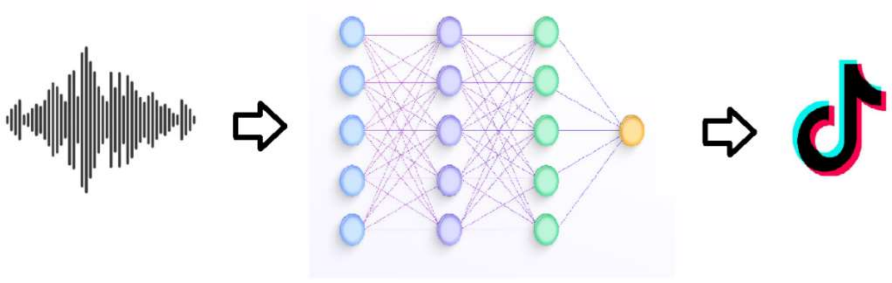
Machine Learning for TikTok: Predicting Song Virality from Audio Content



Omer Sideman, 315232785

omersideman@campus.technion.ac.il

Tomer Stopler, 325045961

tomerstopler@campus.technion.ac.il

Introduction:

In recent years, there has been an escalating surge in the popularity of the Chinese social media platform, TikTok. This surge is notably pronounced among the youth, but it's also making considerable headway among adults. To put things in perspective, TikTok presently boasts a user base of nearly 1.67 billion registered accounts, with approximately 1.1 billion of them actively engaged. This grants TikTok an extraordinary reach for content exposure.

Within this social ecosystem, users have the capacity to share videos of varying durations, spanning a diverse spectrum of content — from dance routines and musical performances to informative, educational, and current events-driven clips.

A predominant feature in many of these videos is the inclusion of songs and melodies. The creators behind these musical compositions receive recognition through royalties determined by the frequency of their songs' use.

This prompts an intriguing question: is there a specific musical genre that tends to garner significant traction on TikTok? In simpler terms, are there particular songs that find their way into a multitude of TikTok videos?

Project Idea:

This capacity for discerning intricate patterns and relationships within complex data is precisely why we turned to machine learning for this project. It has demonstrated remarkable success in a wide range of audio analysis tasks, including speech recognition, music genre classification, emotion recognition for sentiment analysis, and even specialized fields like bird song classification. Machine learning's ability to uncover these hidden correlations enables us to extract valuable insights from audio data, making it an ideal tool for predicting the viral potential of songs based solely on their audio signatures.

<u>Data Collection</u>:

Data collection presented a unique challenge in this project. While Spotify offers a public API for access, TikTok, unfortunately, does not and as a result, finding popular songs in TikTok wasn't a simple task. Initially, we attempted to leverage Spotify playlists like "viral TikTok songs 2023" as a workaround. However, it became evident that this method would not provide the level of accuracy we sought. Additionally, utilizing Spotify's popularity metric proved insufficient, as it did not align directly with TikTok's popularity metrics, as you can see in figure 1, where we compare the Spotify popularity score to the number of TikTok videos with that song and the number of likes for that song on TikTok.

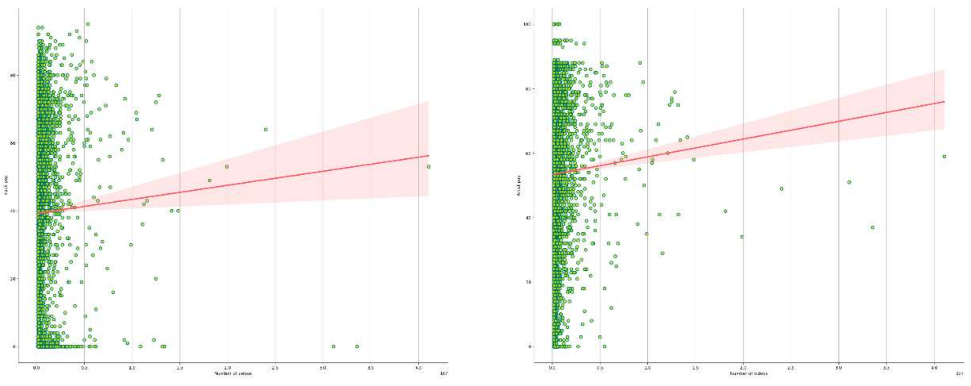


Figure 1- Spotify's popularity metric is on the y-axis. TikTok 'videos per song' feature on the left, and TikTok 'likes per song' feature on the right on the x-axis.

This discrepancy was exemplified by instances where a song gained virality on TikTok due to its humorous lyrics but did not translate into high Spotify streaming numbers. Ultimately, we located a reliable and current list of TikTok trending tracks[1] procuring it through web scraping to ensure we were working with precise and up-to-date data. After cleaning the data, loading Spotify features for each song (more details in the next chapter), and downloading the songs audio, we were left with a dataset of nearly 4000 tracks, ranging in popularity from 33 million videos to 140,000. We can see the distribution of popularity measures from chartex in figure 2.

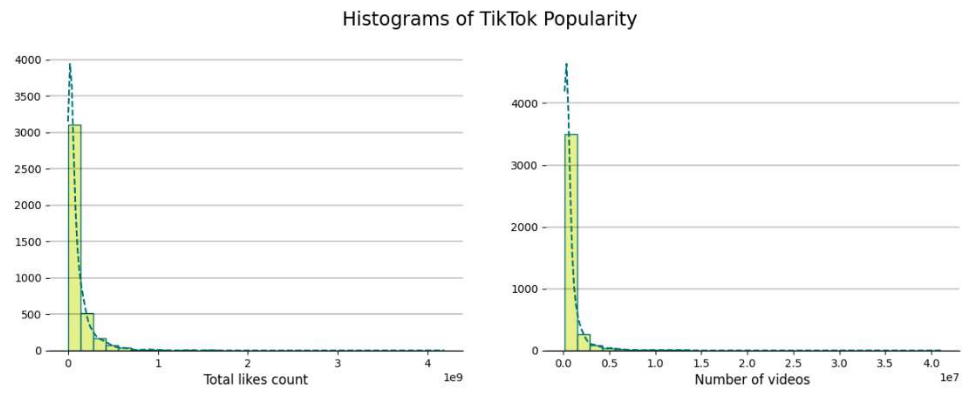


Figure 2- On the y-axis, the number of songs and on the x-axis, total likes for all videos that use this song on the left and number of videos posted in TikTok that uses this song on the right.

[1] https://chartex.com/tiktok-music-chart-top-songs-from-tiktok/sort/number-videos-desc

Feature Extraction:

The Spotify API gives us detailed info about each track, including popularity metrics, as well as high-level audio features like loudness, energy, and “acousticness.” A brief explanation of these features is available in the Spotify API documentation[2].

First let’s look at the popularity metrics provided by Spotify.

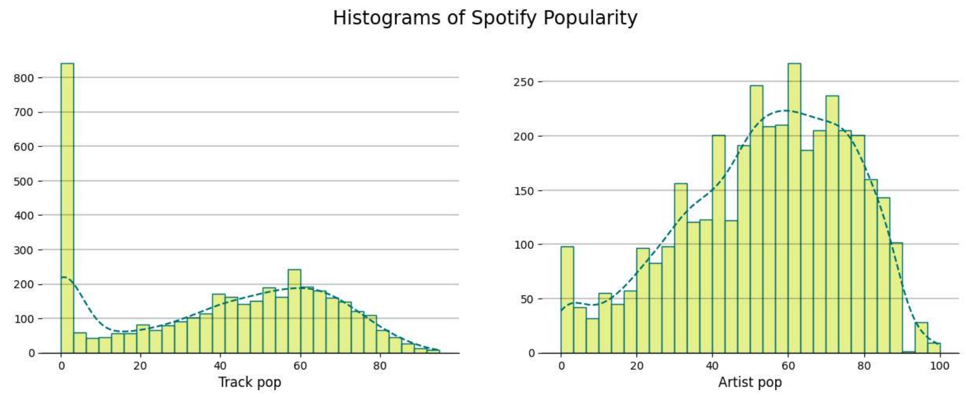
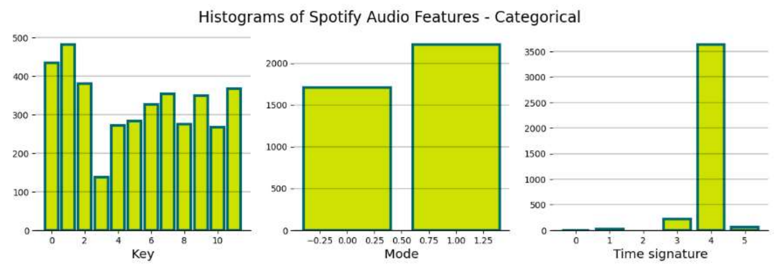


Figure 3-Spotify popularity mertics. On the left is Track pop and on the right is Artist pop

We can see these metrics look very different from the TikTok popularity metrics (videos and likes), and indeed we have seen that there is very little correlation between the two. Furthermore, it's important to emphasize that these metrics were deliberately excluded from our dataset after this initial analysis.

Now on to the Spotify-provided audio features. To gain insights from this data, we have generated histograms that visualize the distribution across the entire dataset, as can be seen in figure 4.

[2] https://developer.spotify.com/documentation/web-api/reference/get-audio-features



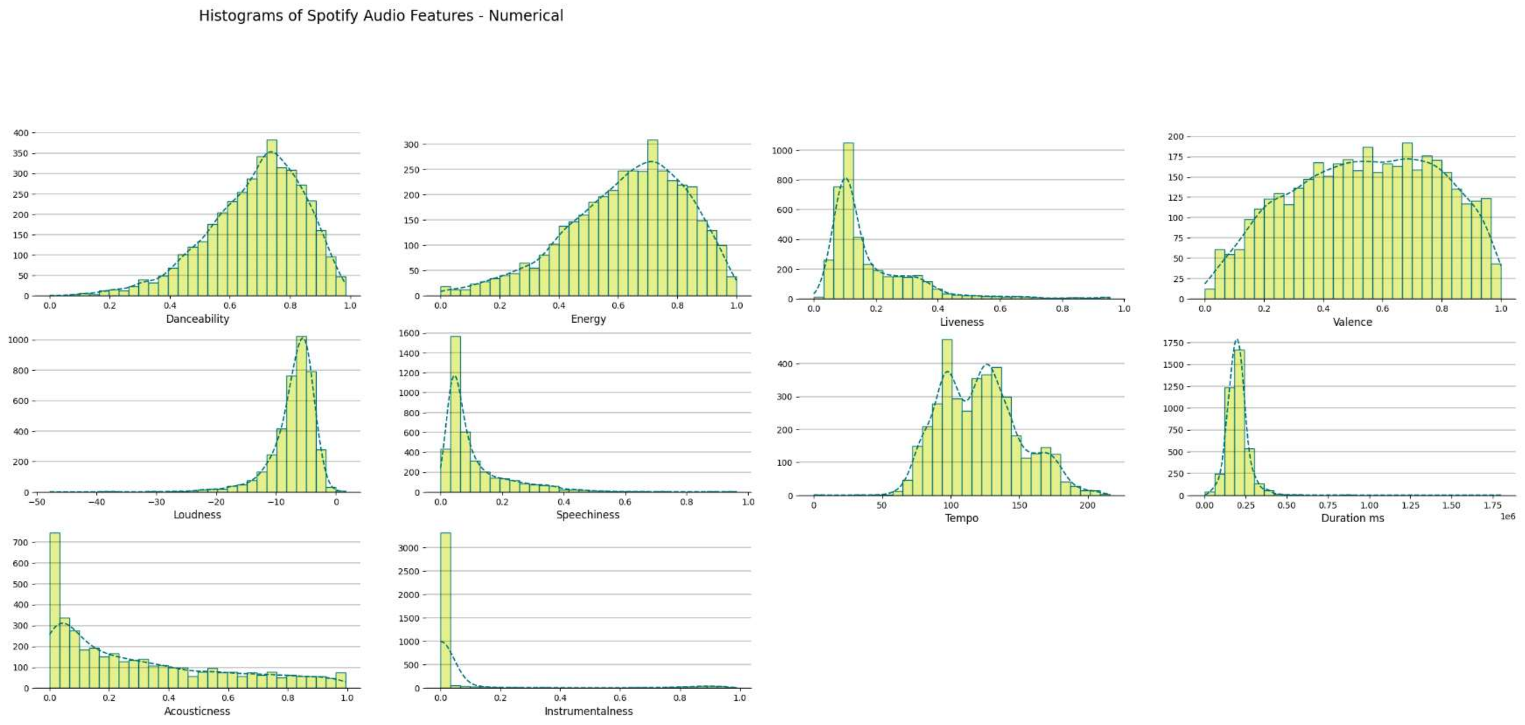


Figure 4-Spoitfy feature distribution, the categorical features in the upper graphs and numerical on the lower graphs.

Looking at these histograms, we can see that the songs in this dataset, which all have a substantial presence on TikTok (all with over 100,000 videos – We will talk about the possible problem with it in the conclusion section), share some common characteristics:

- They are energetic, high tempo, loud, and danceable.
- They are mainly music, as opposed to speech.
- They are not very long, usually under 5 minutes.
- They are usually recorded in a studio, not in live settings.
- They are not entirely acoustic.
- They overwhelmingly incorporate lyrics, making them non-instrumental.
- Nearly all follow a regular 4/4 beat.

In addition to these features, we have also extracted some more fundamental features that are popularly used in audio analysis. The features distribution can be seen in figure 5.

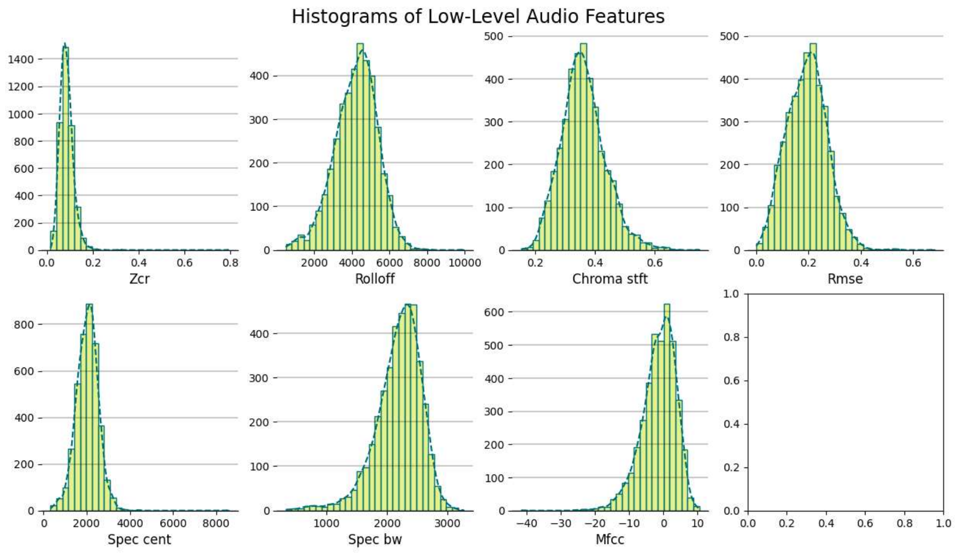


Figure 5-The distribution of the extracted features.

- **Zero Crossing Rate:** the frequency at which a signal changes from positive to negative or vice versa. Typically, it exhibits higher values in the presence of sharp, percussive sounds commonly found in genres like metal and rock.
- **Spectral Centroid:** the "center of mass" for a sound, calculated as the weighted average of the frequencies present in the sound, when the weights are the amplitudes of the frequencies.
- **Spectral Rolloff:** the frequency below which a specified percentage of the total spectral energy lies. In our case we used 85%.
- **Spectral Bandwidth:** measure of the spread of the frequency content (we used it with the default value $p = 2$ and therefore it is exactly like standard deviation for frequencies).
- **Chroma Short-Time Fourier Transform (Chroma STFT)**: represents the pitch content of an audio signal over time. For these histograms we have taken the mean value for each track.
- **Mel-Frequency Cepstral Coefficients (MFCC):** Arguably one of the most pivotal techniques in audio signal analysis, MFCCs distill complex audio data into a compact set of features, typically numbering between 10 and 20. They succinctly capture the overarching shape of a sound's spectral envelope. For these histograms we have taken the mean value for each track.
- **Root Mean Square Energy (RMSE):** quantifies the energy (amplitude) of a signal.

The pressing question remains: do these features suffice to differentiate songs that achieve extraordinary viral success from the rest of the pack?

We will now look at correlations between the different features.

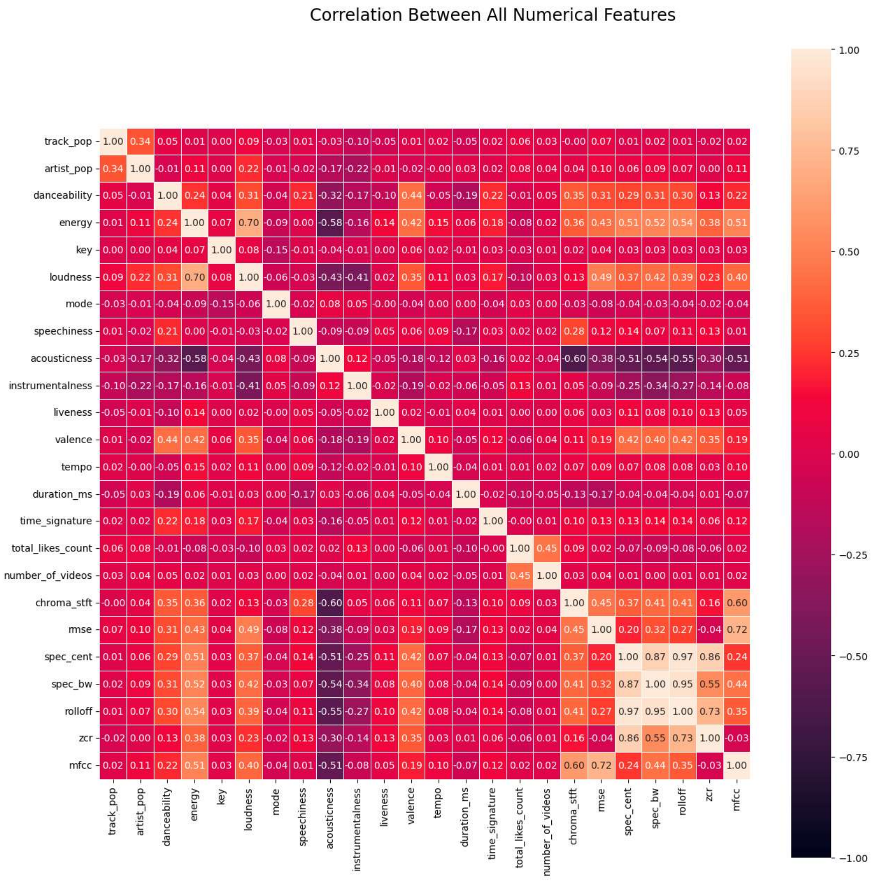


Figure 6-The correlation matrix between all numerical features.

Some takeaways:

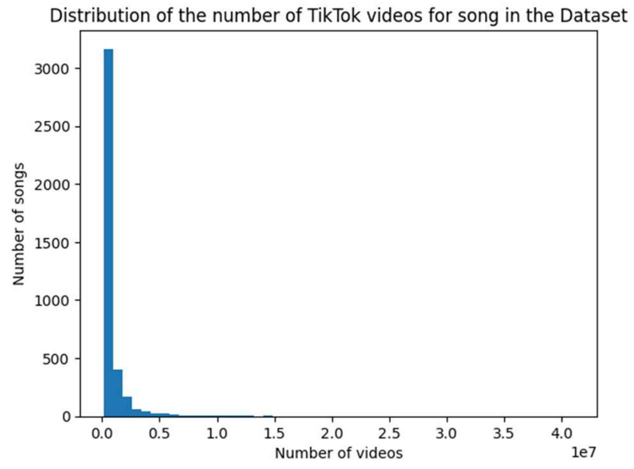
- Spotify's artist popularity metric demonstrates a strong correlation with track popularity, suggesting an interconnectedness between an artist's and a track's appeal on Spotify.
- The total number of TikTok likes exhibits a notable correlation with the total count of videos, indicating a clear association between user engagement and content creation.
- Interestingly, there are notable correlations between all our constructed features and energy, loudness and acousticness.
- Spectral centroid is highly correlated with spectral roll off and bandwidth: This makes sense because the spectral centroid is influenced by the distribution of frequencies in the spectrum. If most of the energy is concentrated in the high-frequency region (as indicated by a high spectral roll-off and wide spectral bandwidth), the spectral centroid will tend to be higher. Conversely, if the spectral content is concentrated in the low-frequency region, the centroid will be lower.
- Zero crossing rate is highly correlated with spectral centroid – Zero-crossing rate is related to the rapid changes in the signal's amplitude. When there are rapid changes (high zero-crossing rate), it often indicates a higher presence of

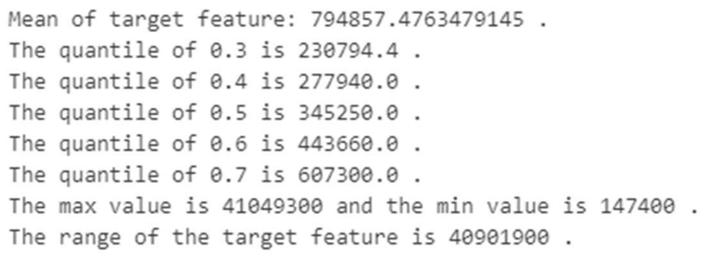
high-frequency components, which can also affect spectral roll-off, spectral bandwidth, and spectral centroid.

Intriguingly, none of the popularity metrics display significant correlations with any of the audio features.
This observation underscores the complexity of the challenge at hand, emphasizing the necessity of leveraging machine learning techniques to unravel the underlying patterns in our data.

Target Feature – Virality:

We measure the virality of songs by the number of videos in TikTok that used this song according to the chartex dataset we have. Let's look at the distribution of the number of videos feature in our dataset and few statistics about it:





```
Mean of target feature: 794857.4763479145 .
The quantile of 0.3 is 230794.4 .
The quantile of 0.4 is 277940.0 .
The quantile of 0.5 is 345250.0 .
The quantile of 0.6 is 443660.0 .
The quantile of 0.7 is 607300.0 .
The max value is 41049300 and the min value is 147400 .
The range of the target feature is 40901900 .
```

Figure 7-Distribution of the 'number of TikTok videos for song' feature on the left and important statistics about the distribution on the right.

We can notice that most (60%) of the songs has less than $5e5$ videos on TikTok and very few has >$1e7$ videos. This makes a lot of sense since very few songs are extremely popular. To create the binary virality feature from number of videos, we need a threshold. We decide to take $5e5$ as the threshold from 2 reasons:

1. This threshold allows the dataset to be balanced. In other words, the number of viral songs and the number of non-viral songs in the dataset is close and by that prevent the option of biased dataset (which can cause in the extreme case to get a model that classify all samples as non-viral, for instance, because by doing so its mis-classify small number of samples).
2. The number of videos require from song to be viral is still high and there are still more non-viral songs, as we would expect in reality.

To sum up, song with number of videos higher than $5e5$ will be classified viral and otherwise non-viral.

Cross Validation:

Cross Validation is a technique to tune the model's hyper-parameter without exploiting the test set.

The problem with using the test set for hyper-parameter tuning is that we would be left with no data to evaluate the trained model. Using the test set for both tasks would be a bad idea because the model is familiar with the samples in the test set (we 'overfit' the hyper-parameters to the test set) and thus the scores won't be reliable.

When doing cross validation, we are splitting the training set into few parts (folds) and then: For each fold, we are training the model on all the rest folds and evaluate it on the current fold. After doing that for all folds, we calculate the mean of the results on all folds, and this is the results of the CV as we can see in figure 8. These results are valid because the evaluation is done on unseen data. In addition, the test set is left untouched.

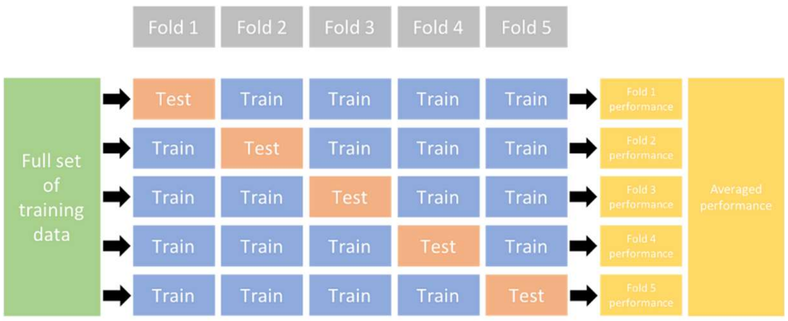


Figure 8-The idea of Cross Validation. The training set is devided into equal sized parts, and then all parts except one is used for training and the last part for validation.

With that algorithm, we can set hyper-parameter configuration and evaluate the model without using the test set.

Now for the search technique:
In few experiments, we deal with large number of hyperparameters so performing exhaustive search in the hyperparameters space become infeasible because the computations grow exponentially with the number of hyperparameters. In addition, applying "greedy" search in the hyperparameter space by running CV on each hyperparameter separately one after the other (when best hyperparameter values from previous running will take place in the next ones) wouldn't get good results. We think so because some hyperparameters are extremely connected, like the learning rate and the scheduler hyperparameters and thus setting the best previous values can cause significant limitation to the next hyperparameters and bad final configuration.

Therefore, we will do something different – we will perform a "heuristic" search on the graph by starting with reasonable configuration of hyperparameters and then change the configuration according to the results. For example, if we get an overfitted model, we should increase the weight decay or the dropout probability. With this way, we probably will get a sub-optimal but good configuration.

Focal Loss:

Focal loss[3] is a loss function that address class imbalance during training. It adds a dynamic factor to each cross-entropy component that helps the model to focus on **hard misclassified samples**. The following is the dynamic factor:

$$(1 - p_t)^{\gamma}$$

Where γ is an hyperparameter and p_t is the model's probability the sample is from the t class.

In the experiments, we will notice a behavior in which the models tend to classify the samples as non-viral and struggle to classify samples has viral, probably because ~65% of the samples are non-viral, and this is **exactly what focal loss supposed to handle**.

In the conclusion chapter, we will also show a possible reason why focal loss didn't work well in our case (problem 2).

[3] https://arxiv.org/pdf/1708.02002v2.pdf

Test with Spotify features:

We start simple: Instead of extracting the features from the audio, we used the premade Spotify features and our well-known features and train simple models, such as KNN, decision trees and simple MLP to classify the binary virality target feature.

With all models except the SVM, we worked on the raw features and didn't apply any feature mapping. With the SVM, we used the RBF feature mapping due to its strength.

The following table summarizes the results:

Name	Train accuracy	Test accuracy
KNN with K=8	0.691	0.595
SVM with RBF without CV	0.647	0.616
SVM with RBF with CV	0.647	0.616
Decision Tree without CV	0.930	0.527
Decision Tree with CV	0.669	0.605
AdaBoost without CV	0.649	0.617
AdaBoost with CV	0.647	0.616
MLP without CV with 64 nodes layer	0.723	0.577
MLP with CV with 256 nodes layer	0.685	0.611

The results aren't impressive. We will mainly focus on the MLP (Fully connected network) here but here are few conclusions for the rest of the models:

1. All models (that weren't overfitted) got almost the same results. The result on the training set is very close to the ratio of the non-viral songs from all songs in the training set - ~64.7%, which indicates the learnable pattern is not far from just classifying all samples as non-viral.
2. When the train accuracy is higher, the test accuracy tends to be low, which is a behavior of overfitting and not generalizing.

Now, we will explore the MLP in depth:
We first need to determine the architecture of the network. To do so, we will use a **confusion matrix**. A confusion matrix for binary classification is a matrix containing the number of TP, FP, TN, and FN (in both axis, 0 is non-viral-Negative and 1 is viral-Positive).
With the simple MLP structure, which is just three layers (width of 64 nodes) with ReLU activation function and SoftMax in the output, we got the matrix in figure 9-left. We can notice the model has high False negative error, meaning it classifies a lot of viral songs as non-viral. We think the model didn't really learn any underlying pattern but just exploited the fact that most of the data is tagged as non-viral. In other words, the models suffer from **underfitting**, probably because it doesn't have enough learnable parameters.

To solve this, we widened the layers to 256 nodes and with small changes in the learning rate and weight decay, we got the matrix in figure 9-right. We can notice that with figure 9-right, we got very good results on the **training set**. However, the capacity of the model is high and thus there is a risk to **overfit** (probably what

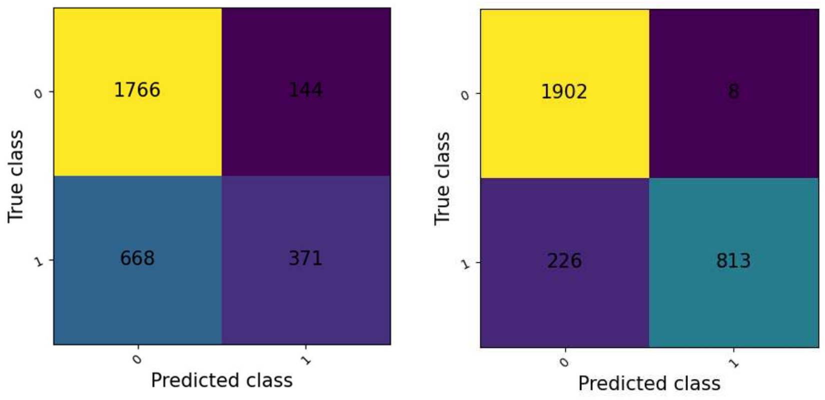


Figure 9-confusion matrices of the 64 nodes layers and the 256 nodes layers respctively on train set.

happens here). However, contrary to the first architecture, this one can learn (or memorize). To avoid memorizing, we will use techniques to prevent overfitting, such as dropout and regularization term.

After determining the model architecture, we perform cross validation on the learning rate and regularization term for the optimizer, step size and gamma for the scheduler and the class weights for the Focal loss.
As mentioned in the cross-validation section, due to large number of hyperparameters, we did not perform exhaustive search but type of "heuristic search". We did it both with and without Focal loss.

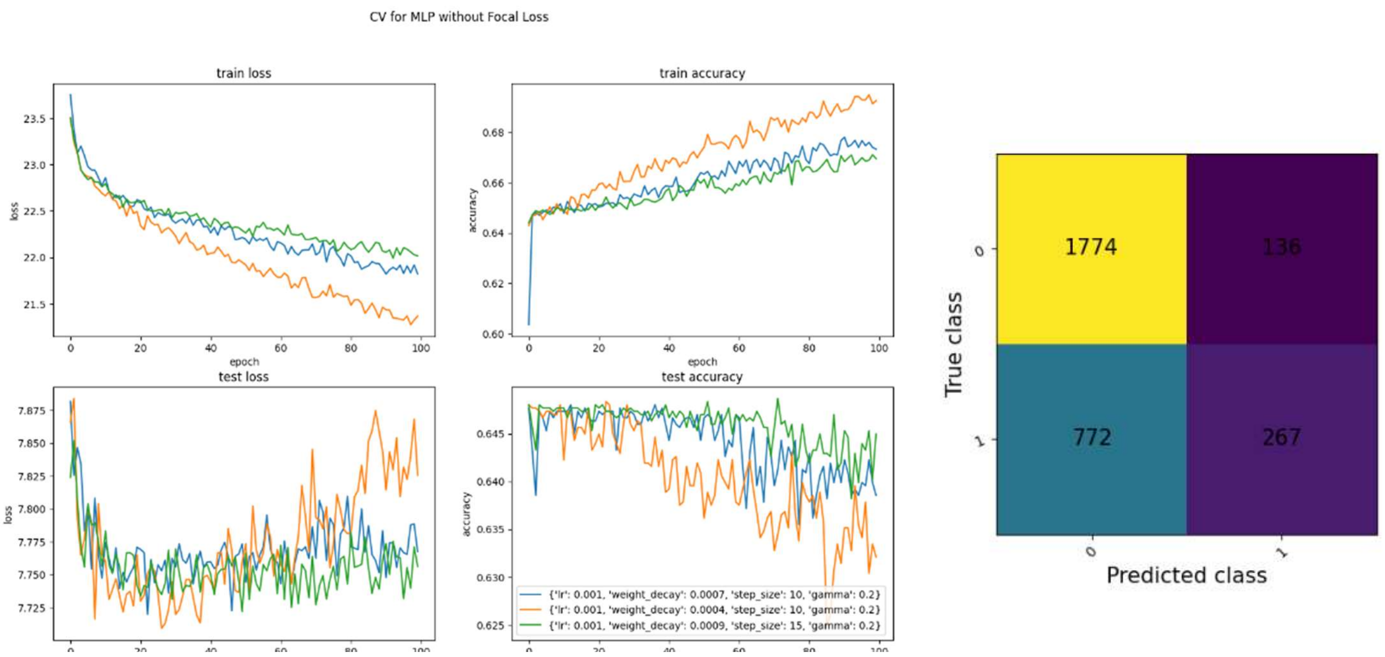
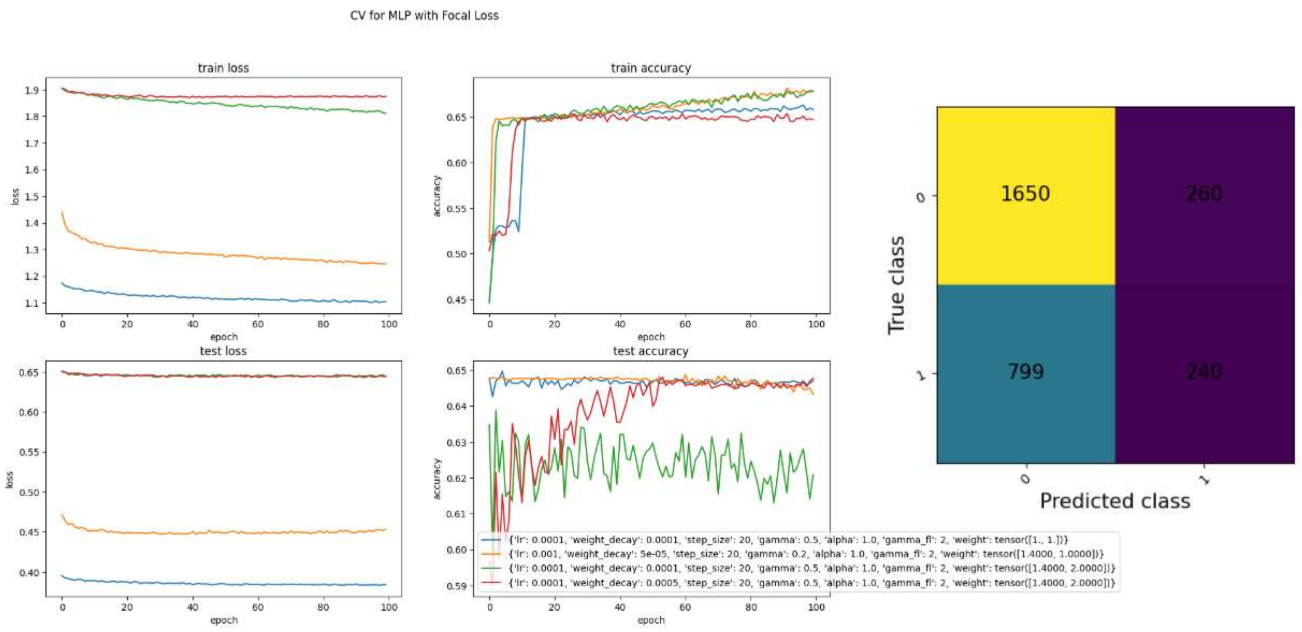


Figure 10- learning curves of CV for MLP without Focal Loss and confusion matrix of the best configuration (green curve)

With Figure 10 we use the basic cross entropy loss. In all configurations, we can notice that when the train loss decreases (and the train accuracy increases), the validation loss is increasing (and the test accuracy is decreasing). In other words, the models suffer from overfitting. However, even when we try to deal with it by increasing the regularization term or with dropout, we just get an underfitted model (green config), both validation and train score changed very little. We can also see the underfitting in the confusion matrix where about ~75% of the viral songs classified as non-viral (on the training set) with the configuration that supposed to be the best (on validation) from all those we tested. One possible reason is our small dataset, and we will discuss deeply about it in the conclusion section.

With Figure 11 we use Focal loss and weighted mean. We can notice a very small improvement with most configurations. It seems that when we punish the model more on misclassified viral samples by using only focal loss (the blue curve), the model struggles to train and there is very little change in the loss scores. However, even if we add weights to different mistakes, the model scores don't change as we expected. In addition, even the confusion matrix on the red curve (which trained both with focal loss and weighted losses that force him to pay attention to misclassified viral samples) is a bit worse than without focal loss – more than 75% of the viral samples were



```
{'lr': 0.0001, 'weight_decay': 0.0001, 'step_size': 20, 'gamma': 0.5, 'alpha': 1.0, 'gamma_fl': 2, 'weight': tensor([1., 1.])}
{'lr': 0.001, 'weight_decay': 5e-05, 'step_size': 20, 'gamma': 0.2, 'alpha': 1.0, 'gamma_fl': 2, 'weight': tensor([1.4000, 1.0000])}
{'lr': 0.0001, 'weight_decay': 0.0001, 'step_size': 20, 'gamma': 0.5, 'alpha': 1.0, 'gamma_fl': 2, 'weight': tensor([1.4000, 2.0000])}
{'lr': 0.0001, 'weight_decay': 0.0005, 'step_size': 20, 'gamma': 0.5, 'alpha': 1.0, 'gamma_fl': 2, 'weight': tensor([1.4000, 2.0000])}
```

Figure 11- learning curves of CV for MLP with Focal Loss and confusion matrix of the best configuration (red curve)

classified as non-viral. It is very surprising considering that focal loss and weighted loss should deal with this exact problem (punish model more on misclassified 'hard' labels).

It seems that focal loss, in our case, just makes the train worse and therefore we will use the green configuration without focal loss for the final training. The learning curve can be seen in figure 12.

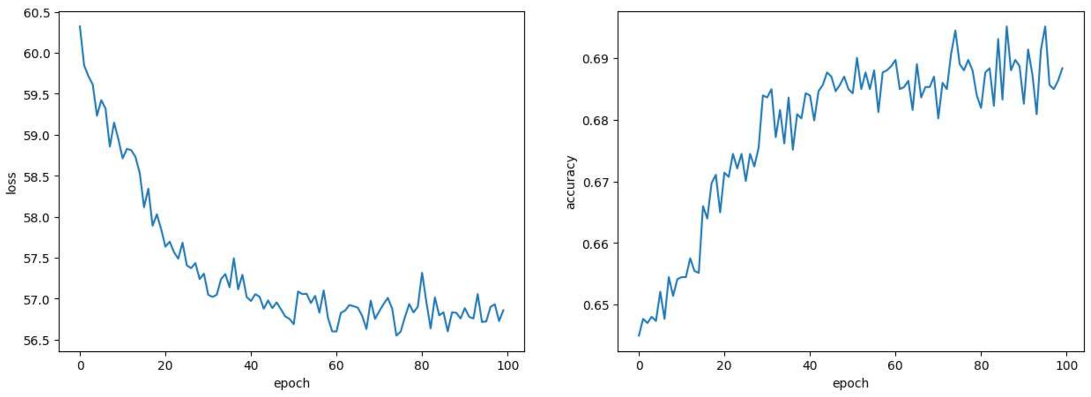


Figure 12-learning curve for MLP without focal loss with the blue configuration

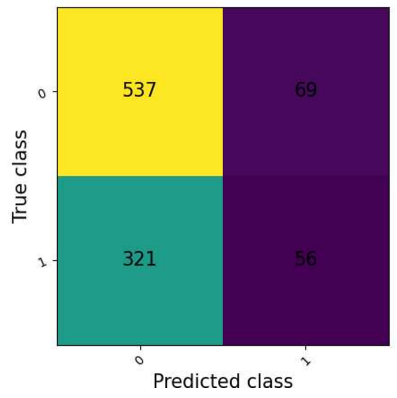


Figure 13-confusion matrix on test set (right)

We can notice in figure 13 that the model continues to suffer from the problem we face in the training set, where the model has a high number of viral songs that are mis-classified as non-viral.

To conclude, the results we got in both of those 2 experiments from the Spotify features are bad. As mentioned previously, we will discuss the possible reasons in the conclusion section.

Background on Iterative learning Algorithms:

Iterative learning Algorithms are Iterative algorithms to find minimum point of specific loss function.

Let X be tensor of our training set, W tensor of the weights of the model and $\mathcal{L}(W;X)$ the loss function. To be clear, X is constant tensor and $\mathcal{L}$ is dependent only on W. With that in mind, those algorithm uses $\frac{\partial \mathcal{L}}{\partial W}$ (the gradient of $\mathcal{L}$ with respect to W) in a specific point of W and made a step in the opposite direction to it (in general, some algorithm, like Adam, do smarter steps based on the gradient). By doing so we are going in the steepest downhill direction – decreasing $\mathcal{L}$ and finding better[4] weights for the model.

In most cases, because the number of samples in the training set is big, we will divide the training set into min-batches with small, fixed size, and for each epoch, we will calculate the loss on every mini-batch and do the gradient step separately. By doing so we prevent heavy computations and avoid loading the entire training set to the memory, that in most cases in not feasible.

Background on Audio:

Let's start by introducing few terms:

- **Sampling rate:** the number of samples that were taken from the original audio signal per second.
- **Channels:** the number of microphones collected data from the audio signal. One microphone audio file is called 'mono' and two microphones file is called 'stereo'.

With those terms explained, we will now introduce the basic structure of audio files. When uncompressed, the audio files are just tensor of size

$$Channels \times (Time * SamplingRate)$$

Where $Time$ is the length in seconds of the audio file.
This tensor is a discrete representation of the audio signals for each channel.
The '.wav' files are like that.
The '.mp3' and '.mp4' are compressed and we will use those in our project. When we uncompress those formats, we get a similar representation as above.

One important visual representation of the audio signal is the spectrogram, and we will explain extensively about it in the next chapter.

[4] by 'better' we mean that with those weights, the loss on the training set is lower and thus the model did better on the training set.

Background on Spectrograms:

Deep learning models rarely use raw audio directly as input – the common practice is to convert the audio into a spectrogram.
A spectrogram is a graph that provides a visual representation of a signal. Time is shown along the horizontal axis, while frequency is displayed along the vertical axis. The color in each pixel represents the amplitude of the 'y' frequency in the 'x' time. Each vertical section of the spectrogram essentially shows the spectrum of the signal at a specific point in time. It reveals how the strength of the signal is distributed across the various frequencies present at that moment.

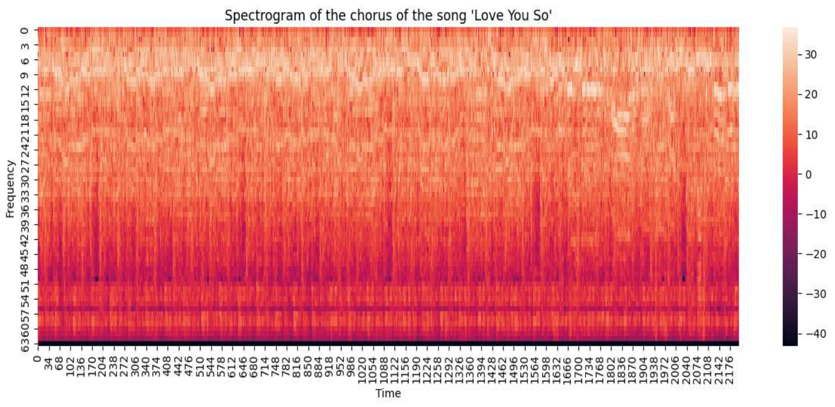


Figure 14-Mel spectrogram of the chorus of the song 'Love You So'

Spectrograms are typically generated using the Short Time Fourier Transform (STFT) algorithm. This technique involves taking a small window of the entire audio signal, say around 100 samples, and running it through the Fast Fourier Transform (FFT). This, as we know, transforms the signal into the frequency domain. Then, it hops over to the next window of audio, perhaps with a step size of 50 samples (note that there's overlap between adjacent windows).

We will heavily use the spectrogram representation in our project from two main reasons:

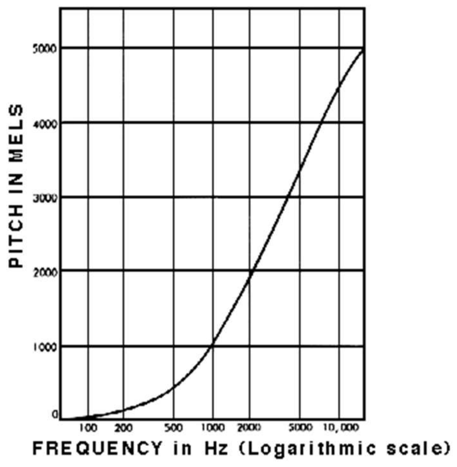
1. For the CNN, we need an image, and the spectrogram is the reasonable way to represent audio as an image – it is compact way to capture the essential features of audio data as an image.
   For the RNN, the high sampling rate (22kHz) for the 30 seconds audio creates extremely long sequences that simple RNN/LSTM/GRU can't handle properly according to research. With spectrogram, we shrink this sequence into length of 2176.
   In both cases, the internet highly recommends using spectrogram.
2. The human hearing is much like a spectrogram – it can sense only discrete range of frequencies with special hair-like cells[5]. In other words, the human ear takes audio signal and decomposes it to basic waves with constant

[5] https://www.nih.gov/news-events/nih-research-matters/hearing-different-frequencies

frequencies. As a result, by using spectrogram, we mimic the behavior of the human ear and potentially will get better results.

Mel Spectrogram:

Humans are better at detecting differences in lower frequencies than higher frequencies. The Mel scale transforms the frequencies in such a way that sounds of equal distance from each other on the Mel Scale, also “sound” to humans as they are equal in distance from one another. In the normal Hz scale, this is not the case because increasing from 500Hz to 1000Hz would sound much different than from 10,000Hz to 10,500Hz although, the increase is similar. The transformation is looking as follows:



$$m = 1127 \cdot \log\left(1 + \frac{f}{700}\right)$$

Figure 15-Mel transformation

The Mel spectrogram is normal spectrogram, but instead of frequency, we use the Mel scale and instead of amplitude we use decibel scale. They are usually used for deep learning models rather than a simple spectrogram.

Choosing the Chorus:

The chorus is a repeating section that encapsulates the core message of the song. It's typically the most potent, dynamic, and unforgettable part of the track. This is precisely why most TikTokers gravitate towards using the chorus for their videos. Given its prominence, utilizing this section of the audio as input for the model is a straightforward path to achieving optimal results. Additionally, in terms of computational resources, working with a shorter audio file (specifically, just the chorus rather than the entire song) and consequently generating a smaller spectrogram demands fewer computations.
For these reasons, we've chosen to focus on the chorus in our project.
Now, the challenge is how to pinpoint the chorus within the audio file. To tackle this, we'll employ a Python library called 'Pychorus'. The fundamental concept behind chorus detection lies in the fact that choruses exhibit repeated patterns throughout the song. By identifying the segment that repeats most frequently, we can reliably identify the chorus.

We need to make sure that all the data is of the same length, for inputting to our models. Through trial and error, we've determined that a 30-second duration strikes the right balance between computational efficiency and performance. Going beyond this threshold doesn't yield significant performance improvements, making it the optimal choice.

The General System Pipeline:

In the **inference stage**, the system consists of 4 main components:

1. Finding the **chorus** in the audio if exists and taking 30 seconds from it. If not exists, take 30 seconds from the middle of the audio file.
2. Convert the 30 seconds audio into **spectrogram**.
3. Apply the **Feature Extractor** on the spectrogram, CNN, RNN, etc. This step will output vector of learned features from the spectrogram.
4. Apply **Classifier** (in our case Fully connected layers) on the feature vector from the previous step and output probability distribution on the two 'labels': viral or not viral.

The pipeline will look as follows:

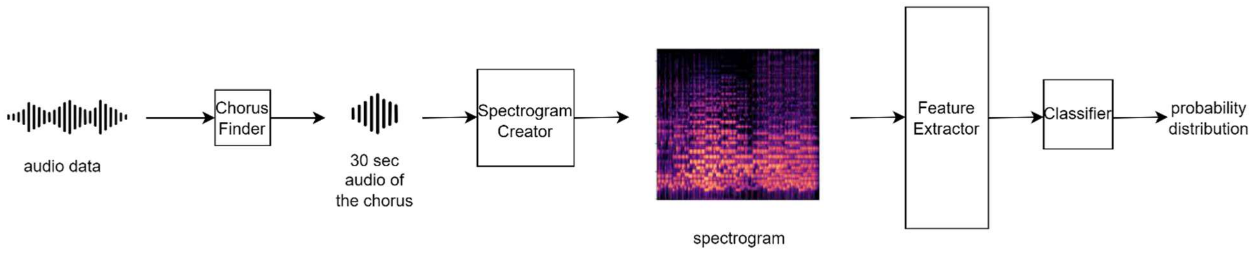


Figure 16-The full system pipeline

However, to avoid redundant computation and because only the last two components are learnable, in the **training stage**, we premade all the spectrograms of all songs' chorus in the dataset and only load them to memory to train the feature extractor and classifier.

Therefore, the pipeline in this phase would look as follows:

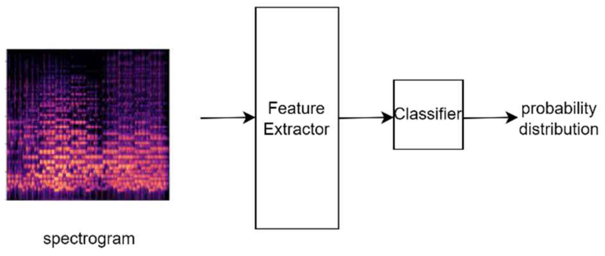


Figure 17-The system training pipeline

Working with CNN:

The convolutional neural network (CNN) is a type of deep learning model and is mainly used for vision tasks.
The main component in the CNN is the Convolution layers, which uses small multi-dimensional filters (tensors with learnable weights) that slides on the bigger input tensor and for each location, calculate the weighted sum of the values of the input that overlap with the filter as we can see in figure 18.

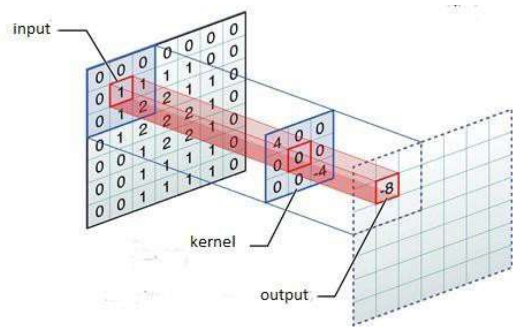


Figure 18-Convolutional layer filter (kernel) behavior.

The importance of the CNN come from two reasons:

1. The filters weights are shared between all positions and as a result, the number of parameters (weights) decrease dramatically compared to the MLP on large inputs like images. This makes the training of the model easier and prevents overfitting.
2. The share of parameters uses the idea of invariance to location of objects and shapes in image (domain specific knowledge). In other words, each filter is somewhat trained to find objects or shapes in the image – In the first layers those shapes are simple like edges, lines, etc. and in the more advanced layers they can find complex structures in the image.

One can ask how vision models can be used and help with audio analysis and the answer for that is divided into 2 main parts:

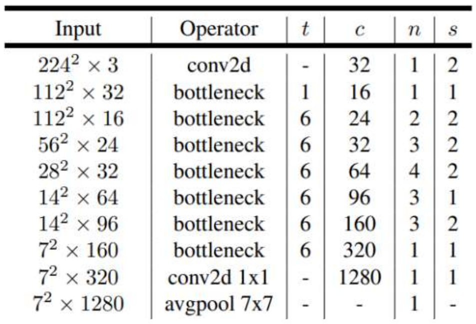
1. The model input – as discussed extensively, audio signal can be represented as spectrogram and the spectrogram can be plugged into the model.
2. Why domain specific knowledge from the vision world would be beneficial in the signal world – empirically, studies and articles on the internet shows impressive results of CNN in the field of audio classification. In addition, fine tuning of CNN models which trained on vision datasets show SOTA[6] results in audio classification.

Those are the reasons that bring us to try CNN.

[6] https://arxiv.org/pdf/2007.11154.pdf

The Model:

We chose to work mainly with pretrained MobileNet V2 model (that used in vision tasks and trained on vision datasets) for our classification task after we tried also VGG11 and AlexNet and we find out it got the best results. The network contains many deep learning tricks like bottleneck blocks. The model architecture can be found in figure 19.



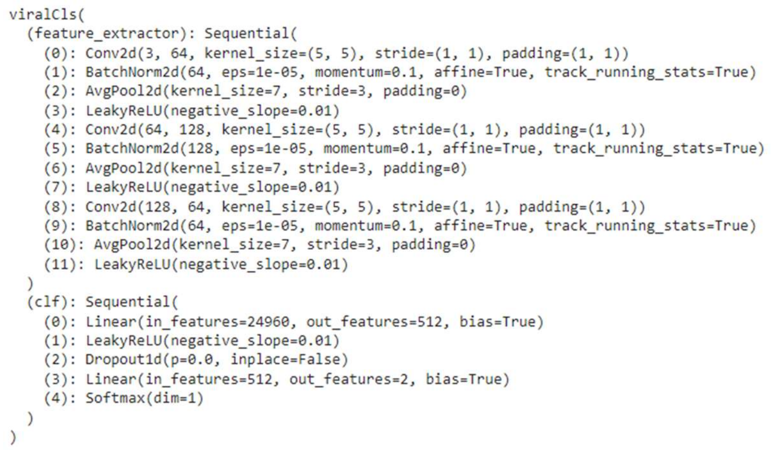
Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-

Figure 19 – MobileNet V2 architecture. t is the expansion factor, c is the number of filters, each line repeated n times and s is the stride of the first block of that type.

Experiments with CNN:

In this section, we will test the advantage using focal loss compared to cross entropy with CNN models. In addition, we will also examine the importance of **well-designed architecture** of CNN model compared to simple shallow models.

We will start with the latest. To examine the effect of simple CNN models and to play with the CNN layers, Conv2D, AvgPool2D and others, we start with a very simple model that we created and tried to overfit it on 16 sample batch. The model structure can be seen in figure 20. We were surprised to find out that even with ~13M learnable

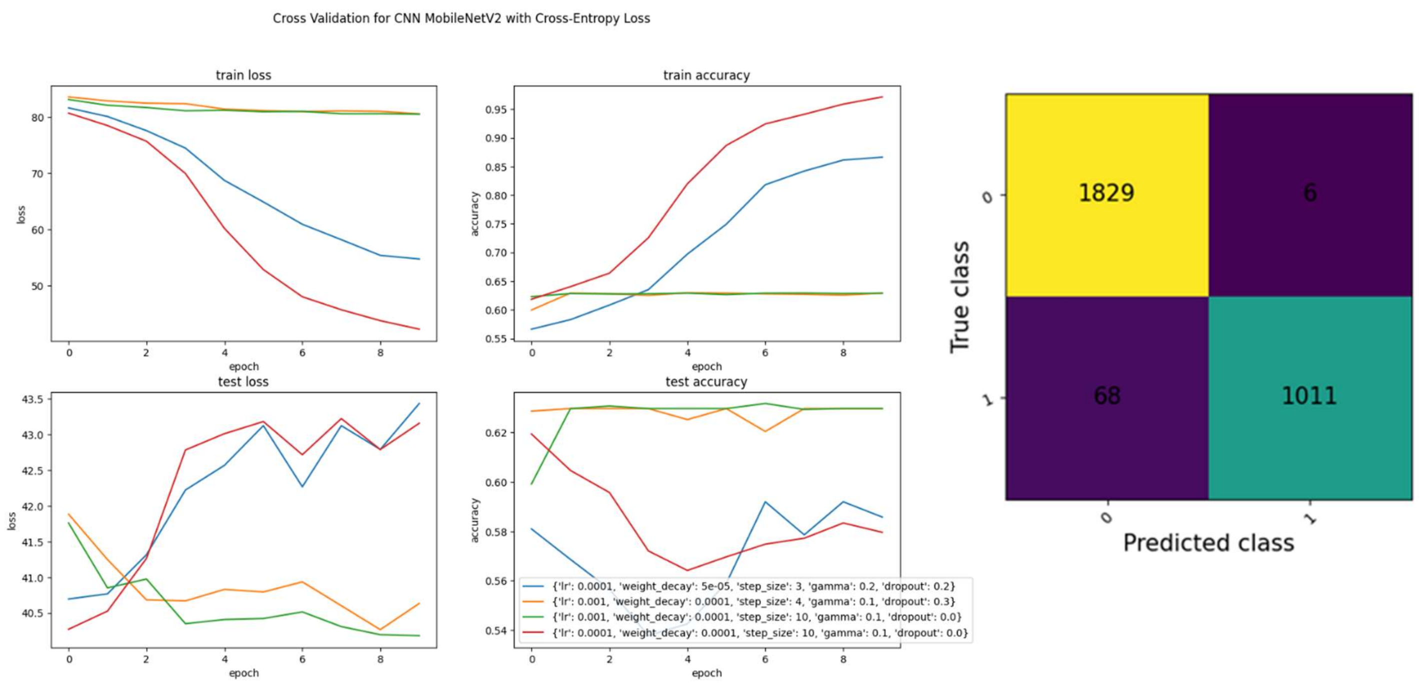


```
viralCls(
  (feature_extractor): Sequential(
    (0): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (2): AvgPool2d(kernel_size=7, stride=3, padding=0)
    (3): LeakyReLU(negative_slope=0.01)
    (4): Conv2d(64, 128, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (6): AvgPool2d(kernel_size=7, stride=3, padding=0)
    (7): LeakyReLU(negative_slope=0.01)
    (8): Conv2d(128, 64, kernel_size=(5, 5), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    (10): AvgPool2d(kernel_size=7, stride=3, padding=0)
    (11): LeakyReLU(negative_slope=0.01)
  )
  (clf): Sequential(
    (0): Linear(in_features=24960, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Dropout1d(p=0.0, inplace=False)
    (3): Linear(in_features=512, out_features=2, bias=True)
    (4): Softmax(dim=1)
  )
)
```

Figure 20-Our CNN model architecture. Notice it is very shallow model compared to the MobileNet V2

params the model couldn't overfit even 16 samples batch. We are certain the failure is not a code problem because in the first epoch, the model changed and after that didn't change at all. With further investigation, we realized that after the first epoch the gradients became zero, meaning the model reach the local minima but this minimum was very bad in terms of model accuracy – 68.75% and considering the fact we wanted to be overfitted. This is unacceptable result. On the other hand, the ~2M params of MobileNet V2 could overfit this batch easily. As a result, we realize that in the CNN world the architecture of the model is extremely important and decide to continue only with the popular MobileNet V2 model.

Now for the CV of the deep MobileNet V2 model:
we present the outcomes of a cross-validation procedure applied to the **pretrained MobileNet V2** model with and without focal loss. Employing 3-fold cross-validation over a span of 10 epochs, we aimed to evaluate the model's performance rigorously, starting with cross entropy.



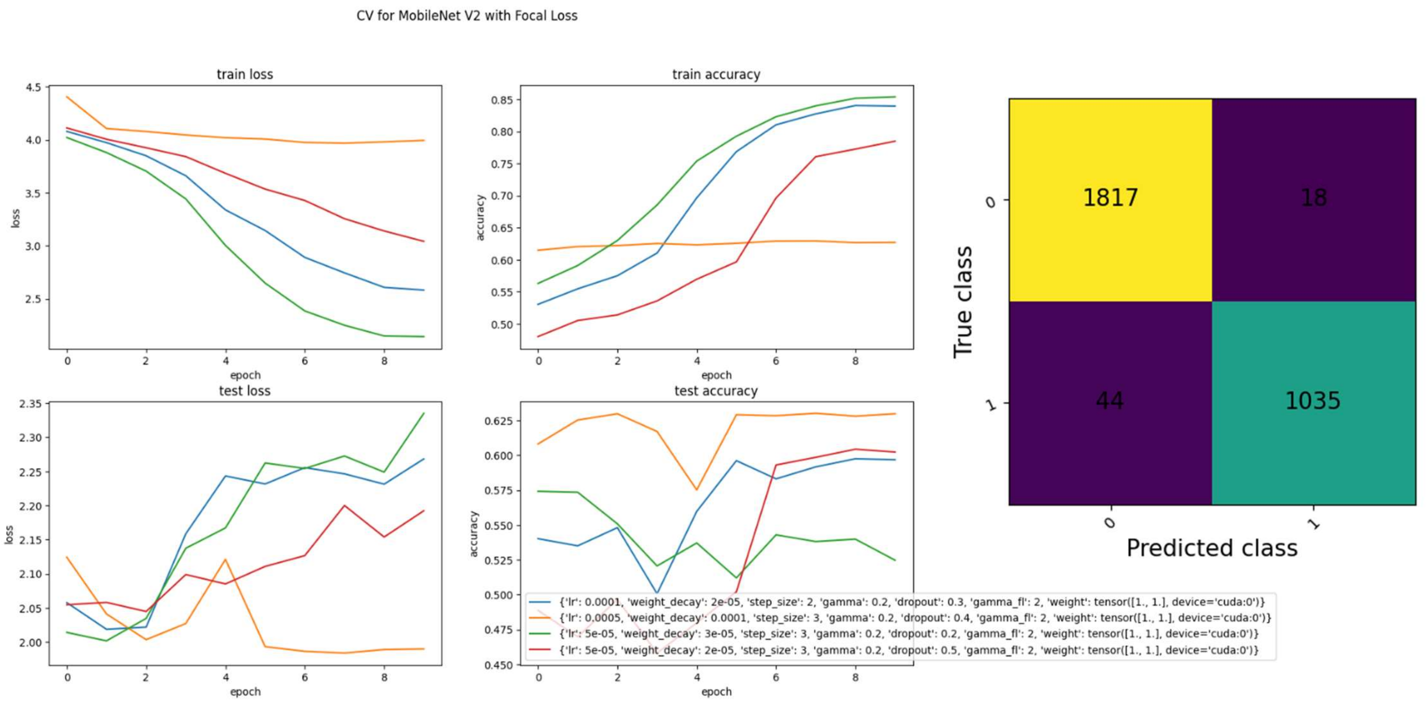
{'lr': 0.0001, 'weight_decay': 5e-05, 'step_size': 3, 'gamma': 0.2, 'dropout': 0.2}
{'lr': 0.001, 'weight_decay': 0.0001, 'step_size': 4, 'gamma': 0.1, 'dropout': 0.3}
{'lr': 0.001, 'weight_decay': 0.0001, 'step_size': 10, 'gamma': 0.1, 'dropout': 0.0}
{'lr': 0.0001, 'weight_decay': 0.0001, 'step_size': 10, 'gamma': 0.1, 'dropout': 0.0}

Figure 21- Cross validation learning curves for MobileNet V2 model with Cross entropy loss on the left and confusion matrix on the training set of the full trained model with the blue configuration (except the step size is 5 and not 3) on the right.

The red and blue configs show overfitting behavior, probably due to the low learning rate which allow them to go lower in the loss ("enter narrow valleys") and also the low regularization on the training. On the other hand, the green and orange configs are underfitted. We can see that they experience very little change in the training and accuracy loss, and this is probably because the regularization on them is too strong, and the learning rate is high. As before, when we put low regularization on the training, the model memorizes the training set and when we put some regularization it struggles to learn.
We changed the step size hyperparameter from 3 in the CV to 5 in the full training because while the CV was just for 10 epochs, the full training was 20 so we tried to have a similar ratio between them in both cases.
From the confusion matrix we can also see the model with the blue configuration get more FN errors then FP and especially relative to the number of non-viral and viral samples but in general the error on the training is very low. We hope the focal loss would help with it.

Now we will move to the focal loss. We thought that because there is some imbalance in the virality feature in the training set (64% of the training set is non-viral and 36% is viral), maybe focal loss would help to solve this.

As before, we run cross validation to tune the model and training hyperparameters. The results can be seen in figure 22.



- {'lr': 0.0001, 'weight_decay': 2e-05, 'step_size': 2, 'gamma': 0.2, 'dropout': 0.3, 'gamma_fl': 2, 'weight': tensor([1., 1.], device='cuda:0')}
- {'lr': 0.0005, 'weight_decay': 0.0001, 'step_size': 3, 'gamma': 0.2, 'dropout': 0.4, 'gamma_fl': 2, 'weight': tensor([1., 1.], device='cuda:0')}
- {'lr': 5e-05, 'weight_decay': 3e-05, 'step_size': 3, 'gamma': 0.2, 'dropout': 0.2, 'gamma_fl': 2, 'weight': tensor([1., 1.], device='cuda:0')}
- {'lr': 5e-05, 'weight_decay': 2e-05, 'step_size': 3, 'gamma': 0.2, 'dropout': 0.5, 'gamma_fl': 2, 'weight': tensor([1., 1.], device='cuda:0')}

Figure 22-Cross validation learning curves for MobileNet V2 model with focal loss on the left and confusion matrix on the training set of the full trained model with the red configuration (except the step size is 5 and not 3) on the right.

Like before, while we succeed in learning the training set, we failed in generalizing to the validation set in 3 out of the 4 configs. In addition, when we apply regularizations on the model such as weight decay and dropout, it just failed to learn the training set, while remaining with bad generalization (the orange curve). The red curve, while start bad, show improvement over the epochs in the validation accuracy and at the same time didn't overfit aggressively to the training set but still learns. This is the reason we choose this configuration for the confusion matrix. As we can notice from it, the confusion matrix on the training set is relatively balanced – there are less FN and more FP errors compared to the confusion matrix without the focal loss. However, we are concerned that the reason for that is just general increase in the number of samples that classified has viral and not a real learning.
As before, we change the step size from 3 to 5 to stick to the epoch's ratio change.

We decide to continue with the focal loss model for the test set evaluation. The results can be seen in figure 23, 24 and the following table.

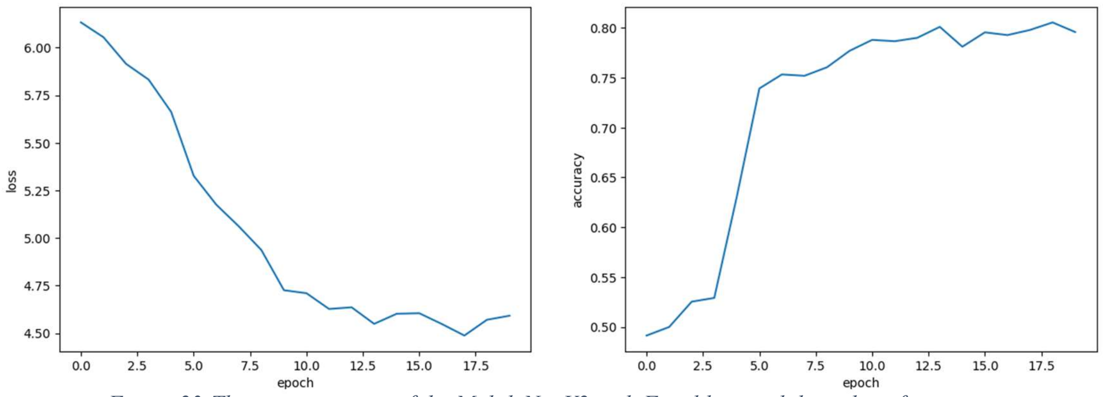


Figure 23-The training curve of the MobileNet V2 with Focal loss and the red configuration

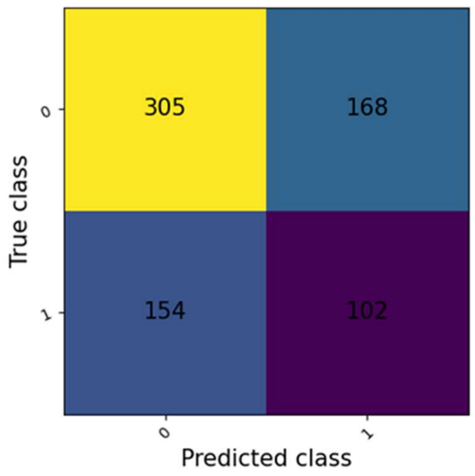


Figure 24-The confusion matrix on the test set of the above trained model

To sum up, the results are:

Train loss	4.591
Train accuracy	0.795
Test loss	1.583
Test accuracy	0.558

As we can see, the results are not good. The model suffers from overfitting (high train accuracy and low test accuracy) and there is a significant imbalance in the confusion matrix that the focal loss surprisingly could not handle.

Working with RNN:

Background on RNN:

The recurrent neural network (RNN) is type of deep learning model used mainly with sequential inputs, like NLP.
The basic idea in RNN is that the RNN unit get both input and hidden state and return both output and new hidden state. With this basic unit, we can pass sequentially on the input, entering each unit from the input to the RNN with the hidden state of the previous unit.
A big drawback with the simple units for RNN is that with long sequences, the training could suffer from vanishing gradients and thus would struggle to learn. In addition, the information gathered from the beginning of the sequence could fade from the hidden state at some point in the sequence, which could be harmful in some cases (like when the input from the beginning is still important in the end).
To solve those problems, the Long Short Time Memory (LSTM) was created. It works like the above, but the basic unit has a structure that helps to solve those problems. The structure is shown in figure 25 and the idea (in general) behind it is that the model can learn how much to pass through from the old hidden state and from the new input to the new hidden state and by that able to keep important data for longer sequences.

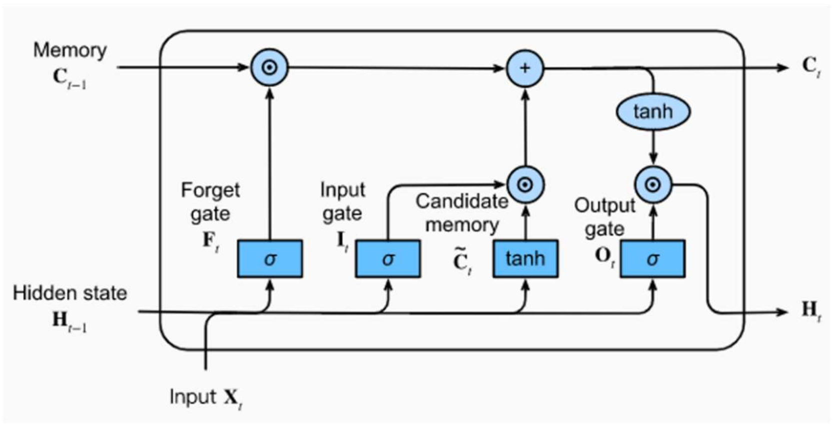

Memory $\mathbf{C}_{t-1}$
$\mathbf{C}_t$
tanh
Forget gate $\mathbf{F}_t$
Input gate $\mathbf{I}_t$
Candidate memory $\tilde{\mathbf{C}}_t$
Output gate $\mathbf{O}_t$
σ
tanh
Hidden state $\mathbf{H}_{t-1}$
$\mathbf{H}_t$
Input $\mathbf{X}_t$

Figure 25-LSTM unit structure

Our use of the RNN:

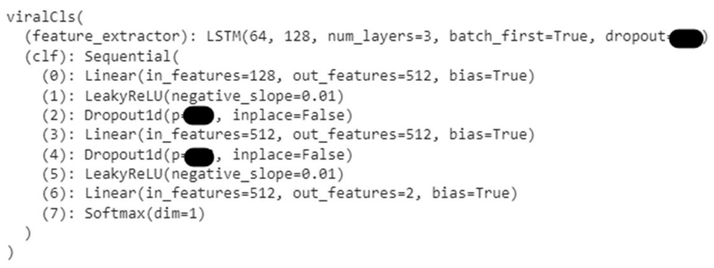
Although spectrograms are more likely to be used as images, they also can be interpreted has a sequential input by **referring to each column of the 2D tensor as a unit in the sequence** (each unit is a decomposition of the signal to frequencies and amplitudes in a window of time). This makes the model input much closer to the human hearing – in addition to what explained before about the human hearing and spectrograms, with RNN, the input will be process in a sequential order, much like a human being that would hear the song.
We will use this interpretation of the spectrogram with our RNN model.

The Model:

The LSTM model has few hyper-parameters related to the architecture of the model: the size of the hidden state, the number of layers of the LSTM and dropout probability. In addition, the following classifier has hyper-parameters such as number of layers and width of each layer.
There is no clear strategy we familiar with or read about on the internet to determine the model structure, so we did trial and error while keeping in mind the following ideas:

1. The hidden state contains all the relevant data of the song. Therefore, choosing a small size hidden state might cause underfitting because essential information would be lost. On the other hand, large hidden state will cause overfitting because the RNN will be able to memorize the song.
2. Using multilayer LSTM helps the model to extract useful features from the spectrogram because it can create different representations of the input between LSTM layers and then extract the right features for the classification.
3. Like with 1, large layer of the classifier or too many layers can cause overfitting. On the contrary, thin layers or small number of them can prevent from the model to learn the complex pattern between the features and cause underfitting.

We decide to go with the following architecture:



```
viralCls(
  (feature_extractor): LSTM(64, 128, num_layers=3, batch_first=True, dropout[redacted])
  (clf): Sequential(
    (0): Linear(in_features=128, out_features=512, bias=True)
    (1): LeakyReLU(negative_slope=0.01)
    (2): Dropout1d(p[redacted], inplace=False)
    (3): Linear(in_features=512, out_features=512, bias=True)
    (4): Dropout1d(p[redacted], inplace=False)
    (5): LeakyReLU(negative_slope=0.01)
    (6): Linear(in_features=512, out_features=2, bias=True)
    (7): Softmax(dim=1)
  )
)
```

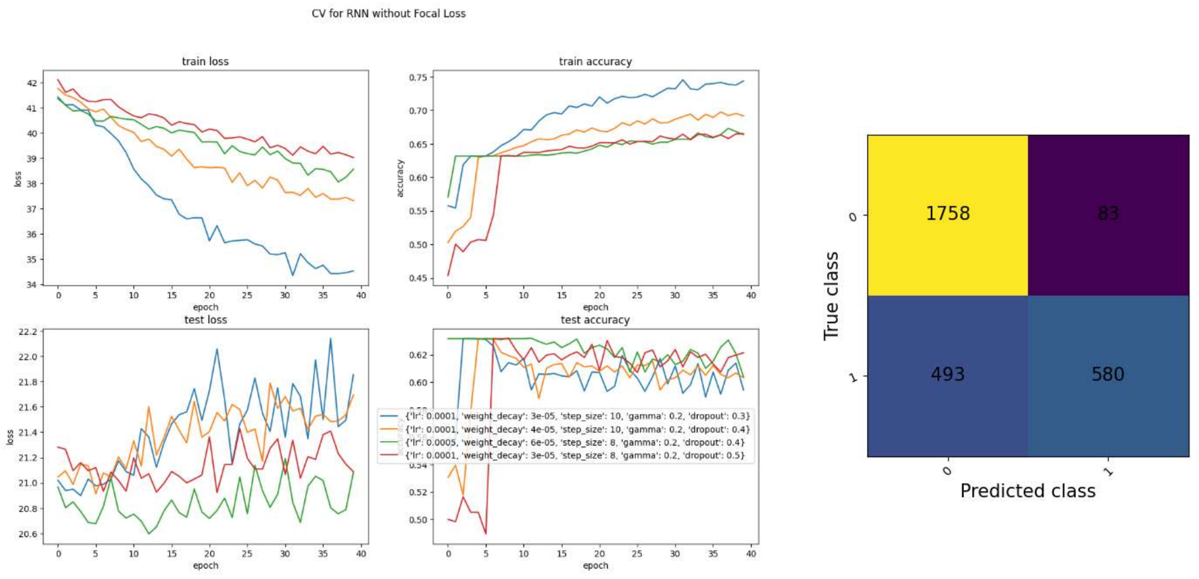
Figure 26-RNN based model architecture. The dropout will be determine in the CV.

Experiments with RNN:

Like with CNN and the simple MLP, we did cross validation on the model hyper-parameters both with Focal loss and with Cross entropy loss and test the trained model.

We run CV on the learning rate and regularization term factor (weight decay) for the optimizer, the number of steps and gamma factor for the scheduler, the gamma exponent and the weights for the focal loss (if used) and the dropout in the model architecture.

The results are the following:

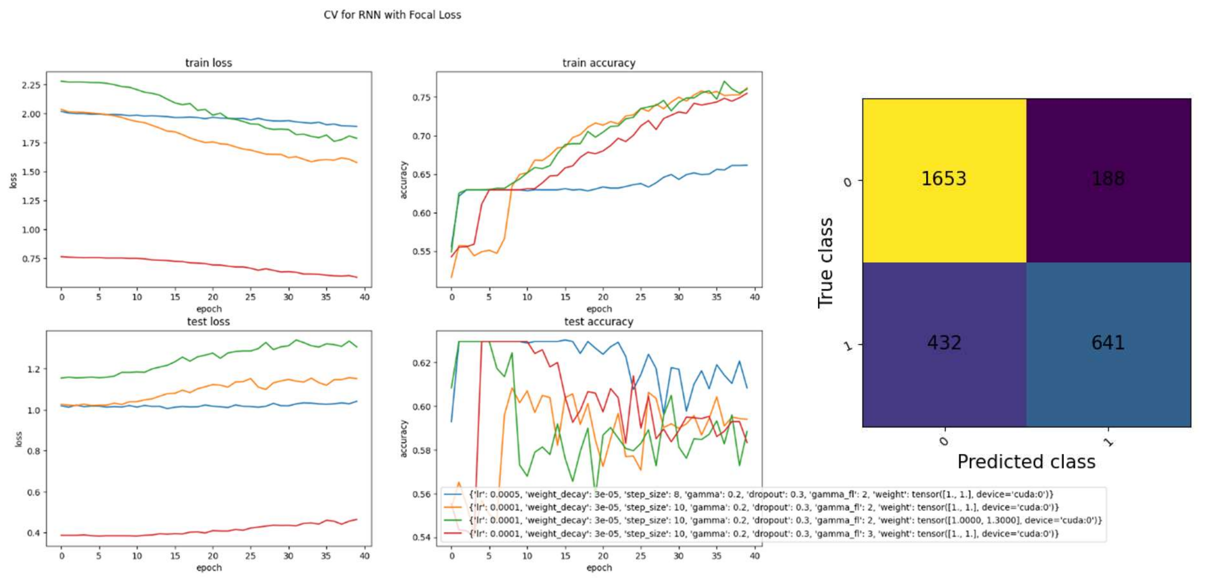


{'lr': 0.0001, 'weight_decay': 3e-05, 'step_size': 10, 'gamma': 0.2, 'dropout': 0.3}
{'lr': 0.0001, 'weight_decay': 4e-05, 'step_size': 10, 'gamma': 0.2, 'dropout': 0.4}
{'lr': 0.0005, 'weight_decay': 6e-05, 'step_size': 8, 'gamma': 0.2, 'dropout': 0.4}
{'lr': 0.0001, 'weight_decay': 3e-05, 'step_size': 8, 'gamma': 0.2, 'dropout': 0.5}

Figure 27-Learning graphs of RNN feature extractor with Cross Entropy Loss and the confusion matrix on the training set of the blue configuration.

With figure 27, we can notice a familiar behavior. While the train loss decreases smoothly and the train accuracy increases, the test loss fluctuates but generally increases. On the other hand, the test accuracy increases dramatically and then fluctuates but stays around the same value of ~61%. In other words, it seems like overfitting because the model improves on the training set but does not improve or deteriorate on the test set (on the accuracy).
From the confusion matrix of the model prediction on the training set, we can see the problem of imbalance where the FN error of ~40% from all viral samples while the FP error is just ~5% from all non-viral. We encounter this problem in the CNN and in the Spotify feature model.

With figure 28, we can notice again the same phenomena we have seen – clear overfitting. In other words, we can notice that when the model train loss decreases the test loss increases. When we try to deal with it by using dropout or weight decay regularization, we just decrease the level of overfitting (the model memorizes less and the train accuracy is lower) but this behavior still presence as we can see from the different curves.
With the confusion matrix on the training set of the model with focal loss, we can notice that more samples are classified as viral, probably because we used focal loss. However, as we can see, there is an increase both in the TP (viral samples that are classified correctly) and FP (non-viral samples that are classified wrongly), which can indicate that the focal loss "force" the model to increase the TP samples at the price of increasing the FP samples. It worth mentioning that this change in the confusion matrix is not due to the weighted mean because in the orange configuration (the confusion matrix's config) all weights are equal.



```
{'lr': 0.0005, 'weight_decay': 3e-05, 'step_size': 8, 'gamma': 0.2, 'dropout': 0.3, 'gamma_fl': 2, 'weight': tensor([1., 1.], device='cuda:0')}
{'lr': 0.0001, 'weight_decay': 3e-05, 'step_size': 10, 'gamma': 0.2, 'dropout': 0.3, 'gamma_fl': 2, 'weight': tensor([1., 1.], device='cuda:0')}
{'lr': 0.0001, 'weight_decay': 3e-05, 'step_size': 10, 'gamma': 0.2, 'dropout': 0.3, 'gamma_fl': 2, 'weight': tensor([1.0000, 1.3000], device='cuda:0')}
{'lr': 0.0001, 'weight_decay': 3e-05, 'step_size': 10, 'gamma': 0.2, 'dropout': 0.3, 'gamma_fl': 3, 'weight': tensor([1., 1.], device='cuda:0')}
```

Figure 28 – Learning graphs of RNN feature extractor with cross entropy loss and confusion matrix on the training set of the orange configuration

From the cross validation, it does not seem there is a clear advantage to use focal loss because, although the TP increased slightly, it was due to general increase in the number of samples the model predicted as viral. As we already know, the ratio of viral and non-viral samples in the training set is ~64:36 and therefore, when we see that the FP increases by 74 samples and the TP increases by 34 samples (almost similar to the labels distribution), it can indicate that the model didn't really learn a better underlying pattern and just increased in general the number of samples it would classify as viral.

As a result, we will use cross entropy loss with the blue configuration and the results are the following:

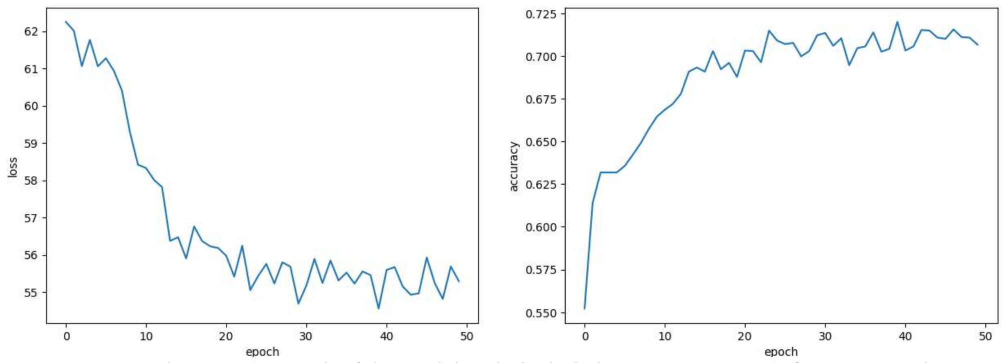


Figure 29-The training graph of the model with the bule hyperparameter configuration and cross entropy loss

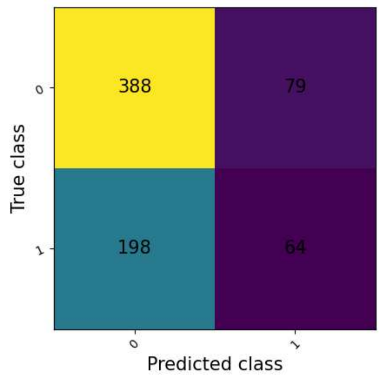


Figure 30-confusion matrix on the test set of the above model

To sum up, the results are:

Train loss	55.292
Train accuracy	0.706
Test loss	15.628
Test accuracy	0.629

As we can see here, the results are still not good, but not because of overfitting but rather due to underfitting caused by the regularization we applied on the training.

Conclusions:

We have approached the problem from 3 different ways: with pre-made Spotify features, with CNN based feature extractor and with RNN based feature extractor. With all 3 we got similar results of an overfitting behavior when we aren't forcing regularization.
Some models succeeded in memorizing the training set but failed in generalizing and some failed in both, due to our attempts to prevent overfitting.

There are few possible reasons for our failure:

1. Our dataset wasn't large enough – it consists of ~3000 samples, which at the beginning of the project seems like decent number of songs. However, due to the complexity of the task and the large models (lots of learning parameters), it is possible that we needed more samples to prevent overfitting and using only dropout and weight decay regularization couldn't handle it by themselves.

2. Our 'non-viral' samples are somewhat viral. We defined non-virality has song with less than $5e5$ videos on TikTok platform. Considering it in retrospect, there is a chance that by taking list of the most viral songs in TikTok and then labeling the more viral as viral and the less viral as non-viral, we tried to distinguish between undistinguishable classes, at least from the audio file. At the beginning of the project, we thought that the more danceable or rhythmic songs will be viral, but as can be seen in figure 31, most of the songs in the

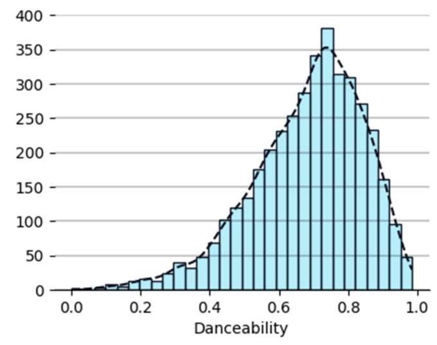


Figure 31-distribution of danceability in the dataset

dataset have high danceability score and this can indicate that the differences between songs are not necessarily related to the audio itself. In other words, there is a chance that in term of audio virality, all the songs we worked with are viral, but there are some that became extremely viral due to 'non-audio' related things, and this brings us to the last problem.

3. The model doesn't have the full picture. There is a chance it is just impossible to predict virality of songs in TikTok from audio – missing essential data like the singer and the political, social or security situation the song has come because and into. In addition, although it is possible to extract the words from the audio, we didn't do so explicitly and tried to understand the meaning of the

song (we did not first extract the words and then feed them into NN but just fed the audio, both the melody and the words, into the NN).

Sadly, we did not fully answer our research question, but this just mean there is more to investigate as follow up. First and foremost, the dataset size should be increased as described in problem #1 and as mentioned in problem #2, we should create a dataset where the non-viral are **really** non-viral songs and not **less** viral.
After doing so, it is also interesting to test a model which get both the audio and the songs words and to see if there is an effect on the scores. It is interesting because it can indicate if TikTok users pay attention to the words in the songs and not just the melody.
In addition, it is possible to test the results with Transformer base models. We tried to use Transformer model in our project and the results did not change dramatically (so we did not add it to the report, but the notebook can be found in the submitted directory[7]), but maybe with the correction described earlier, the transformer strength would be visible.

[7] Although we added the notebook, there is a mistake in the code. We added CLS token at the beginning of the sequence (like in BERT models) to use the output in the CLS position for the classifier. However, we wrongly added the CLS to a different dimension of the tensor (specifically the frequency/Mel scale dimension) and thus the results there are questionable.