Song Genre Classification using ViT-B16

By Nick Rommel and Alex Cerullo

Introduction:

Background:

More and more music is created and uploaded to different services every single day. With the growing number of songs being uploaded, there also rises the need to classify this music into different genres. Manually labeling these music tracks is tedious and time consuming, and is also subject to the inherent biases of the labeler. It is both more efficient and more accurate to train a machine learning model to handle the classification of this music automatically. We aim to build a network that will accomplish this. Audio tracks can be transformed into spectrograms, which can then be fed into image classification networks. Our planned methodology is described in more specific detail in the following section.

We are using the ViT-B16 model, originally created by Dosovitskiy et al. in the Vision Transformer introductory experiment: “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”. This model, alongside other variants, is available to download from pytorch and is fully incorporated into the pytorch module. ViT’s are incredibly data hungry, so the only way we could reasonably use one in our experiment was by employing transfer learning methodologies and downloading a pre-trained ViT. Our modifications to the ViT-B16 will be discussed at length in the Methods section.

Related Work:

It is not a novel idea to handle music genre classification by way of image classification deep-learning techniques on mel-spectrograms of audio tracks. However, many of the existing experiments for this topic use Convolutional Neural Networks as the image classification technique of choice. Hassen et al (Source 5) and Pelchat et al (Source 11) both used Convolutional Neural Networks in their song genre classification experiments. Neither of them used a Vision Transformer for these Computer Vision tasks.

The work done by Niizumi et al is the most similar to our own. In this paper, the researchers introduce Masked Modeling Duo (M2D), which is a new method for implementing Masked Autoencoder self-supervised training methodologies (Source 13). While the focus of their experiment was not music genre classification, they used a “vanilla ViT-Base” as their network of choice. They also did not use a pre-trained ViT, and instead trained it from scratch using their new M2D training methodology. They achieved 83.9% accuracy on the GTZAN dataset.

Methods:

Dataset:

Originally, we were planning on deepak-newzera/spectrogram_data_max_music_dataset-1 from HuggingFace as our primary dataset, as it had sufficient amounts of data for using a Vision Transformer. However, we encountered implementation issues when attempting to use this dataset in our code and could not get it to function properly. Incidentally, we were unable to successfully pull neither models nor datasets from HuggingFace.

The main thing that we needed to do to the chosen audio dataset in order for it to work in our experiment using a ViT was to convert the audio segments into mel-spectrograms so that image classification could be performed. The plan to convert the audio files was to use the python Librosa module, as that module has perfect built in functionality for generating mel-spectrograms from audio files. However, executing these conversion techniques proved to be completely unnecessary thanks to the dataset we ended up using for training our models.

We used a modified version of the GTZAN dataset (described as “the MNIST for music”, Source 1). This version of the GTZAN dataset was found on Kaggle, as it had been curated for a Music Genre Classification competition. The GTZAN dataset contains 1000 audio tracks, with 100 tracks for each of the 10 genres of music present in the data. The Kaggle version of the dataset contained both the GTZAN original .wav files as well as mel-spectrogram images generated for each of the audio files from the original dataset. And because this dataset already had the mel-spectrograms generated for us, there was no need for implementing data transform techniques to convert the files to a usable format ourselves. This dataset can be found at the link listed in Source 10, in the “Sources” section.

Transfer Learning:

In order to properly implement the ViT_B_16 model, we accessed PyTorch’s pre-trained model database and downloaded the default weights and biases of the model. This default model was pre-trained on the ImageNet-1K, with images resized to 256x256 pixels, and then cropped to 224x224 this way, the network was built to be size agnostic, as every training image underwent some transformations. In addition to resizing, the data was normalized using a mean of [0.485, 0.456, 0.406] and a standard deviation of [0.229, 0.224, 0.225].

Set up like this, the ViT was well suited for identifying features in a wide range of images, but it still needed to be fine tuned for this specific task. To accomplish this we reinitialized the ViT head, and created a 3 layer MLP with 10 outputs. Our first few trials involved training the entire ViT body and MLP output, however the training time was roughly 8 seconds per minibatch of 100 images. In order to optimize training time, we froze the training gradients of the ViT body, and trained only the MLP itself. This resulted in a minibatch training time of 0.24 seconds.

We had intended to utilize the technique described by Gradient Descent The Ultimate Optimizer [14] to fine tune the training weights of the network, however we were unable to use this system to train hyper-parameters while simultaneously freezing the parameters in the ViT body. As we were once again training the entire ViT, the training time increased massively, creating the same dilemma as before of balancing computation time vs effectiveness.

Experiments and Analysis:

All model training runs were conducted using the cuda python module and pytorch on an NVIDIA RTX 3080ti GPU.

Hyperparameter Sweep:

Our initial sweep of hyper-parameters observed Learning Rates from 0.001-0.01 with step increments of 0.0002. This resulted in a total of 46 separate Learning Rates being tested. In addition, we used Weight Decays of 0.005-0.015 with increments of 0.01, for a total of 11 weight decays. This resulted in a total of 506 different Hyperparameter models trained. We ran this sweep for a total of 25 epochs per network, and each epoch we performed a validation check, and then if the Validation Accuracy was higher than the previously recorded best, we copied the weights and biases of the network to a temporary holding network using the built in state_dict() function in Pytorch. This allowed us to take a snapshot of best performance from the entire range of hyper parameters.

After analyzing the results from our first sweep we noted that the best performance occurred with Learning Rates from 0.001-0.005, and these models had their best results after 5-15 epochs, while the loss of the networks began to increase after those 5-15 epochs. As a result we limited our secondary sweep to include Learning Rates from 0.001-0.005 with step increments of 0.002, Weight Decays of 0.005-0.015 with increments of 0.01, and 15 total epochs. The second difference in this sweep was the manner of choosing the best network. While previously we had found the network with the highest validation accuracy, here we saved the network with the lowest loss. We made this decision because during a qualitative analysis of our run data, we noticed that as the loss of a function increases, it becomes less able to generalize its predictions. While it may have a low error for things that it has seen or been trained on, a network with high loss will have high error on a novel dataset. Meanwhile, a network with lower loss will be more versatile when confronted with new images.

The training error and loss were recorded for each minibatch, and the validation error and loss were recorded for each epoch. After all 25 epochs, this data, and the time required to run the model were saved to a .txt file with the hyper-parameters of the model acting as a name for the results.

Data Pre-Processing:

Much of the necessary data preprocessing was already taken care of thanks to the standard data curation methodologies of Kaggle. Due to that, the audio files were already in mel-spectrogram form, which made them ready to be fed into the encoder of the Vision Transformer. The following are the data transformations conducted on the images before they were loaded by the DataLoader python module.

Many of the transforms we applied to the data were following instructions and recommendations in the PyTorch documentation for using the Vit_B_16 model. We first center-cropped the images in order to remove much of the white space that surrounds them, and then we resize them to 224x224 squares. We then normalized the images using the recommended values from the pytorch documentation.

We employ the random_split() Pytorch function to split the dataset into Train (80%), Validation (10%), and Test (10%) sets. These splits are then fed into 3 different calls to DataLoader with the aforementioned transforms applied. The characteristic “flattening” of the images into patches is done by the ViT in its module.

Results:

Our initial hyperparameter sweep brought a few interesting conclusions to light. Namely, Weight Decay had a minimal and almost unnoticed effect on the performance of the network overall. Figure 1 displays the average validation error for each hyperparameter. As you can see from Figure 1a, with different Learning rates, the average Validation error ranges from ~55% Error to ~35% Error. This is contrasted with the Weight Decays in Figure 1b, regardless of the Weight Decay, the average Validation Error is ~40%. This shows that the choice of Learning Rate is much more important than the choice of the Weight Decay.

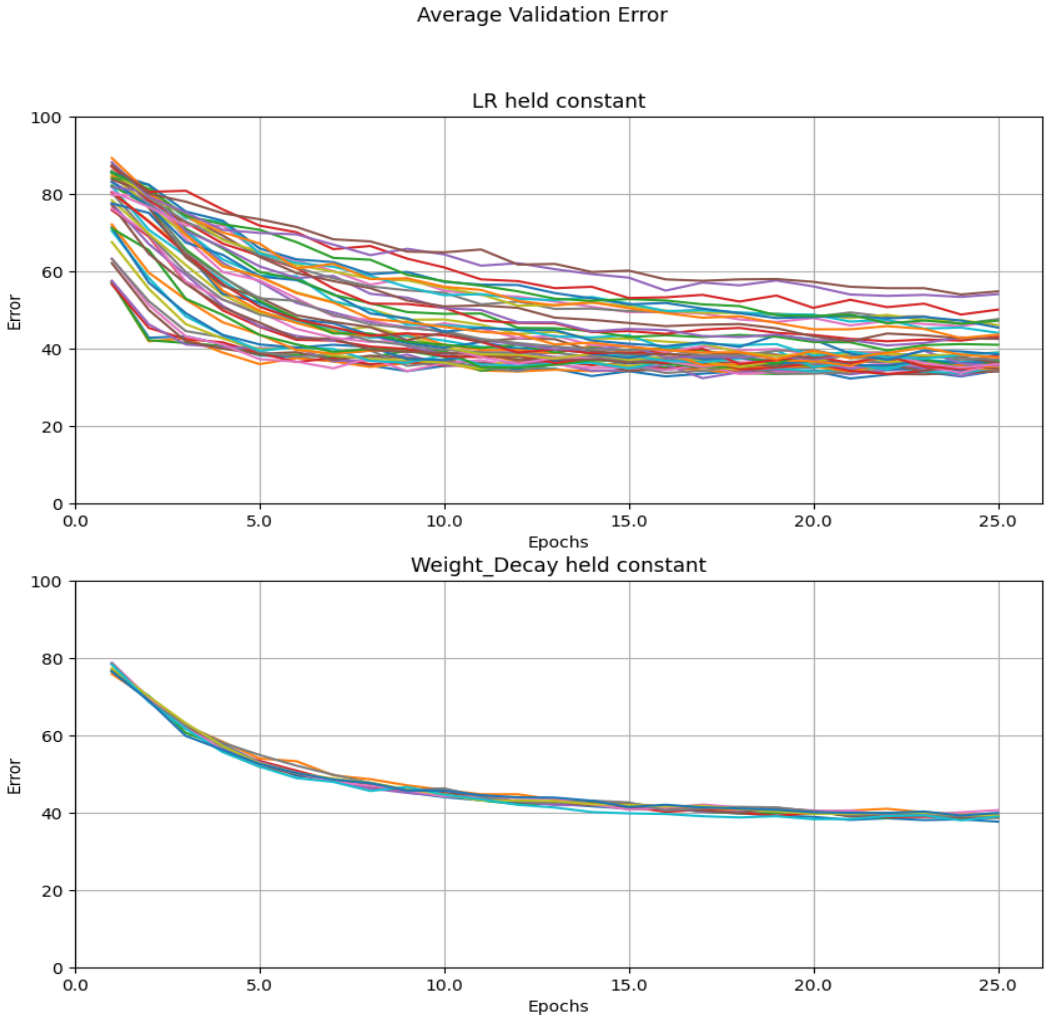

Figure 1

(Figure 1a: Top each line shows the error of a single Learning Rate while averaging across all 11 Weight Decays. Figure 1b: Bottom each line shows the error of a single Weight Decay while averaging across all 46 Learning Rates.)

After the recognition that Learning Rate has the primary effect on model performance we moved to looking at the effects specific learning rates had on the Loss and Error of the networks. Figure 2 displays the average validation error from Figure 1, but spread out over 9 subplots and separated to 5 Learning Rates per subplot. These graphs show that at high Learning Rates (LR:0.008-0.0098) the accuracy is generally 40-50%% or higher. Additionally, we note that learning rates of 0.001-0.003 have a validation error of around 35%.

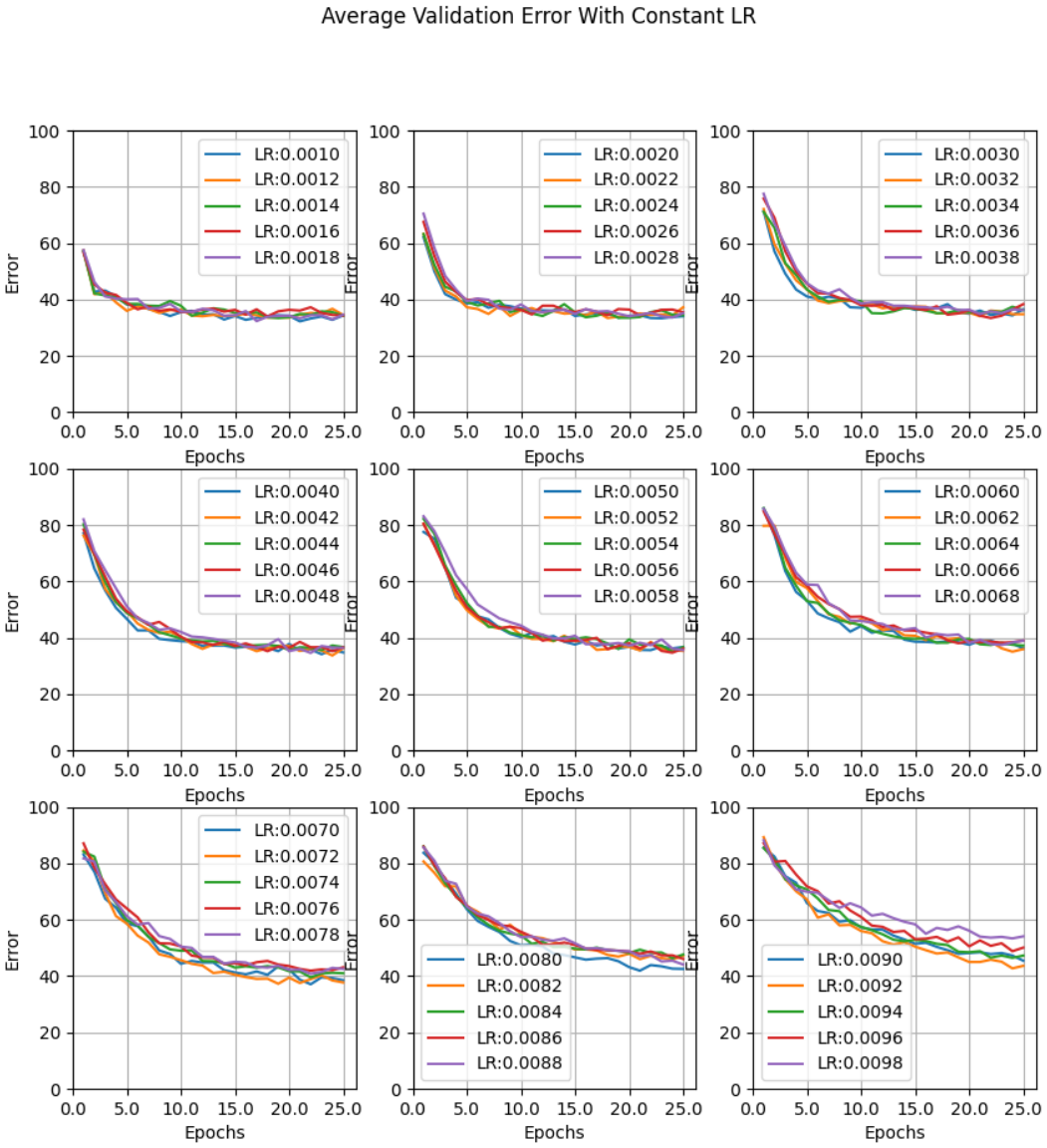

Figure 2

Figure 3 displays the average validation loss instead of error. In contrast to Figure 2, where the error begins to plateau after a series of epochs, this figure shows a ‘boomerang’ effect. After 5-10 epochs for learning rates 0.001-0.004. This shows that while Error does not change with further training, Loss reaches a minimum after a few epochs, and further training will lessen the model’s ability to generalize. We believe this is partially due to the limited size of our dataset (only 1000 samples). Despite this, the loss of the low learning rate networks does lower to around 1.125 before beginning to increase, which is in contrast to high learning rate networks where the minimum loss is 1.25 or above, and does not have as pronounced a ‘boomerang’.

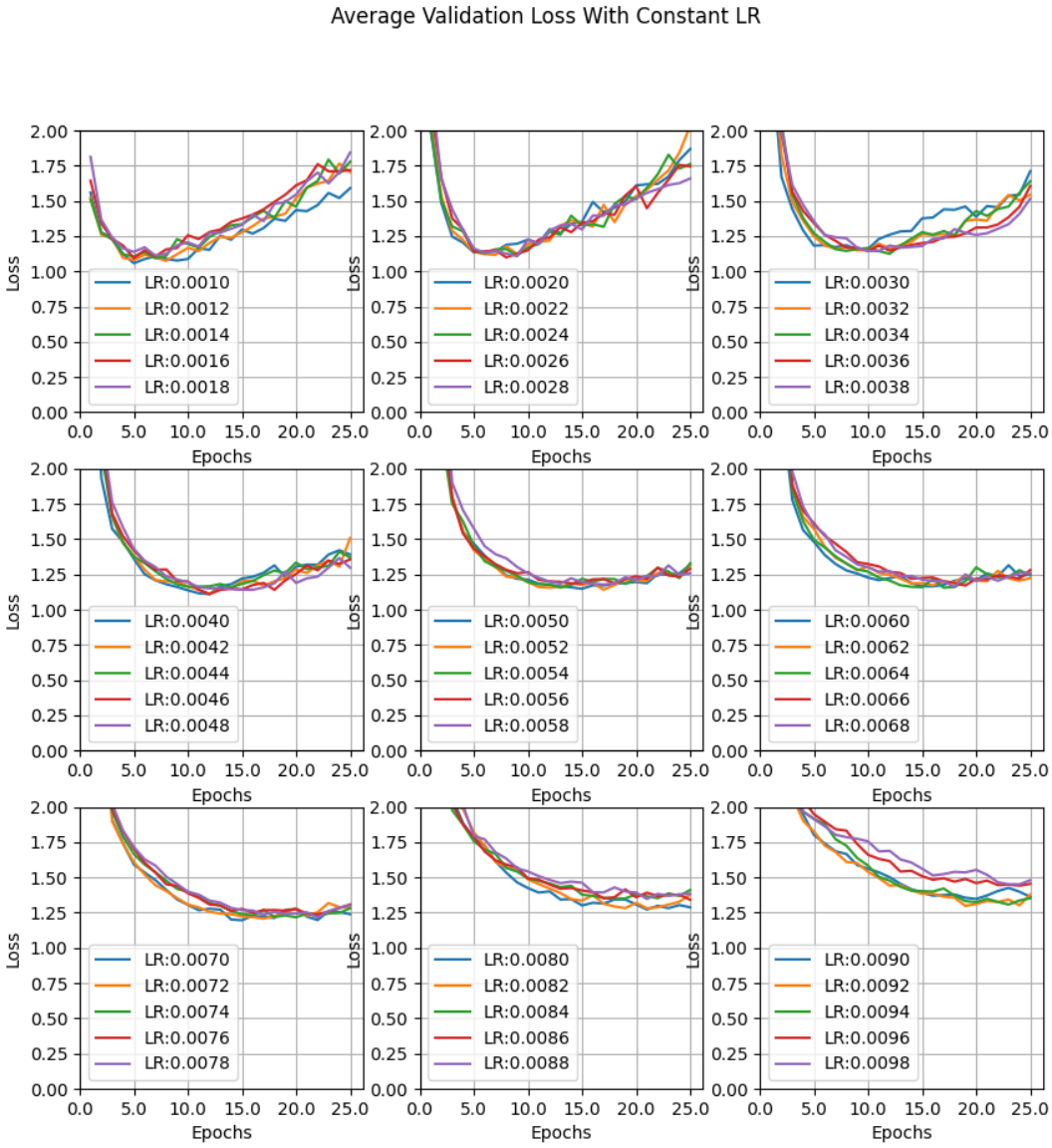

Figure 3

Figure 3: The top row shows that low learning rate has a ‘boomerang’ effect on the average loss of the model. They reach the minimum loss early, and further training will increase this loss

During the first sweep while we were saving the network with the best validation error we achieved a Final Testing Error of 40% and a Final Testing Loss of 1.86 using a Learning Rate of 0.0026, and a Weight Decay of 0.005. In contrast, when saving the network with the best validation loss, we achieved a Final Testing Error of 35% and a Final Testing Loss of 1.137 using a Learning Rate of 0.0016 and a Weight Decay of 0.012.

Conclusion:

The goal of our experiment was to successfully employ transfer learning on a baseline ViT model, that was pretrained on the ImageNet data set. Following the conclusion of our second hyperparameter sweep, we achieved a Test accuracy of 65.66% on classifying the 10 music genres in the GTZAN dataset. State of the art (SOTA) performance is just over 83% test accuracy (Source 13). However, that SOTA performance was employing a methodology that trained from scratch on the target dataset in a self-supervised fashion. We transferred the ImageNet-trained ViT and fine-tuned it on a dataset consisting of mel-spectrograms, which are very unlike most other images, especially the images of the ImageNet dataset and their respective features. Therefore, the discrepancy between these two accuracies can be more easily explained. While our results are not currently ready for commercial or analytical use, the fact that the models were fine tuned on equipment available to the average person reflects that more dedicated personnel and equipment can achieve greater accuracies, especially when used with larger datasets.

Limitations and Ideas for Future Work:

One of the limitations we faced was the lack of available data. Thankfully, the modified GTZAN dataset we found served our purposes in training our different model combinations. But the training data only had access to 800 images of the total 1000 that make up the entire dataset. This is not a lot of data, especially where a ViT is concerned. For future experiments, endeavoring to successfully download and access larger audio datasets for use in model training would facilitate better results. Failing finding such a dataset, we could figure out how to use the YT-DLP audio extraction tool (https://github.com/yt-dlp/yt-dlp) to generate our own dataset by stripping audio from youtube videos.

We were also limited by our available compute. While we did have access to a GPU for training, it was only a single unit. Certainly not a GPU cluster like those available to many research facilities. If we had more compute available, as well as sufficient data (as mentioned in the previous paragraph), training a ViT from scratch would have been much more feasible. Training the ViT from scratch might have led to better performance and accuracy.

Another methodology to employ in a future experiment would be to use the M2D of Niizumi et al (Source 13), or some other Masked Autoencoder method to institute self-supervised training on the target dataset. This would allow the ViT to learn the important features of the spectrograms in a self-supervised fashion.

Another avenue to explore in future research is how the distribution of the data in the data loaders affects the final test accuracy. As mentioned previously in the report, we used the Pytorch random_split() function to split our data into three sets. It would be interesting to see how using a different function or a different library's module (train_test_split() from sklearn, for example) could affect the accuracy. Additionally, we could experimentally determine if it is better

to randomize the test set with different data each sweep, or see if permanently reducing the dataset to create a static, hold-out test set would improve accuracy.

Links and Repo:

GitHub Repo:
https://github.com/nick-rommel/CSC561-Final-Project/

Weights and Biases Project page:
https://wandb.ai/foxx-skulk/CSC561-Final-Project/overview

Youtube Presentation Link:
https://www.youtube.com/watch?v=MTV8496kwUE&ab_channel=AlexCerullo

Sources:

1. Won, M., Spijkervet, J., & Choi, K. (n.d.). *Music classification: Beyond supervised learning, towards real-world applications*. Datasets - Music Classification: Beyond Supervised Learning, Towards Real-world Applications. Retrieved April 12, 2023, from https://music-classification.github.io/tutorial/part2_basics/dataset.html
2. Hermez, Celestin. “Music Genre Classification with Tensorflow.” *Medium*, Towards Data Science, 11 Aug. 2020, https://towardsdatascience.com/music-genre-classification-with-tensorflow-3de38f0d4dbb.
3. Connelly, E. (2017, April 26). *How to build a simple song recommender system*. Medium. Retrieved April 12, 2023, from https://towardsdatascience.com/how-to-build-a-simple-song-recommender-296fcbc8c85
4. Won, M., Spijkervet, J., & Choi, K. (n.d.). *Music classification: Beyond supervised learning, towards real-world applications*. Datasets - Music Classification: Beyond Supervised Learning, Towards Real-world Applications. Retrieved April 12, 2023, from https://music-classification.github.io/tutorial/part2_basics/dataset.html
5. “Classifying Music Genres Using Image Classification Neural Networks”, Alan Kai Hassen, (Et al).
6. https://huggingface.co/datasets/deepak-newzera/spectrogram_data_max_music_dataset-1
7. https://huggingface.co/m3hrdadfi/wav2vec2-base-100k-gtzan-music-genres/tree/main
8. K. M. Hasib, A. Tanzim, J. Shin, K. O. Faruk, J. A. Mahmud and M. F. Mridha, "BMNet-5: A Novel Approach of Neural Network to Classify the Genre of Bengali Music Based on Audio Features," in *IEEE Access*, vol. 10, pp. 108545-108563, 2022, doi: 10.1109/ACCESS.2022.3213818.
9. A. K. Sharma *et al*., "Classification of Indian Classical Music With Time-Series Matching Deep Learning Approach," in *IEEE Access*, vol. 9, pp. 102041-102052, 2021, doi: 10.1109/ACCESS.2021.3093911.
10. GTZAN Dataset, https://www.kaggle.com/datasets/andradaolteanu/gtzan-dataset-music-genre-classification
11. Pelchat, N., & Gelowitz, C. M. (2020). Neural Network Music Genre Classification. *Canadian Journal of Electrical and Computer Engineering*, *43*(3), 170–173. https://doi.org/10.1109/cjece.2020.2970144
12. A. Dosovitskiy et al., “An Image is Worth 16x16 Words: Transformers for Image REcognition at Scale”.
13. Niizumi, D., Takeuchi, D., Ohishi, Y., Harada, N., & Kashino, K. (2023). Masked modeling duo: Learning representations by encouraging both networks to model the input. *ICASSP 2023 - 2023 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. https://doi.org/10.1109/icassp49357.2023.10097236
14. Chandra, K., Xie, A., Ragan-Kelley, J., & Meijer, E. (n.d.). Gradient Descent: The Ultimate Optimizer. *Cornell University*.