

# **Multimedia Content Analysis**

## *Homework 4 – Music Genre Classification*

**Student ID:** P76124786

**Student Name:** 徐向廷

**Department:** CSIE (1<sup>st</sup> Year Graduate)

### **Problem Statement**

In this project, our objective is to analyze audio data and classify these segments into their respective music genres. We will explore various deep learning methodologies, architectures, and features to address this task. Specifically, we will focus on *supervised learning*, adjusting the weights and biases of a neural network through algorithms like *backpropagation* and *gradient descent*. This guarantees that the output predictions of our model resemble the ground truth provided. Nevertheless, *deep learning* approaches have their limitations, which we will discuss in later sections.

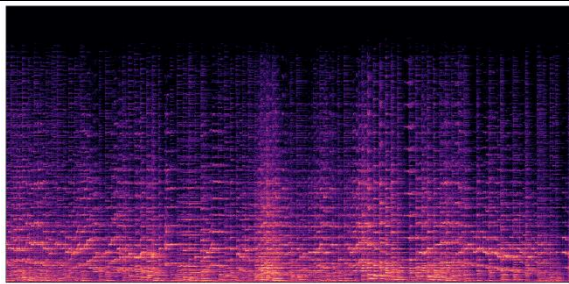
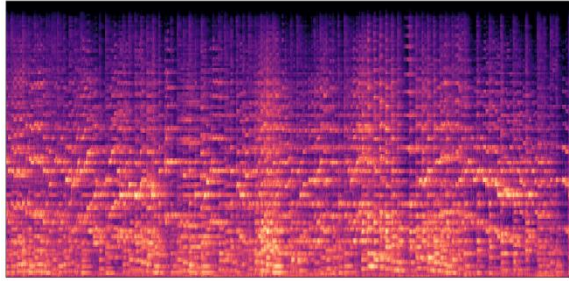
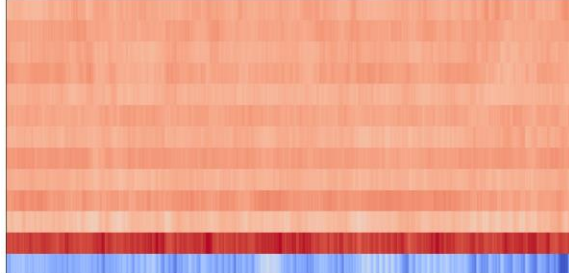
### **Feature Extraction & Preprocessing**

In training deep learning models, it's essential to *preprocess* our data to a format that can be fed into the model, or even transform the data to make it rich in even more meaningful information. In the case of audio data, we aim to convert it from the *time domain* to the *spectral domain*, because *frequency* is crucial for distinguishing *pitch* and other features relevant for music genre classification. To accomplish this, we employ the *Short-Time Fourier Transform*, the formula of which is listed below.

$$X[m, k] = \sum_{n=0}^{N-1} x[n] * w[n - m] * e^{-j2\pi nk/N}$$

The ***Short-Time Fourier Transform*** is used to map audio data from the time domain to the ***frequency domain***, providing us with a ***spectrogram***. However, spectrograms often have a ***high number of dimensions***, leading to ***sparse*** information relevant to the music; and according to ***the curse of dimensionality***, data points become more uniformly spaced apart in these higher dimensions, which necessitates ***exponentially*** more data for the classifier to effectively model the data distribution.

Since music is a human creation and the concept of genres is a human construct, it's beneficial to convert the information to a scale that better aligns with ***human perception***. To achieve this, we employ the Mel-scale ***filter bank*** to convert the spectrogram to the ***Mel scale***, resulting in the ***Mel-Cepstrum***. Additionally, we apply logarithm and utilize the Inverse ***Discrete Cosine Transformation*** to extract these Mel-Frequency Cepstral Coefficients (***MFCCs***). The following figures illustrate the differences among spectrogram, Mel-Cepstrum, and MFCC, where we use the same audio segment (***classical.00000.wav***).

	<p><b>Figure 1.</b> Spectrogram</p>
	<p><b>Figure 2.</b> Mel-Cepstrum</p>
	<p><b>Figure 3.</b> MFCCs</p>

While the spectrogram and the Mel-Cepstrum may appear somewhat similar, they differ significantly in their *number of dimensions*. For instance, the *spectrogram* of the 30-second audio has a size of **(1025, 1293)**, whereas the *Mel-Cepstrum* has only **(128, 1293)**, as we have chosen a relatively large value for the “*n\_mels*” parameter.

In this experiment, we executed the preprocessing scripts using *Python 3.10* on *Google Colab*. The libraries employed include *Librosa* for audio processing, *NumPy* for matrix operations, and *TQDM* for progress bar visualization.

## Experimentation with Recurrent Neural Networks

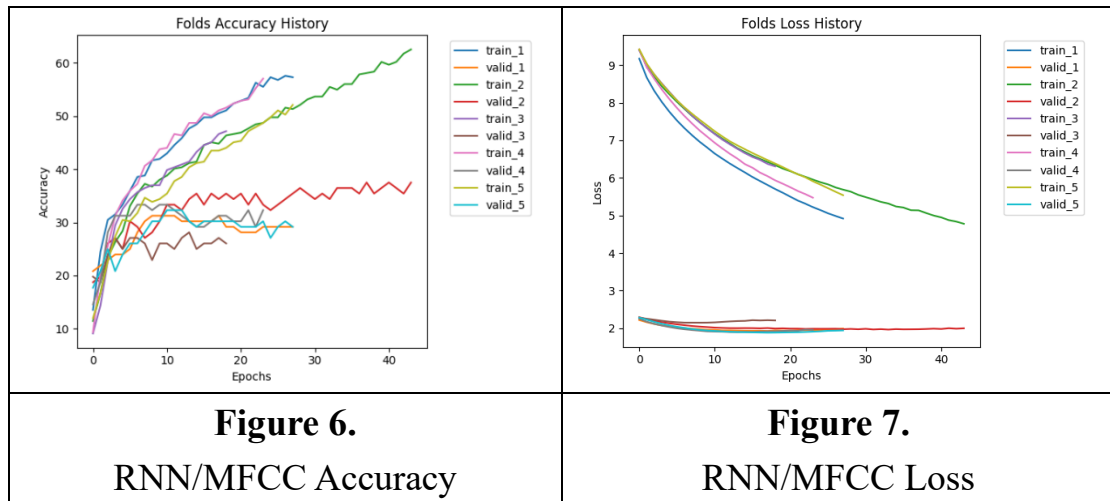
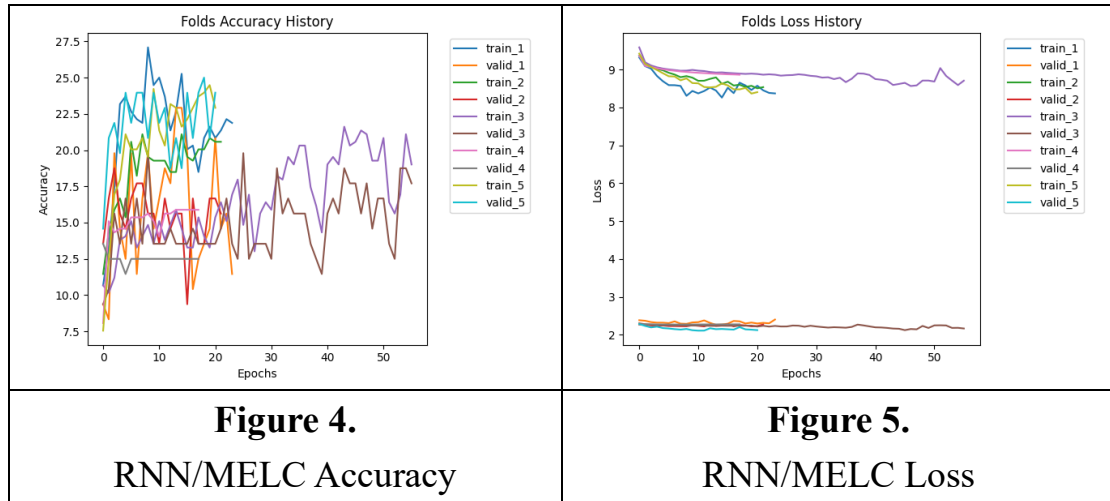
When classifying audio data, which is essentially a *time series* problem, it's intuitive to adopt *recurrent networks*. Unlike *fully-connected layers*, recurrent networks feed back the output values of the *previous timestep* to the same layer at the current timestep, or specifically, to the same neuron that produced the output. Recurrent cells typically use the tanh *activation function* instead of ones like *ReLU* or *sigmoid*.

In this experiment, we are also training on *Google Colab*, making use of *V100* and *A100* GPUs. The deep learning framework used is *TensorFlow Keras*. We will, at times, refer to Mel-Cepstrum as **MELC** for simplicity.

Initially, we will employ a single layer of *RNN cell* followed by a *linear layer* with a Softmax activation function at the end to get a *probability distribution* for all classes. The *Softmax* formula is shown below.

$$P(x_i) = \frac{e^{x_i}}{\sum_{j=1}^k e^{x_j}}$$

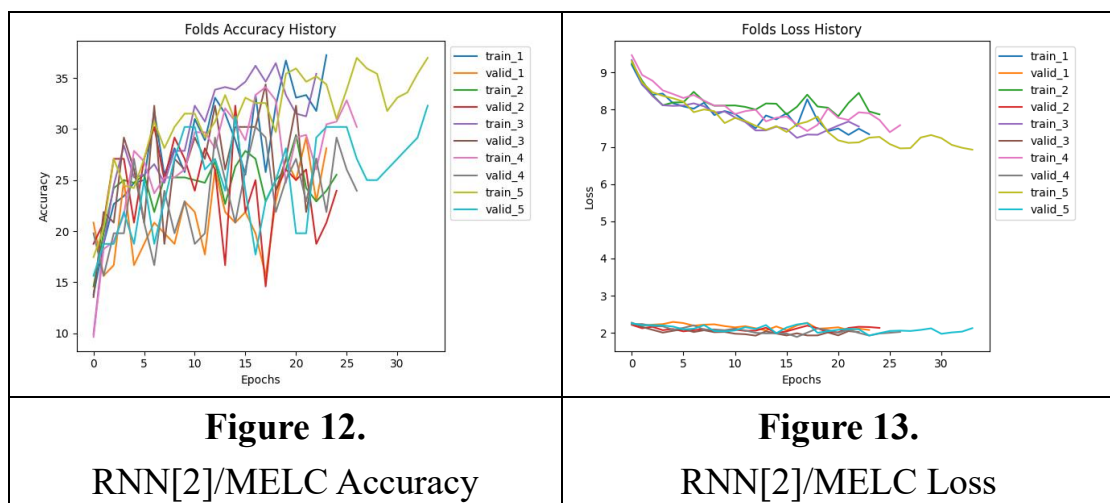
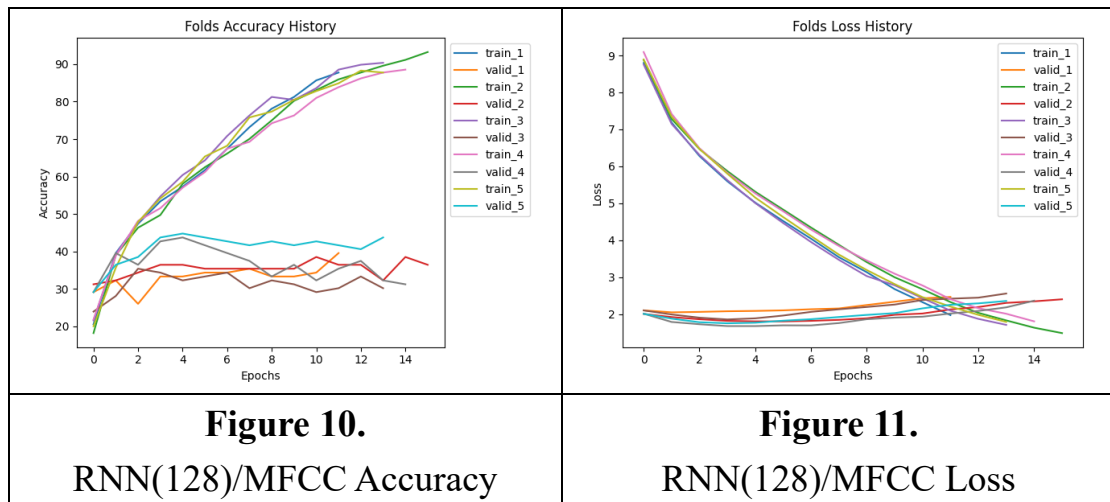
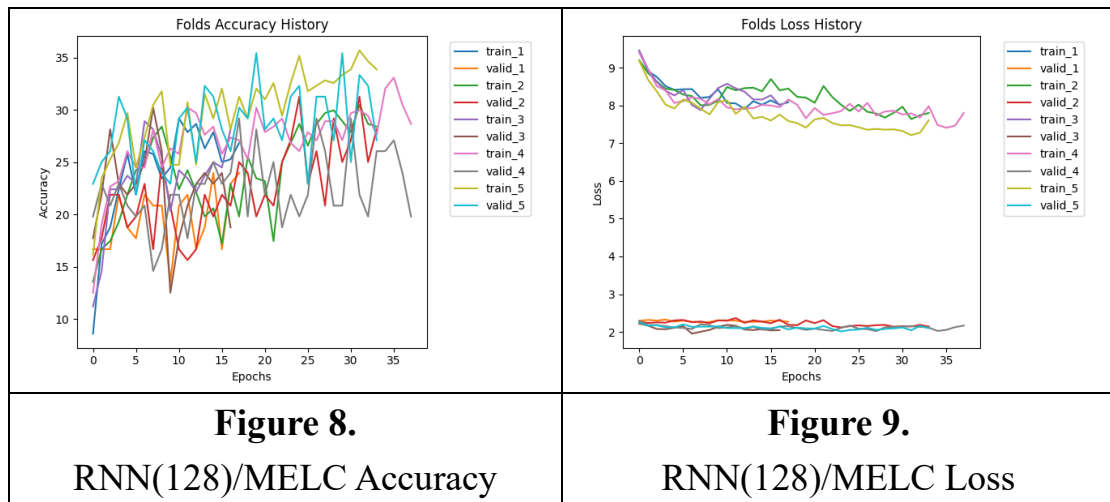
To combat *overfitting*, we monitor the *validation loss* during training and terminate the training process when the loss stays stagnant for **10** epochs. This approach is known as the *early stopping* mechanism. Similarly, we typically overwrite the previous *checkpoints* only if the current model achieves the best *validation accuracy*.

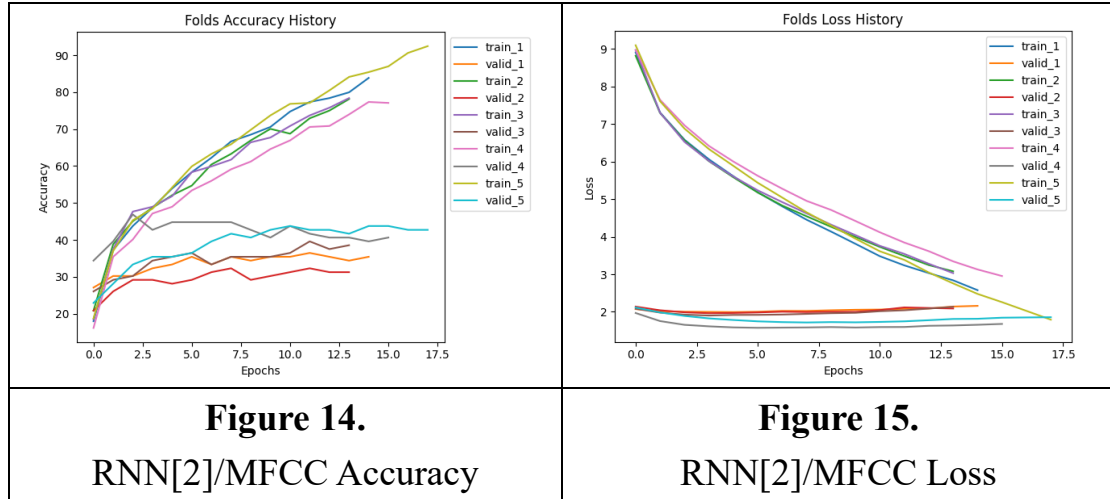


The figures above illustrate the accuracy and loss for both training and validation data across all folds in the **5-fold cross-validation** framework. It is evident that MFCC outperforms MELC, with a validation accuracy of **32.50%** for MFCC and only **20.00%** for MELC. Additionally, it should be noted that the number of *timesteps* is **1300**, so the data are *transposed* and then *padded* or *truncated*.

The figures suggest that the model struggles to learn patterns from the MELC data, likely because it has significantly more *dimensions* than there are data points for each class, which is **50**.

So far, we have only experimented with a single **RNN** layer containing **32** neurons, and we cannot help but consider the possibility that increasing the *number of layers* or *neurons per layer* may improve performance.





The figures above illustrate that by adding more *neurons* or *layers*, we can effectively improve the validation accuracy of our RNN model. When we increased the number of RNN neurons from 32 to **128**, the validation accuracy improved to **30.00%** (+10%) for MELC and **40.42%** (+7.72%) for MFCC. Furthermore, utilizing *two* RNN layers with **64** neurons each, the validation accuracy increased to **31.67%** (+11.67%) for MELC and **39.79%** (+7.29%) for MFCC.

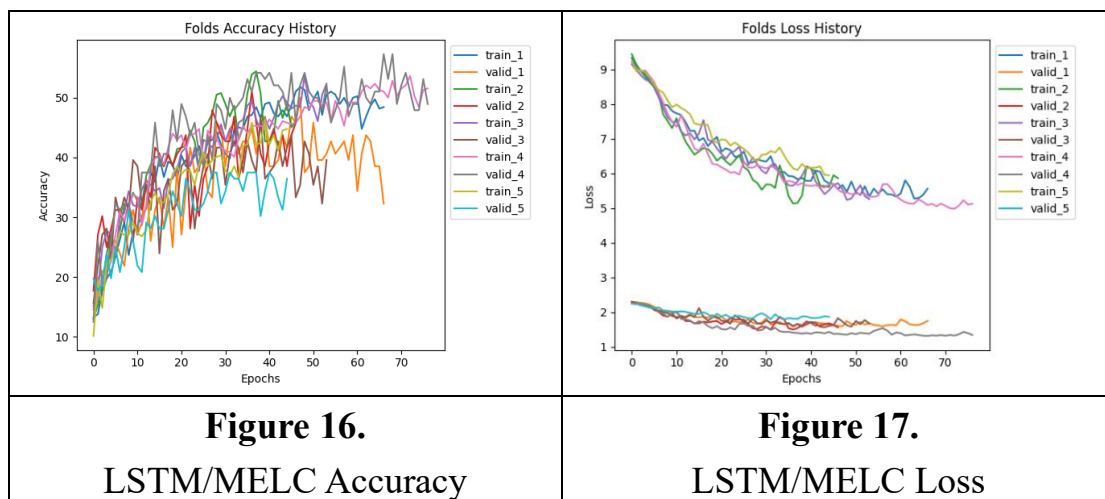
The validation loss is often lower than the training loss because we are using the *Categorical Cross-Entropy Loss* (CCE), which sums up the losses among each class without averaging the values by the *batch size*. Since there are *fewer* validation data points compared to training data, the validation loss tends to be lower. To obtain a fair comparison, we could multiply the validation loss by a factor of **4**, since we are using *5-fold cross-validation* here.

$$CCE = - \sum_{i=1}^N \sum_{j=1}^C t_{i,j} * \log(p_{i,j})$$

Increasing the number of neurons or layers does not seem to improve the *fluctuation* of the accuracy or loss for the *MELC* data during training. This suggests that *RNNs* may not be the best choice for this task, as they have a fatal flaw in their capacity to *retain memory* over a long period of time. With **1300** timesteps in our data, RNNs could be *unsuitable* for this task. We will explore other recurrent models to address this issue.

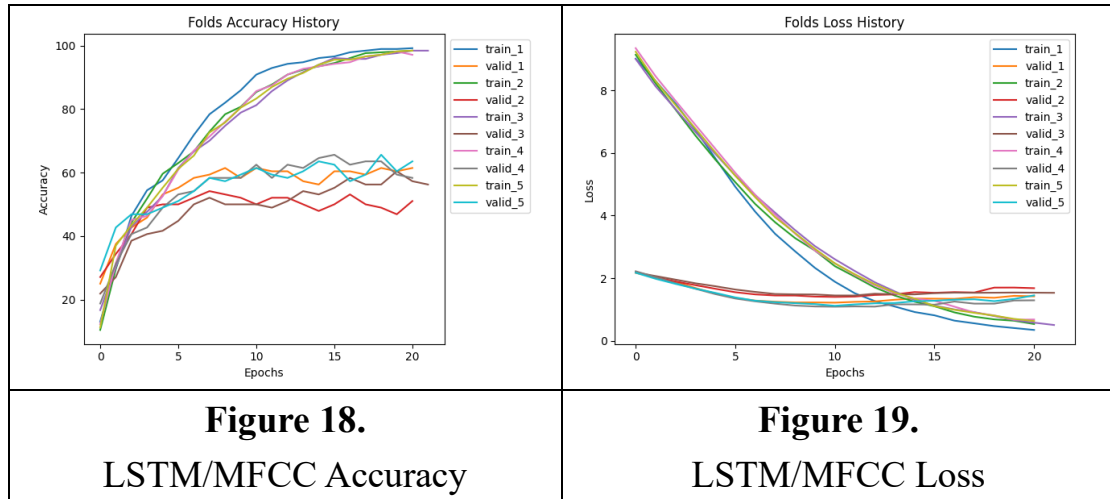
The **LSTM** component introduces the concept of **long-term** and **short-term memory** to recurrent networks, where the former is represented by the **cell states**, the short-term memory by the **hidden states** in each cell. Within the LSTM component, there are three distinct gates that control how information is retained. The **forget gate** utilizes a sigmoid function to adjust the long-term memory by a factor between 0 and 1. The input gate determines which and how much short-term memory gets passed to the long-term memory. It uses a **sigmoid function** for the amount (**0~1**) and a **tanh function** for the information (**-1~1**). Lastly, the **output gate** controls the flow of information from the cell state to the hidden state.

For the LSTM model, we will utilize the same **Adam optimizer** with a **learning rate** of **0.001**. We'll employ a single **LSTM cell** with **64** neurons, as we have observed from our experiments with prior RNN models that **32** neurons are insufficient. To leverage **memory efficiency**, we select the next number in the **powers of 2 series**, which is **64**. The following figures depict the training results of this model.



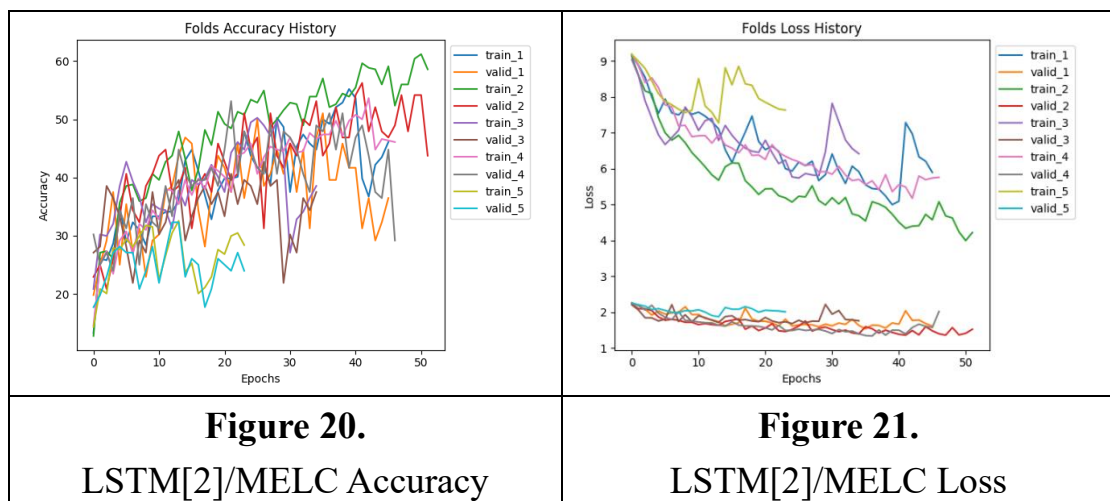
From the figures above, we observed a significant improvement in the **validation accuracy** after changing the model architecture from RNN to LSTM. This model achieved a validation accuracy of **48.54%** even on the **MELC** data, which is a **+28.54%** improvement from the 32-neuron RNN model and a **+18.54%** improvement from the 128-neuron variation. On top of that, the **fluctuations** appear to be more **controlled**.



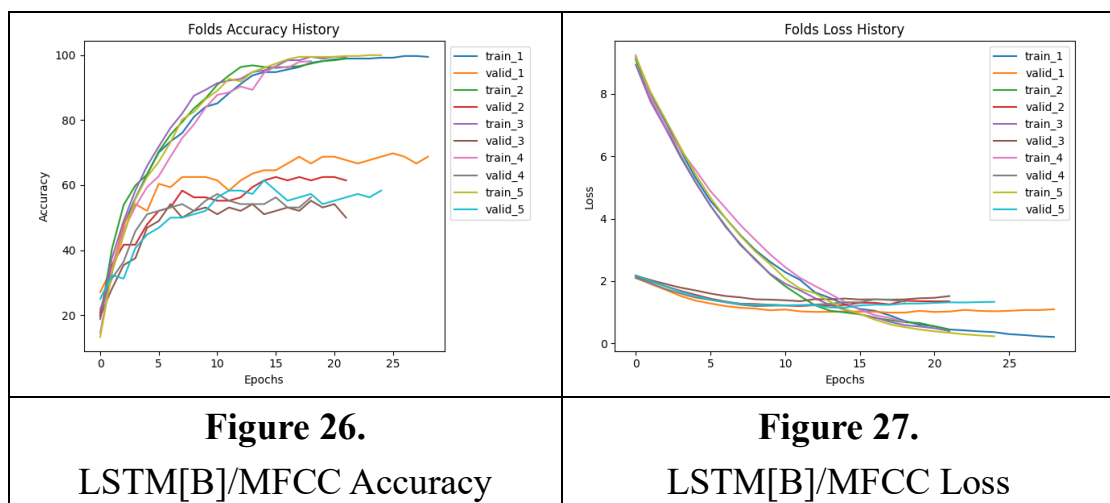
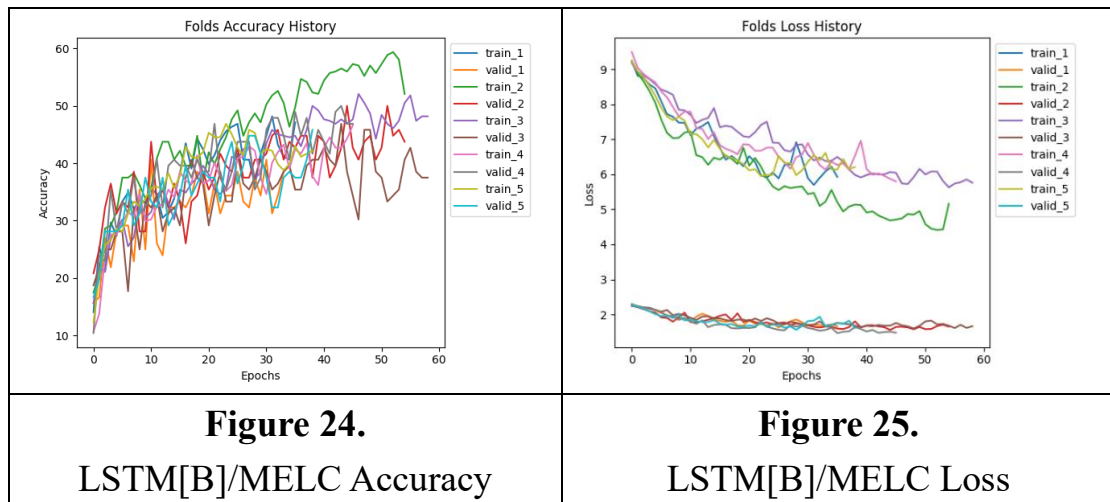
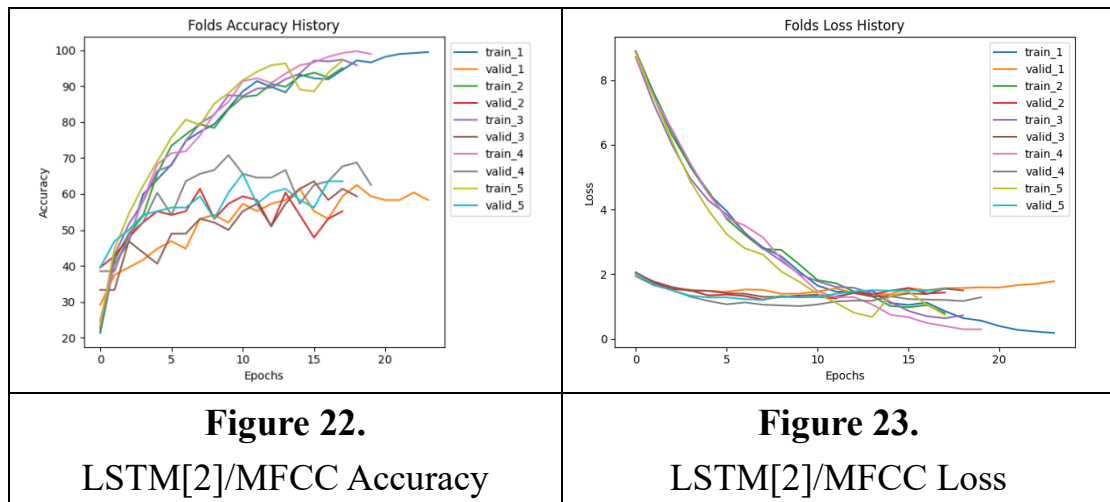


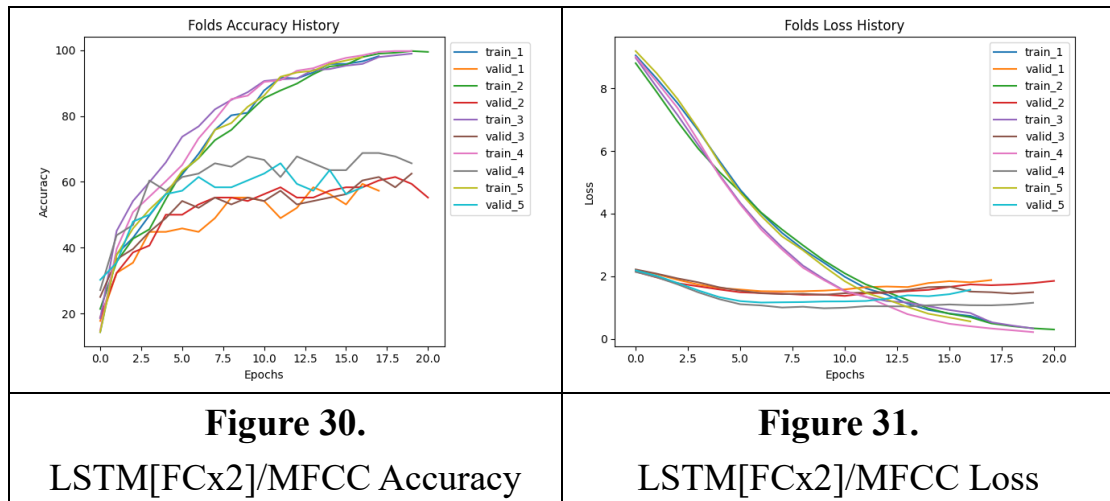
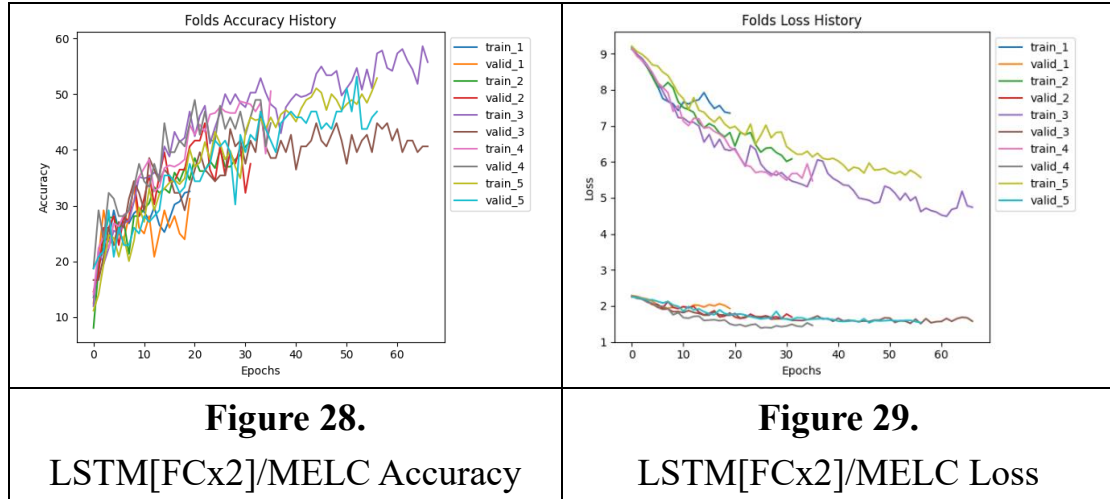
On the other hand, the performance of this model on the *MFCC* data has increased significantly, reaching a **training accuracy** of **98.54%**, which is a new record. For the **validation accuracy**, it averages at **61.46%** across the 5 folds, which is a whopping **+21.04%** improvement from the **128-neuron** RNN and a **+28.96%** improvement from the vanilla **32-neuron** RNN model.

Similar to how regular *RNN models* received significant performance improvement by adding neurons or layers, we wonder if the same would happen for *LSTMs*. Next, we will explore the effects on performance when adding more LSTM layers or fully connected *classification layers*. Moreover, we will observe the effects of making the LSTM *bidirectional*, enabling it to learn from left to right and from right to left. The following figures depict the experimentation results.







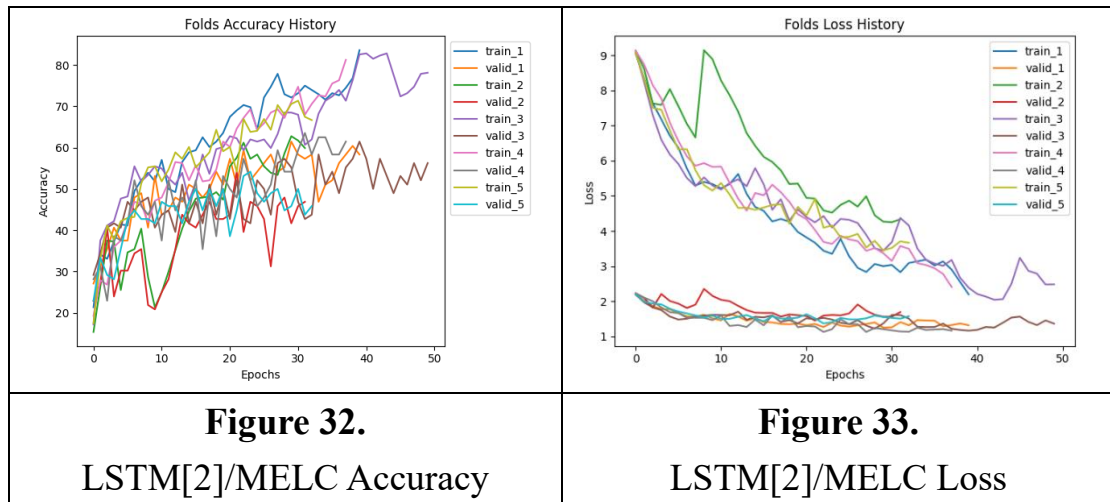


The figures above show that adding more LSTM or linear layers does indeed contribute to better model performance. Using the LSTM with only one layer as baseline, the validation accuracy of 2-LSTM is **64.79%** (+3.33%) for MFCC and **47.08%** (-1.46%) for MELC. For Bi-LSTM, the validation accuracy is **61.25%** (-0.21%) for MFCC and **43.67%** (-4.87%) for MELC. Additionally, the validation accuracy of LSTM with 2 linear layers is **63.54%** (+2.08%) for MFCC and **45.42%** (-3.12%) for MELC.

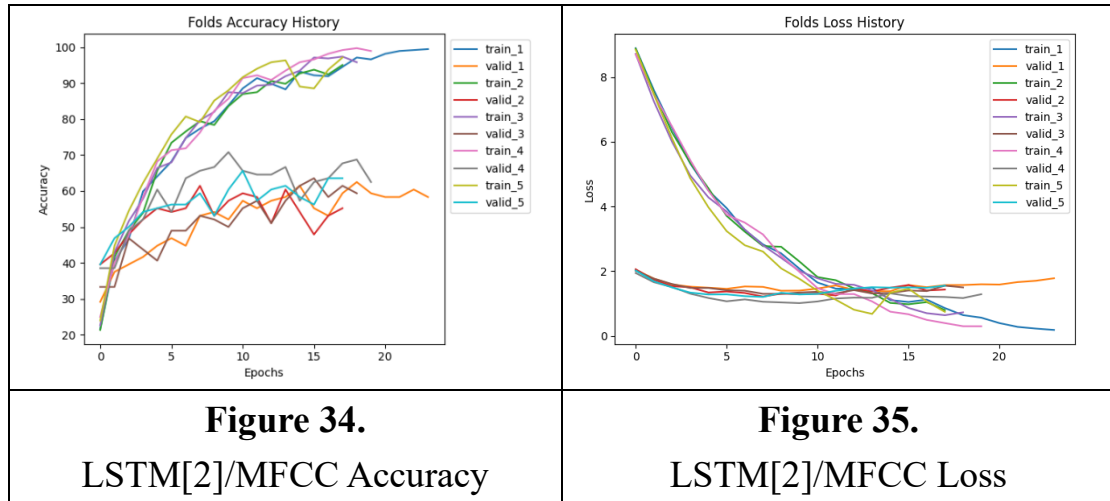
The *worst* one of the three is the *bidirectional* LSTM, as it undermines the validation accuracy of both *MELC* and *MFCC* data. This is likely due to *overfitting*, as its training accuracy is the best among all three methods for both data. This model tends to focus on how the *vector* changes across both directions in time, which may not be as important for music genre classification but could contribute to *memorization*.

<b>Table 1. Recurrent Network Performance Juxtaposition (VA)</b>		
<b>Model Type</b>	<b>MELC Data [128]</b>	<b>MFCC Data [13]</b>
RNN(32)+1FC	20.00%	32.50%
RNN(128)+1FC	30.00%	40.42%
RNN(64)[2]+1FC	31.67%	39.79%
LSTM(64)+1FC	<b>48.54% (+28.54%)</b>	61.46%
LSTM(64)[2]+1FC	47.08%	<b>64.79% (+32.29%)</b>
Bi-LSTM(64)+1FC	43.67%	61.25%
LSTM(64)+2FC	45.42%	63.54%

So far, we have only ventured out using data preprocessed using a fixed set of parameters ( $n\_mels = 128$ ,  $n\_mfcc = 13$ ). However, there lies the possibility that these numbers are not optimal for extracting important features from the music audio files. Here, we will set  $n\_mfcc$  from **13** to **20** and  $n\_mels$  from **128** to **40** and observe the results.



We can perceive from the figures above that by reducing the number of Mel components in **MELC**, the performance indeed increased multiple folds. The training accuracy now becomes **76.35% (+25.78%)**, while the validation accuracy **58.96% (+11.88%)**.



As for the performance of MFCC data when the number of *MFCCs* is increased from **13** to **20**, the training accuracy increased slightly, while the *validation accuracy* decreased by several percentage points. This is likely a result of overfitting, as we introduce *more dimensions* of less meaningful information while retaining the same number of training data. The validation accuracy is **62.29%** (**-2.5%**), and the *training accuracy* is **97.86%** (**+0.1%**).

<b>Table 2. Number of Mel Components Juxtaposition (Accuracy)</b>		
<b>Quantity</b>	<b>Training</b>	<b>Validation</b>
20	73.54%	55.83%
30	76.25%	57.71%
40	<b>76.35%</b> ( <b>+25.78%</b> )	<b>58.96%</b> ( <b>+11.88%</b> )
50	63.54%	50.63%
128	50.57%	47.08%

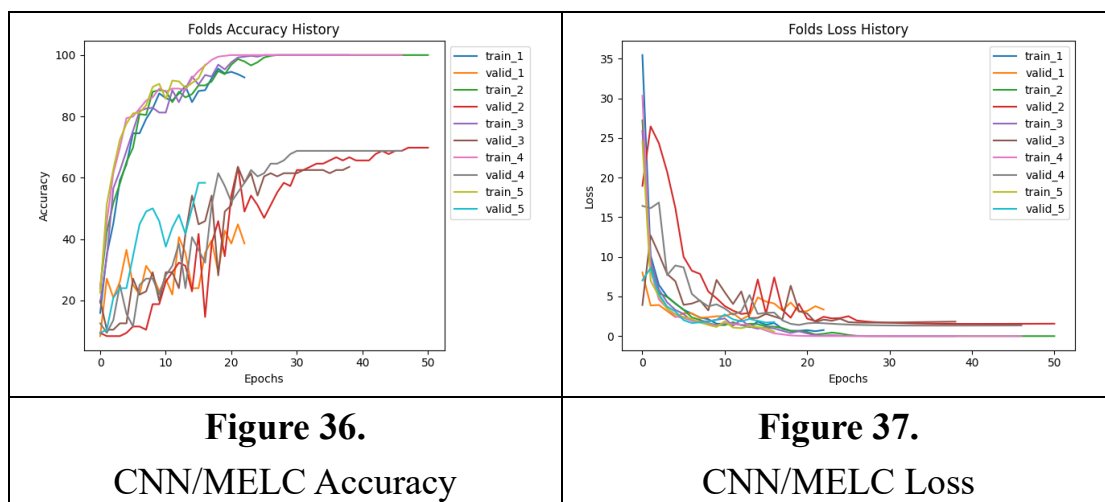
From the table above, we observe that **40** as the *Mel coefficients* quantity is optimal among all five options. However, since training each model takes more than 1 hour using the *V100* GPU and consumes a significant amount of *computing units* on *Google Colab*, we won't be experimenting to find the exact optimal value between **30** and **50**. In conclusion, it is crucial to choose *appropriate parameters* when preprocessing data.

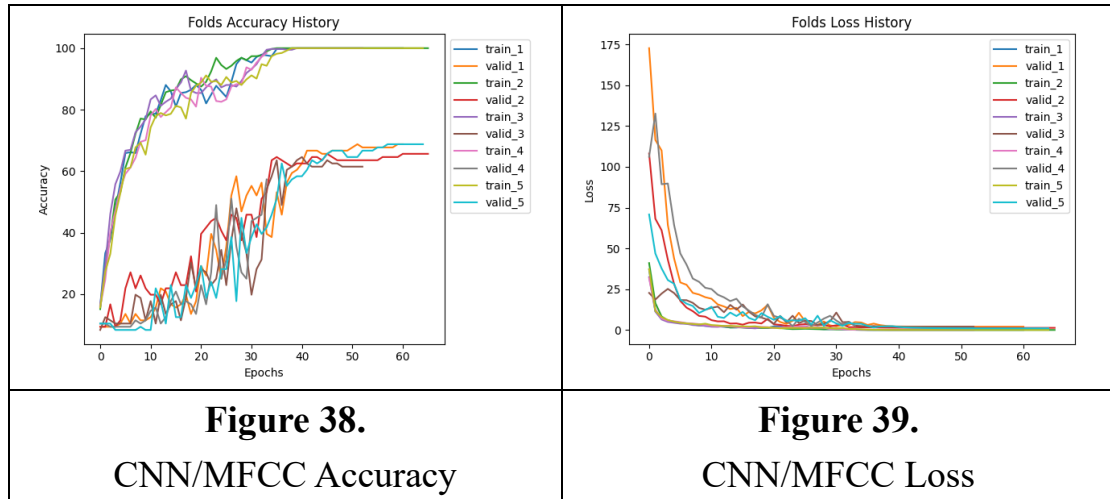
## Experimentation with Convolutional Neural Networks

Another neural network architecture which we can consider for solving music genre classification is the *convolutional neural network*. A CNN consists of several *convolutional layers*, each responsible for extracting *visual information* using so-called *kernels* which produces *feature maps*. A kernel is essentially a *matrix* that *strides* through the image, using its *weights* and *biases* to complete the operation for *feature detection*. With different parameters inside the kernel, different features are extracted, and in the case of *deep learning*, the parameters are adjusted using *large data* and algorithms like *back propagation* and *gradient descent*. At the end of convolutional layers, before the *fully connected* layers, is the *flatten* layer or *global average pooling* layer, depending on which CNN architecture.

If we consider time as another *dimension* in space, the result are figures like *spectrogram*, *mel-cepstrum*, and *MFCC* diagrams. Here we resize the images into **224x224**, and utilize **five convolutional blocks**, each of which consists of a *convolution layer*, *batch normalization layer*, *Leaky-ReLU activation layer*, and then a *max-pooling layer* for *subsampling*. After the convolution layers are a *flatten layer* and a *classification layer*.

The following figures depict the training results of this implementation of a convolutional neural network for the *music genre classification* task.

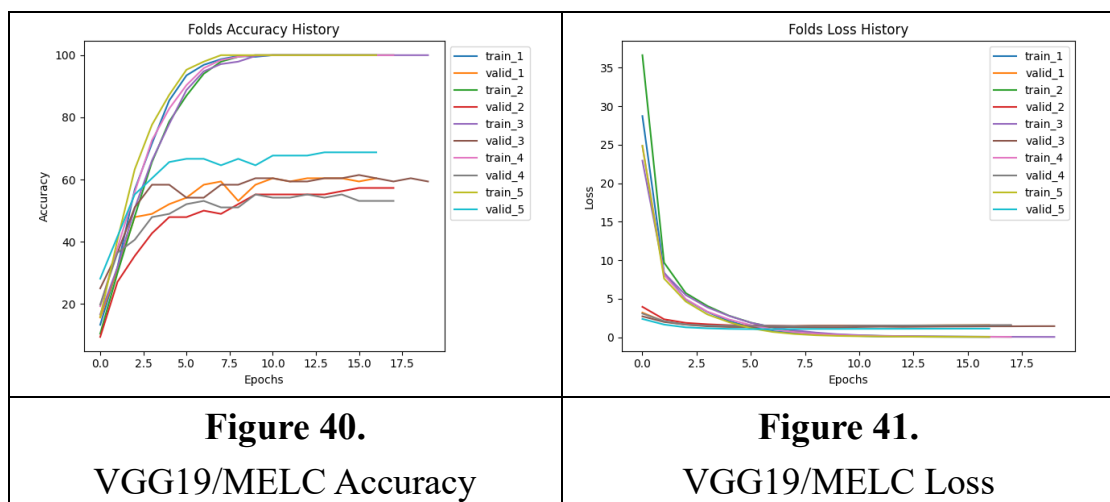


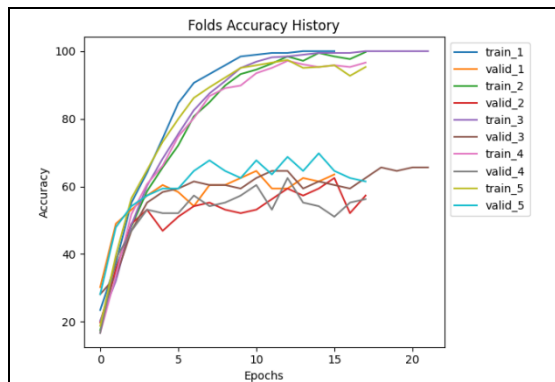


For a better understanding of the performance improvement, we are using the best recurrent models as a baseline. The training accuracy for *MFCC* is **99.69% (+1.93%)**, while the *validation accuracy* is **65.00% (+0.21%)**. As for MELC data, the *training accuracy* is **98.49% (+22.14%)**, while the validation accuracy is **61.04% (+2.08%)**.

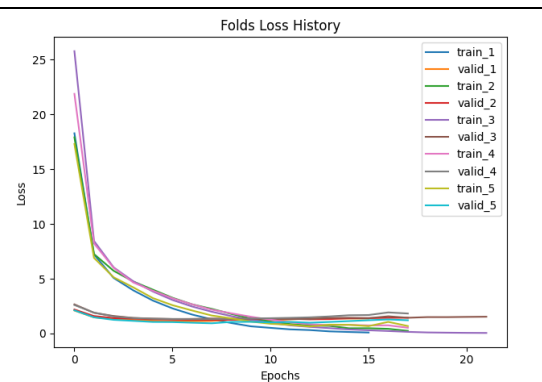
It is evident that this simple CNN model *outperformed* any recurrent models which we had previously experimented with. This is especially true with *MELC*, since its training accuracy improved by dozens of percentages, indicating that *Mel-Cepstrum* is more suitable for models like *convolutional neural networks*.

We will further experiment with other convolutional networks such as *VGG19* and *ResNet50*. However, we will not use the *pretrained weights* since our data *differ* significantly from the *ImageNet* dataset.

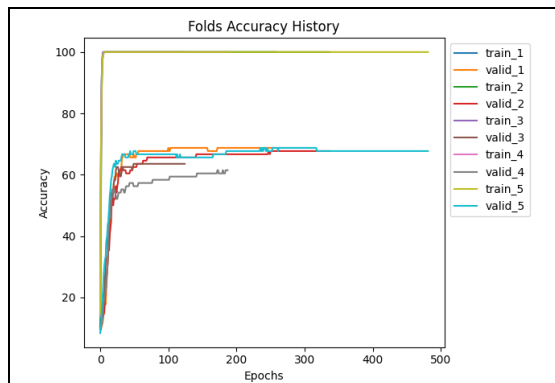




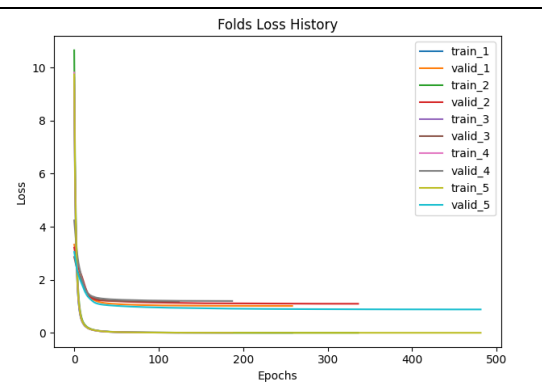
**Figure 42.**  
VGG19/MFCC Accuracy



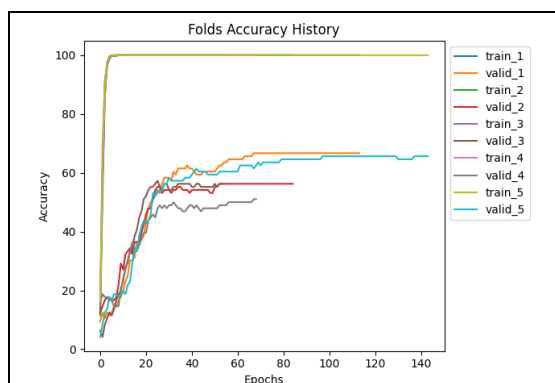
**Figure 43.**  
VGG19/MFCC Loss



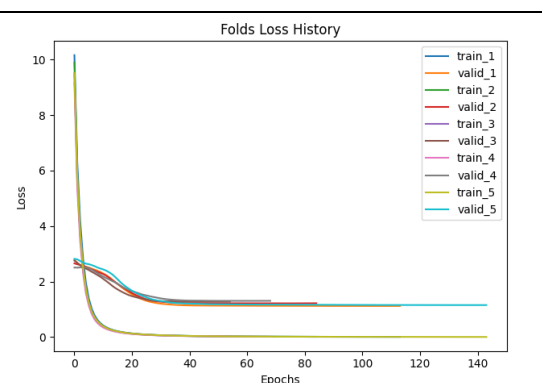
**Figure 44.**  
ResNet50/MELC Accuracy



**Figure 45.**  
ResNet50/MELC Loss



**Figure 46.**  
ResNet50/MFCC Accuracy



**Figure 47.**  
ResNet50/MFCC Loss

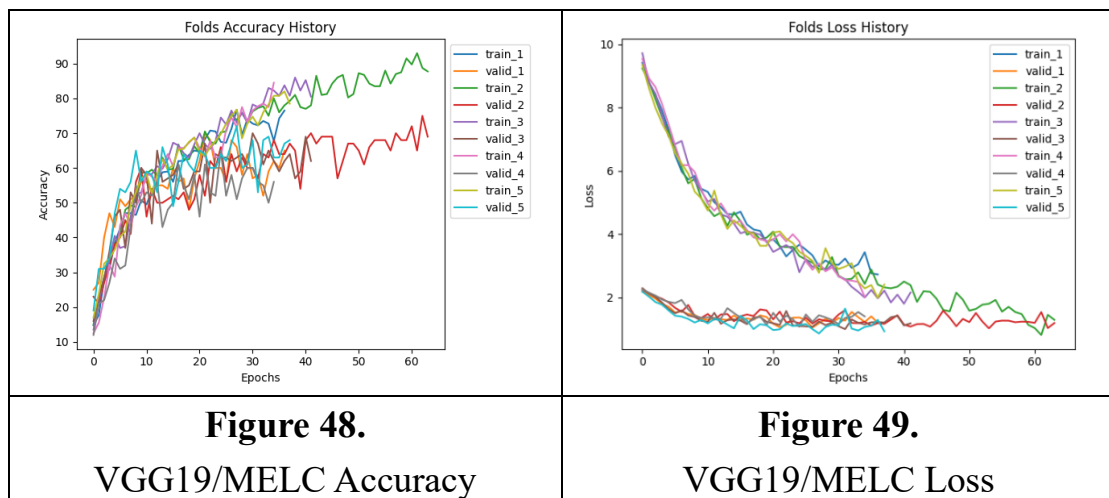


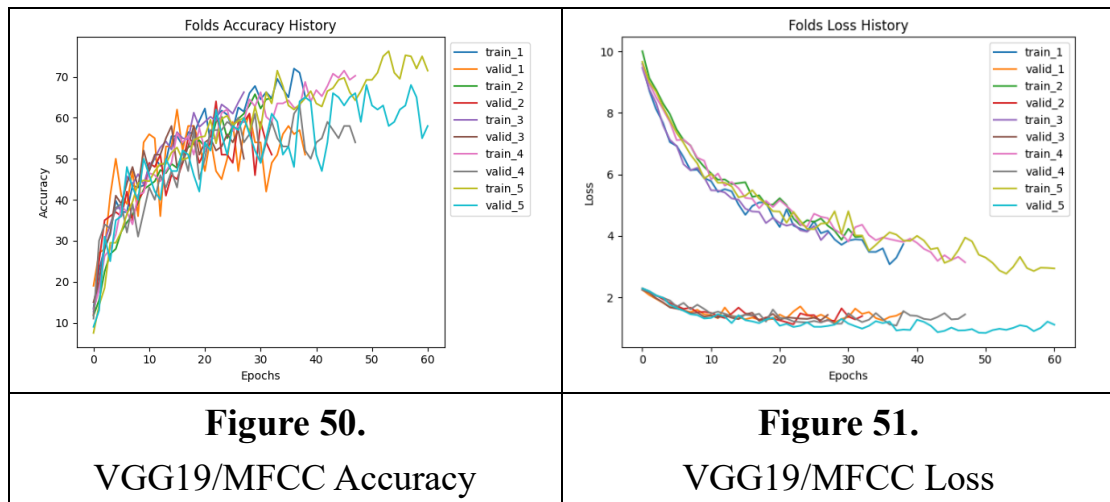
Since the model diverged rapidly soon after training began, we had to opt for a *smaller learning rate* ( $1e-5$ ) due to the complex architecture of the models. The validation accuracy for *VGG19* is **65.00%** (+0%) for MFCC and **60.63%** (-0.41%) for MELC; and regarding *ResNet50*, the validation accuracy is **59.38%** (-5.62%) for MFCC and **66.04%** (+5%) for MELC.

<b>Table 3.</b> Convolutional Network Performance Juxtaposition (VA)		
<b>Model Type</b>	<b>MELC Data [40]</b>	<b>MFCC Data [13]</b>
CNN	61.04%	<b>65.00%</b>
VGG19	60.63%	<b>65.00%</b>
ResNet50	<b>66.04%</b> (+5%)	59.38%

It is indeed surprising that *ResNet50* outperforms all other models in the *validation accuracy* of MELC Data. This is likely due to its underlying *architecture*, which is better at extracting *fine features* present in *MELC* compared to *MFCC*.

Nevertheless, it is quite intriguing to consider the possibility of further performance gains through *image augmentation*. By augmenting the data with techniques such as rotation, flipping, or zooming, we may be able to introduce some *additional variations* into the dataset, allowing the model to learn more *robust features* and potentially improve its performance even further.





Through the use of image augmentation involving *shifting*, *zooming*, and *brightness adjustments*, the validation accuracy of *VGG19* increased to **70.40% (+9.37%)** for *MELC* and **63.00% (-2%)** for *MFCC*. This kind of discovery further proves that these deep convolutional neural networks are better at extracting *finer features*. With sufficient data and appropriate image augmentation, the performance can be astounding.

This improvement well underscores the importance of *data augmentation* techniques in enhancing the performance of deep learning models, which is particular for tasks such as *music genre classification* where fine and detailed features are crucial for accurate classification.