CNN).

(2.5) Extended Modeling



Before building our neural net architectures, we had to first export our mel-spectrograms in .png form. We used an inverted RGB, whereas the blacks now became white. From our own due diligence it was found that this type of coloring usually presents more useful information for the network to learn from. To accelerate the spectrogram calculations we used another neural network (DALI[6] from Nvidia) to extract features and then save each image. By accelerating this process, we were able to process our audio signals at a fraction of the time as compared to librosa while getting nearly the exact same output. This method of feature extraction is also useful in regards to "data loading" and forming an end to end model pipeline.

It is also worth mentioning that for this portion of modeling, we split our oversampled dataset with a different technique. Instead of splitting labeled audios directly, we split and copied the folders that contained audios for each class, randomly, to produce two new folders labeled "train" and "validation". Each folder contains subfolders labeled by classes bearish, bullish and neutral.

(2.5.1) Convolutional Neural Network (CNN)

We experimented with 4 separate CNN architectures. Each model contains a different framework regarding layering or preprocessing. The constant throughout all 4 was the scaling aspect of our data generator. We were required to scale our pixel data before feeding input into our CNNs.

[6] GitHub. (n.d.). GitHub - NVIDIA/DALI: A GPU-accelerated library containing highly optimized building blocks and an execution engine for data processing to accelerate deep learning training and inference applications. [online] Available at: https://github.com/NVIDIA/DALI [Accessed 10 Aug. 2021].

Each model performed almost the same when examining its accuracy and loss plots per epoch. We noticed a consistent sign of overfitting after an average of 10-15 epochs. We attempted more experiments with each model trying different regularization techniques such as l1 or l2 penalties and adding "Dropout" layers, however performance still remained stagnant across the board. Since model training consistently seemed to plateau after x amount of time, it might be possible that our data could be flawed either in regard to the augmentations used, the amount of data, or even a result of the unbalanced nature of our dataset.

Two of the models not found in our notebooks that did not demonstrate any positive results on the dataset were built using "transfer learning", through the use of the "state-of-the-art" performance level models of ResNet50[7] and VGG16[8]. After numerous attempts, the two transferable models were not able to surpass 60% accuracy even with countless hours of training, layer editing and parameter tuning. Moreover, 2 custom neural networks were able to produce results above 60% accuracy. While they also seemed to begin overfitting around the same amount of epochs during training. See Figure 17 below to view loss and accuracy plots of our 3 best performing models:

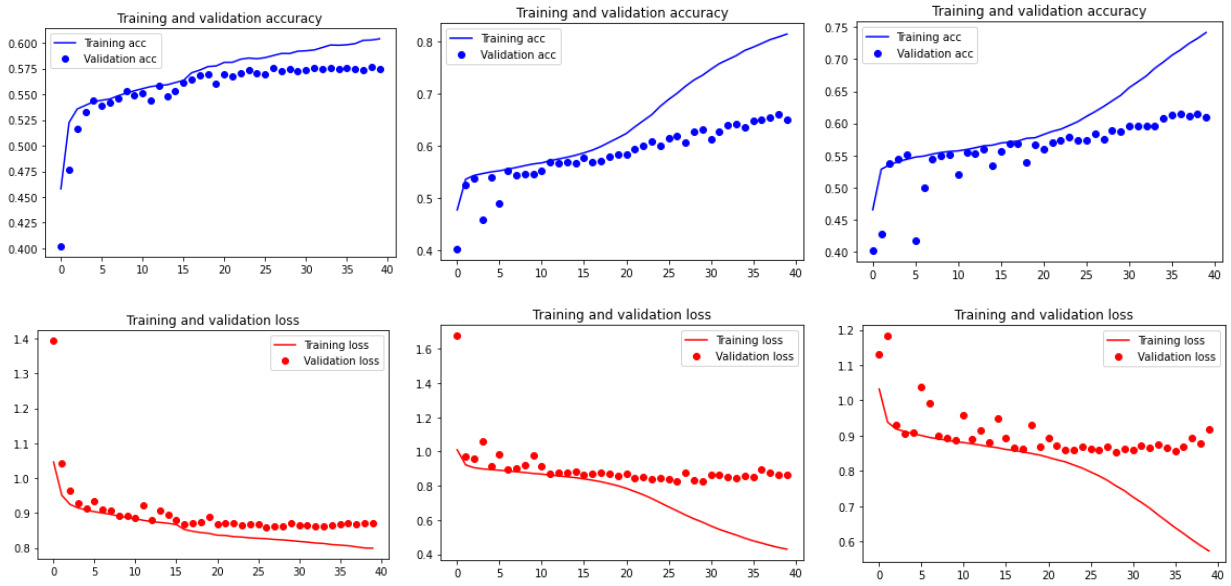


Figure 17

(3) Findings

Our SVC models clearly performed the best, with the accelerated version achieving a weighted avg. precision, recall and f1-score of 0.94, 0.95, 0.94 respectively. The non-accelerated sklearn version scored the same in recall, however, precision and recall were slightly lower than the accelerated, which was expected. Sticking with the SVC model produced using RAPIDS, it received its highest score in recall with the bearish class, correctly identifying positive cases of bearish 95% of the time. Precision (0.94) and

[7] He, K., Zhang, X., Ren, S. and Sun, J. (2016). Identity Mappings in Deep Residual Networks. arXiv:1603.05027 [cs]. [online] Available at: https://arxiv.org/abs/1603.05027.

[8] Simonyan, K. and Zisserman, A. (2015). Published as a conference paper at ICLR 2015 VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. [online] . Available at: https://arxiv.org/pdf/1409.1556.pdf.

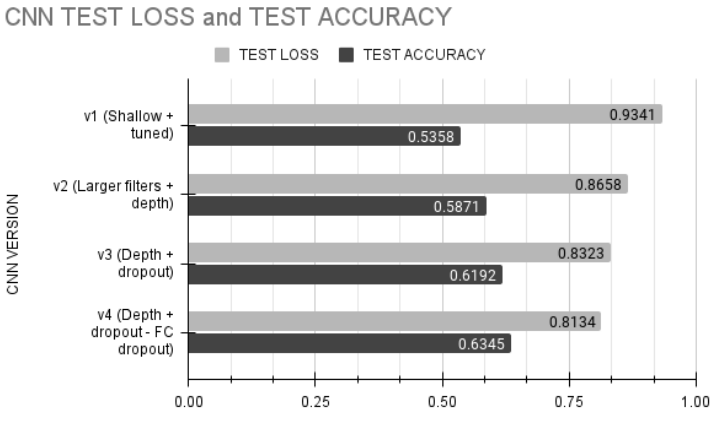
F1 (0.94) were also the highest for the bearish class, so we were able to correctly identify bearish audios 94% of the time based off of precision and looking at F1 score, we can see that our predictions for these audios were accurate also 94% of the time. However, looking at the weighted average of these metrics, they are a bit lower, most notably F1 at 0.92. Still, this was a tremendous jump from other models we produced. Both logistic regression and XGBoost scored rather poorly in comparison.

Based on its accuracy score alone, the logistic regression build was the worst performing overall, reaching only 0.43. It was not able to break the 50% mark in most categories, containing a weighted avg. for F1 of 0.42, precision of 0.43 and recall of 0.43. However, digging a little deeper, we can see that its weighted avg. F1 scored slightly higher than our XGBoost. XGBoost on the other hand, scored at 0.47 for accuracy, Even though its performance seems better than logistic regression, that might not be the case when looking closer at the individual metrics. We can see that the weighted avg. for F1 with XGB was only rated at 0.40. Meaning that on average, logistic regression will make more correct predictions, although not by much. Another characteristic to take into consideration are the weighted avgs. Of both precision and recall since we received decently higher scores in those categories with a 0.55 and 0.47 respectively as compared to 0.43 and 0.43 with logistic regression. Moreover, by taking a closer look at the models performances using the individual classes statistics a large spread between the metrics recorded is visible. It ranges from a 0.16 in recall with the bearish class, and increases up to 0.89 with the neutral class. Then looking at class precision, we have a 0.7 for the bearish class and a 0.44 for the neutral class, showing a considerably inverted correlation to the recall scores. This relationship demonstrates how our dataset might be a strong contributing factor to the poor performance from XGB overall. We can see that it is strong at identifying the label when the class has a large amount of data to review, and it also revealed that the bearish class must have strong attributes that are shared between each audio signal to be able to correctly predict a bearish audio segment which contains significantly less data points than the other two classes.

That being said, the baseline line models provided valuable insight into not only some possible flaws contributing to an inconsistent performance, but also what to expect moving forward when applying the analysis to our deep learning efforts.

CNN

As you can see in the figure to the right, out of our four models the last version (v4) produced the best results. It achieved a test loss of approx. 0.8134 and an accuracy of 0.6345. This was nearly a 100% increase in score from our original v1 model. The difference between the two was a drastic increase of architecture depth, where we increased the amount of Conv2D layers as well as their filter sizes. V4 also contained



multiple dropout layers, each dropout placed directly after a MaxPooling block. However, going from v3 to v4 we removed the dropout layer at the end of the fully connected layer.

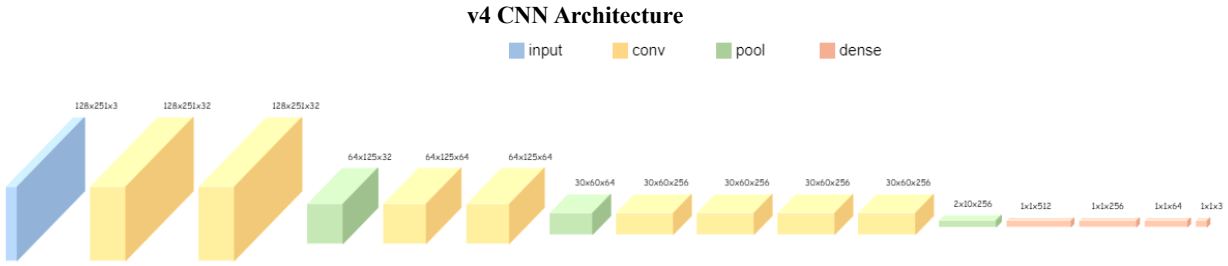


Figure 19

In noticing the changes that lead up to our final model, it is clear that the constant leading to subtle performance increases was the depth and width of the neural net. In the figure above, this can be visualized for our best model (v4). Through our research we found that increasing network depth is a trivial way to reduce overfitting when using a fairly large dataset for training. However, even though we were able to gain a slight increase in performance through added size it still did not meet expectations. After v4 we actually produced a few more models that were even larger but we could not beat its performance, and training seemed to get erratic after about 10 epochs.

We believe that the stagnation of model performances even after trying different parameters is a result of the way we compiled our dataset, whether it might be through augmentations or the distribution of classes as we mentioned previously.. However, comparing the CNNs to our baseline models it is interesting how our SVC was able to produce results drastically better than v4. Further investigation of interpretability might help us solve why that is.

(4) Conclusions

Tying the entire analysis together with our original goal of classifying sentiment generated from financial audio, we can conclude that it is in fact possible and our approach can be a viable solution to the problem; predicting and identifying if a speech audio segment is bullish, bearish or neutral. Although our results that were produced seemed relatively inconsistent, when we dig deeper into the classification reports in relation to our ensemble dataset we can see that the models are learning and picking up unique attributes to each class from our extracted features (mel-spectrograms). We can especially see this happening with our SVC model and XGBoost.

Being that we were unsure of the original dataset and how it would perform since we used an experimental approach in regards to the annotation process, the achieved outcomes of our baseline models were much better than expected. Now moving forward we are able to “fine-tune” our approach and correct the missteps along the way which could have led to our performance inconsistency. After doing so, we will more than be able to consistently predict and classify any unseen audio into its respective class

and improve the way market sentiment is generated. Thus, leading to better investment decisions and truly fair buy/sell opportunities, limiting risk overall and allowing the user to be more confident in its ideas.

(4.1) Future Work

- Re-evaluate augmentation approach
    - Try new data augmentations or limit the amount used
    - Augment features instead of original signals
- Collect more data
    - Ensemble more balanced set
- Incorporate LSTM into CNN architecture for more temporal features
- Try different feature extraction methods -evaluate results
    - Mel-frequency cepstral coefficients (MFCCs)
    - Chromagram features
        - Chroma Energy Normalized (CENS)
        - Constant-Q
    - RMS
    - Spectral
        - Centroid
        - Bandwidth
- Add more depth to CNN model
    - Add new preprocessing layers
    - Different Conv2D and MaxPooling approach
    - Increase width

Long Term

- Deploy functional model as application
- Test in “real world” conditions

(5) Client Recommendations

1. Use classification results metrics to improve model accuracy
2. Use the findings from our EDA to better understand the data collected
3. Try different algorithms comparable to SVM, like “DecisionTree”

(6) Consulted Resources

1. Simonyan, K. and Zisserman, A. (2015). Published as a conference paper at ICLR 2015 VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION. [online] . Available at: https://arxiv.org/pdf/1409.1556.pdf.
2. Westhausen, N.L. and Meyer, B.T. (2020). Dual-Signal Transformation LSTM Network for Real-Time Noise Suppression. Interspeech 2020. [online] Available at: https://www.isca-speech.org/archive/Interspeech_2020/pdfs/2631.pdf [Accessed 10 Aug. 2021].
3. GitHub. (n.d.). GitHub - iver56/audiomentations: A Python library for audio data augmentation. Inspired by albumentations. Useful for machine learning. [online] Available at: https://github.com/iver56/audiomentations [Accessed 10 Aug. 2021].
4. Park, D., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. and Le, Q. (n.d.). SpecAugment: A Simple Data Augmentation Method for Automatic Speech Recognition. [online] . Available at: https://arxiv.org/pdf/1904.08779.pdf.
5. Brian McFee, Alexandros Metsai, Matt McVicar, Stefan Balke, Carl Thomé, Colin Raffel, Frank Zalkow, Ayoub Malek, Dana, Kyungyun Lee, Oriol Nieto, Dan Ellis, Jack Mason, Eric Battenberg, Scott Seyfarth, Ryuichi Yamamoto, viktorandreevichmorozov, Keunwoo Choi, Josh Moore, Rachel Bittner, Shunsuke Hidaka, Ziyao Wei, nullmightybofo, Darío Hereñú, Fabian-Robert Stöter, Pius Friesch, Adam Weiss, Matt Vollrath, Taewoon Kim, Thassilo, 2021. librosa/librosa: 0.8.1rc2. https://doi.org/10.5281/zenodo.4792298
6. GitHub. (n.d.). GitHub - NVIDIA/DALI: A GPU-accelerated library containing highly optimized building blocks and an execution engine for data processing to accelerate deep learning training and inference applications. [online] Available at: https://github.com/NVIDIA/DALI [Accessed 10 Aug. 2021].
7. Raschka, S., Patterson, J. and Nolet, C. (2020). Machine Learning in Python: Main developments and technology trends in data science, machine learning, and artificial intelligence. arXiv:2002.04803 [cs, stat]. [online] Available at: https://arxiv.org/abs/2002.04803 [Accessed 10 Aug. 2021].
8. www.readbeyond.it. (n.d.). aeneas: automagically synchronize audio and text. [online] Available at: https://www.readbeyond.it/aeneas/ [Accessed 10 Aug. 2021].
9. Team, K. (n.d.). Keras documentation: About Keras. [online] keras.io. Available at: https://keras.io/about/.
10. TensorFlow. (2019). TensorFlow. [online] Available at: https://www.tensorflow.org.
11. Thoma, M. (2017). Analysis and Optimization of Convolutional Neural Network Architectures. [online] . Available at: https://arxiv.org/pdf/1707.09725.pdf#page=134 [Accessed 10 Aug. 2021].
12. Ioffe, S. and Szegedy, C. (2015). Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. [online] arXiv.org. Available at: https://arxiv.org/abs/1502.03167.
13. Araci, D. (2019). FinBERT: Financial Sentiment Analysis with Pre-trained Language Models. arXiv:1908.10063 [cs]. [online] Available at: https://arxiv.org/abs/1908.10063.
14. huggingface.co. (n.d.). Hugging Face – The AI community building the future. [online] Available at: https://huggingface.co [Accessed 10 Aug. 2021].
15. KDnuggets. (n.d.). Audio Data Analysis Using Deep Learning with Python (Part 1). [online] Available at: https://www.kdnuggets.com/2020/02/audio-data-analysis-deep-learning-python-part-1.html [Accessed 10 Aug. 2021].
16. Yoon, S., Byun, S. and Jung, K. (n.d.). MULTIMODAL SPEECH EMOTION RECOGNITION USING AUDIO AND TEXT. [online] . Available at: https://arxiv.org/pdf/1810.04635.pdf [Accessed 10 Aug. 2021].
17. wolfma61 (n.d.). How to use batch transcription - Speech service - Azure Cognitive Services. [online] docs.microsoft.com. Available at: https://docs.microsoft.com/en-us/azure/cognitive-services/speech-service/batch-transcription [Accessed 10 Aug. 2021]
18. Ananthapadmanabha, T.V., Prathosh, A.P. and Ramakrishnan, A.G. (2014). Detection of the closure-burst transitions of stops and affricates in continuous speech using the plosion index. The Journal of the Acoustical Society of America, 135(1), pp.460–471.
19. Wyse, L. (n.d.). Audio spectrogram representations for processing with Convolutional Neural Networks. [online] . Available at: http://dorienherremans.com/dlm2017/papers/wyse2017spect.pdf [Accessed 10 Aug. 2021].