Project Assignment: Spectrogram-Based Music Genre Classification CNN

Report by
Sanjaya Gunawardena
Liyang Fang

Objective:
The goal of this project is to develop a Convolutional Neural Network (CNN) to classify music genres based on spectrogram images of audio tracks. You will preprocess the audio data, convert it into spectrograms, and then build and train a CNN to predict the genre of each music track.

1. Data Preparation

1.1 Dataset Collection

The dataset consists of spectrogram images of audio tracks, each labeled with its respective genre. The spectrogram images were obtained by converting audio files into visual representations.

1.2 Data Loading and Preprocessing

Define Paths and Image Dimensions

- **Path to Dataset:** The dataset directory contains subdirectories for each genre, with spectrogram images inside.
- **Image Dimensions:** We chose 128x128 pixels for the spectrogram images.

Load and Preprocess Data

We wrote a function to load spectrogram images, convert them to grayscale, resize them to 128x128 pixels, and store them in an array along with their labels. Below is a summary of the preprocessing steps:

1. Load images from the dataset directory.
2. Convert images to grayscale.
3. Resize images to 128x128 pixels.
4. Store images and corresponding labels in arrays.

Encode Labels

We used `LabelEncoder` from `sklearn` to convert genre labels into numerical format, followed by `to_categorical` from `tensorflow.keras.utils` to one-hot encode these numerical labels.

Split Data

The data was split into training (80%) and testing (20%) sets using `train_test_split` from `sklearn`.

Normalize Images

The pixel values of the spectrogram images were normalized to the range [0, 1].

Based CNN model:

We used `tensorflow.keras.models.Sequential` to define our CNN architecture, which included:

- convolutional layers with 2D filters (based model it is 7x7)
- Max-pooling layers
- Dropout layers
- A flattening layer
- Dense layers

2.2 Compile the Model

The model was compiled using the Adam optimizer with an initial learning rate of 0.0001. We used `categorical_crossentropy` as the loss function and `accuracy` as the evaluation metric.

3. Model Training

3.1 Define Callbacks

We defined the following callbacks:

- `ReduceLROnPlateau` to reduce the learning rate when the validation loss plateaus.
- `EarlyStopping` to stop training early if the validation loss does not improve for 15 consecutive epochs, restoring the best model weights.

3.2 Train the Model

We trained the model on the training data and validated it on the testing data.

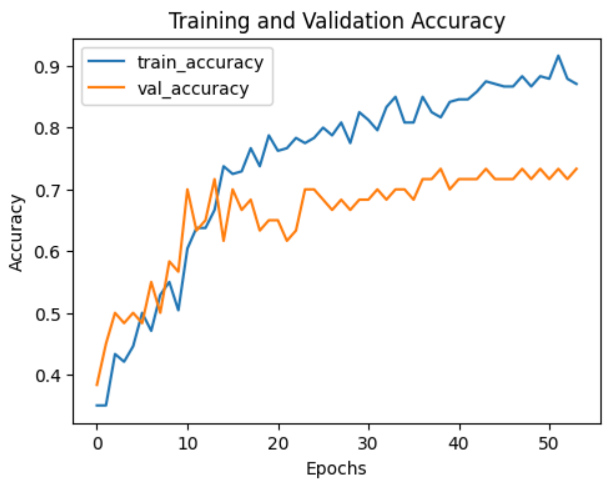
4. Model Evaluation

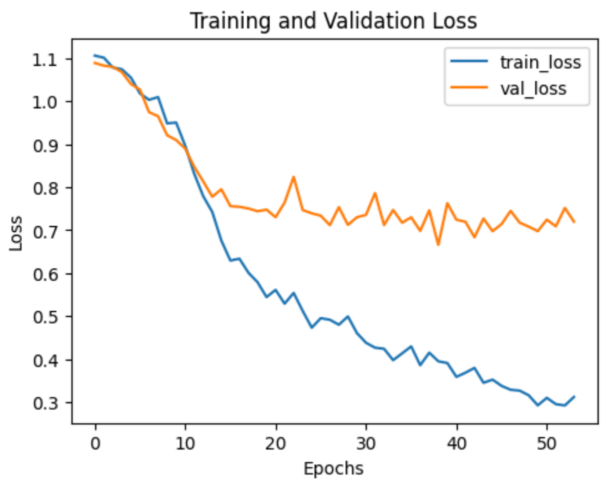
4.1 Evaluate the Model

We evaluated the trained model on the testing data to determine its accuracy.

4.2 Plot Results

We plotted the training and validation accuracy and loss over epochs to visualize the model's performance.

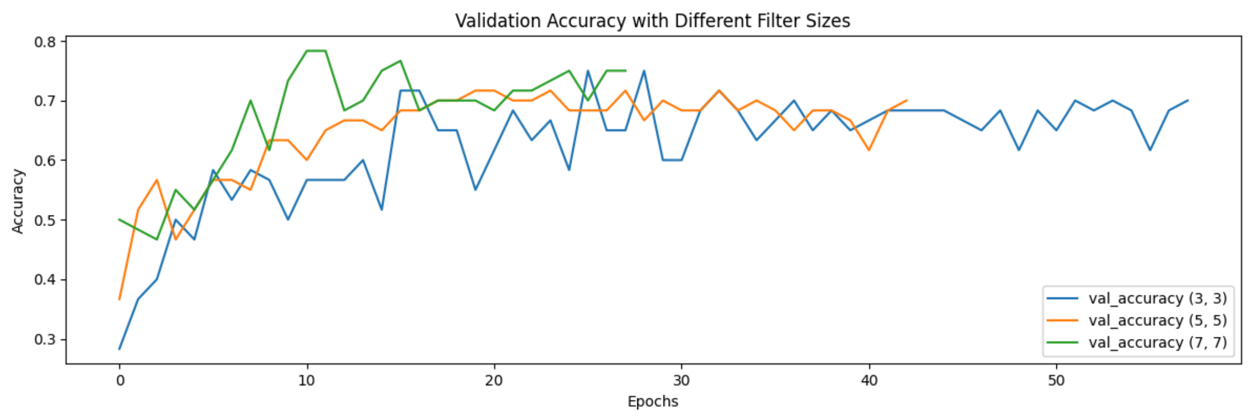


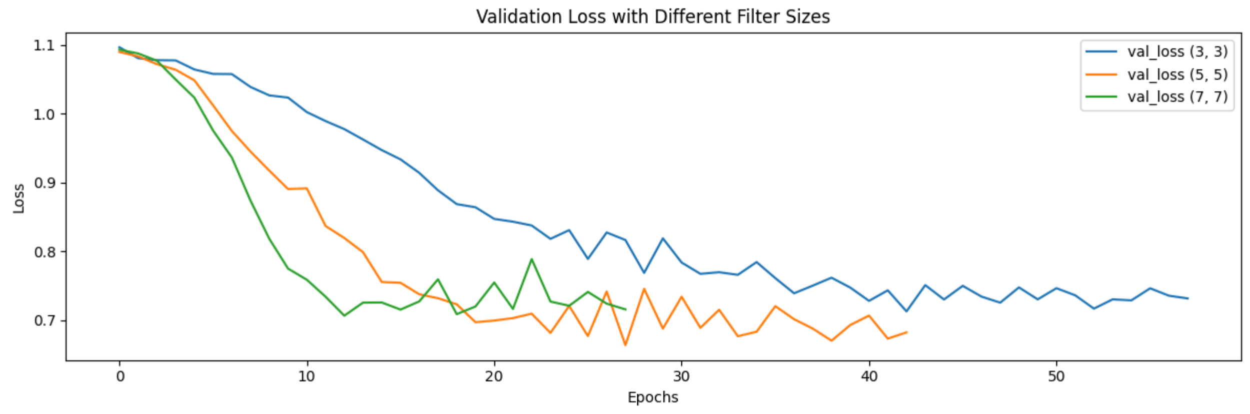


5. Exploration

5.1 Multiple Filter Sizes

We experimented with different filter sizes for the convolutional layers: (3, 3), (5, 5), and (7, 7). The results showed that a filter size of (7, 7) provided the best accuracy.

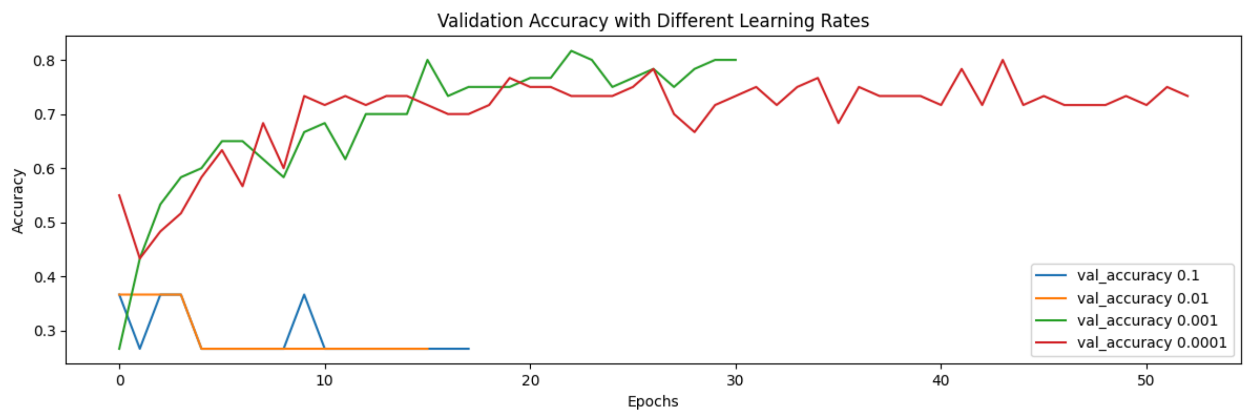


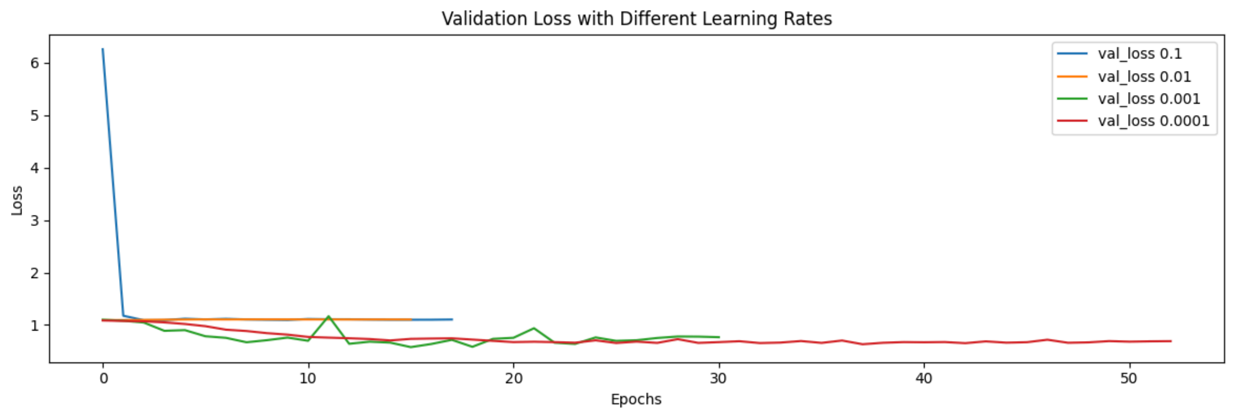


- **Filter Size (3, 3):** The model achieved moderate accuracy with a validav tion loss that decreased steadily but showed signs of plateauing towards the end.
- **Filter Size (5, 5):** The model performed better than the (3, 3) configuration with a lower validation loss.
- **Filter Size (7, 7):** The model provided the best accuracy and the lowest validation loss, indicating that larger filters helped capture more features from the spectrogram images.

5.2 Multiple Initial Learning Rates

We tried different initial learning rates for the Adam optimizer: 0.1, 0.01, 0.001, and 0.0001. The learning rate of 0.0001 yielded the best results, with higher learning rates causing instability during training.

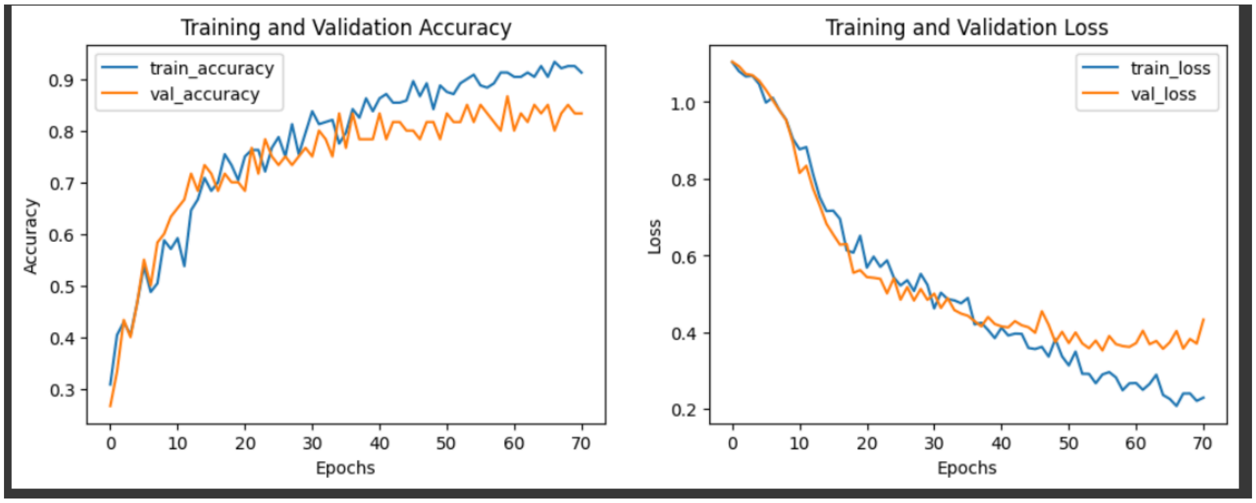


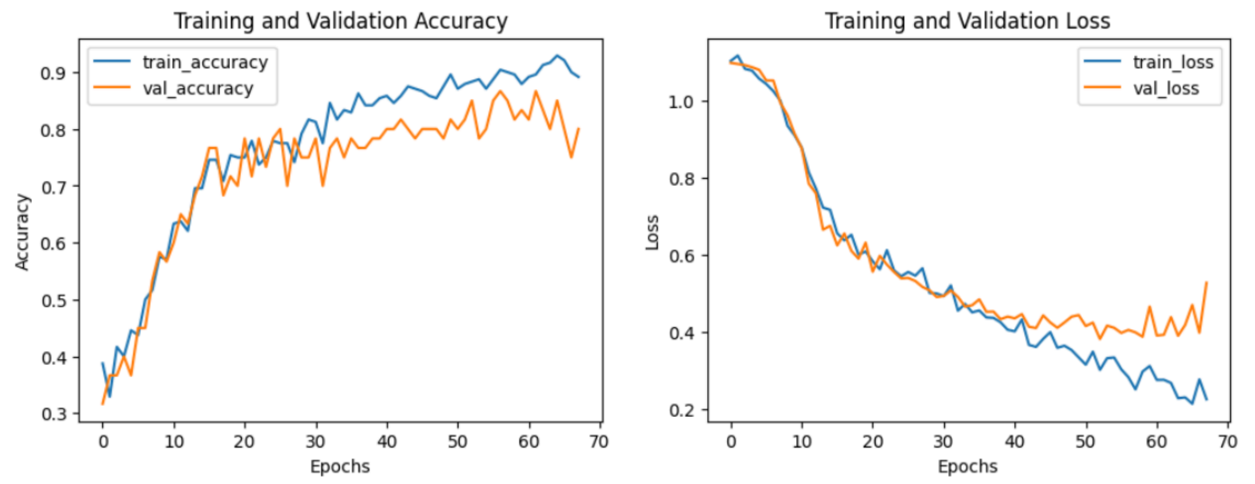


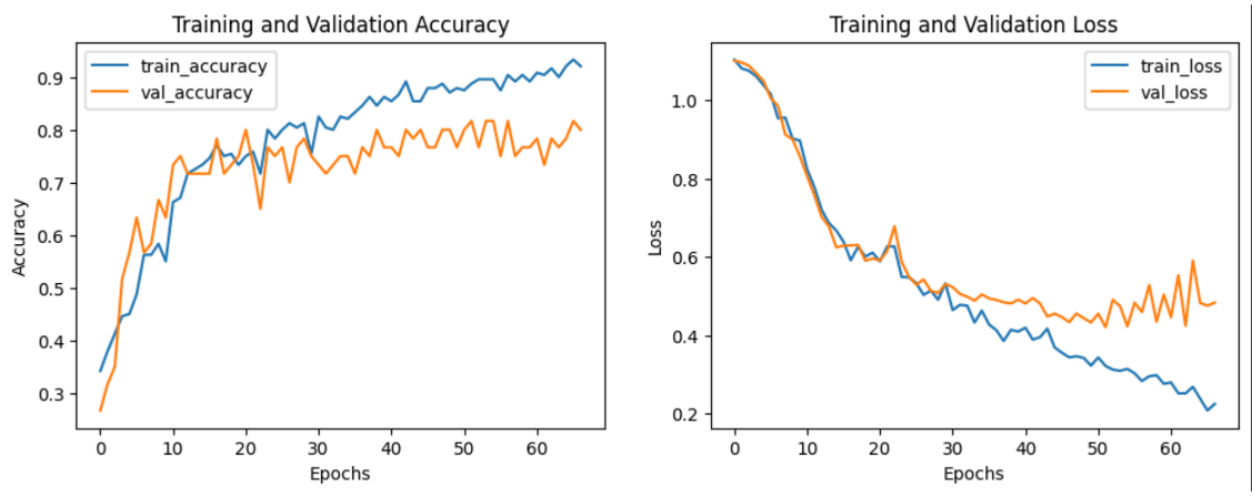
- **The learning rate of 0.0001** produced the best validation accuracy and lowest validation loss, indicating it was optimal for the given problem and dataset.
- **Higher learning rates (0.1 and 0.01)** led to instability and divergent behavior during training, causing higher validation loss and lower accuracy. A moderate learning rate (0.001) provided reasonable performance but was outperformed by the smallest learning rate.

5.3 Multiple Reduce Learning Rates Paradigms

We tried different learning rates paradigms with factor = 0.1 & patience = 5, factor = 0.3 & patience = 10 and factor = 0.5 & patience = 15. All of them, the result accuracy are quiet good but with the increase of patience and factor, it shows the probability with the overfitting.





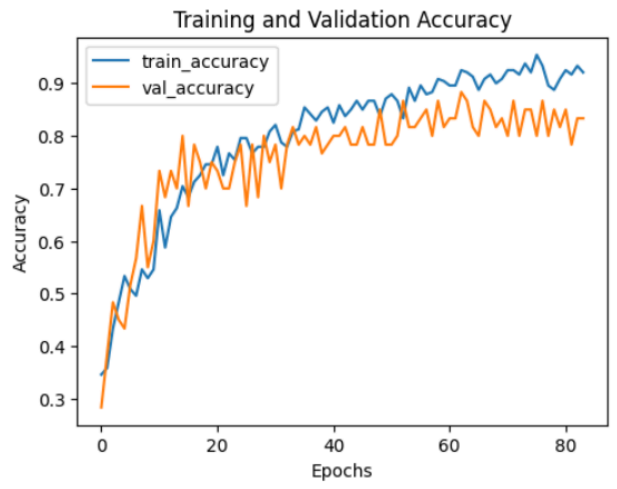


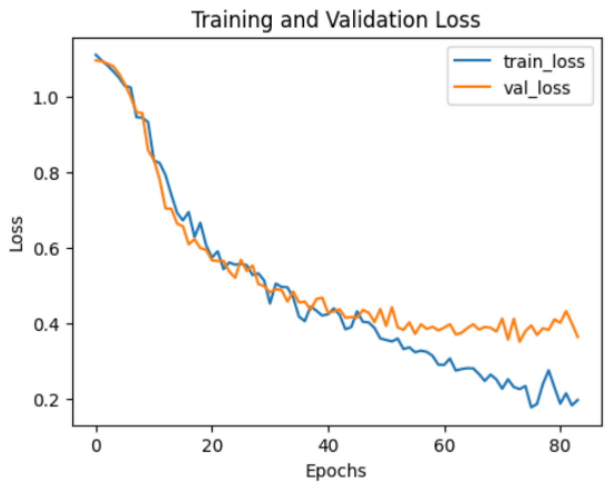
Based on the graphs above, we can see that with the lowest patience and factor, the accuracy of train and validation are most similar and close. Once the increase of patience and factor, the loss difference and accuracy difference are increasing. Therefore, we could say that the best one is with low values on patience and factor. Because it can check the accuracy frequently during the training process.

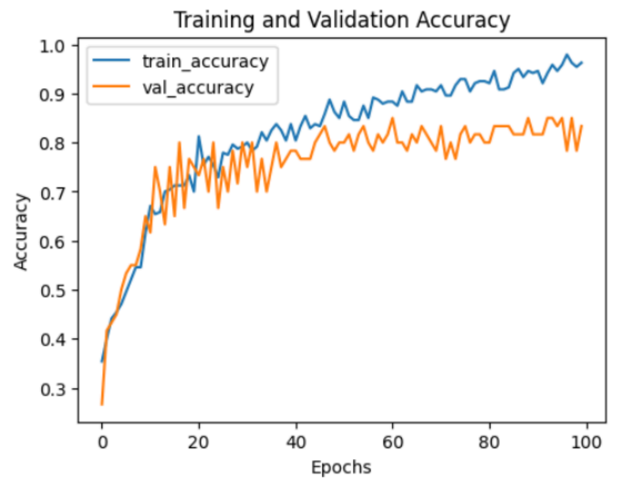
5.4 Multiple Stopping Criteria

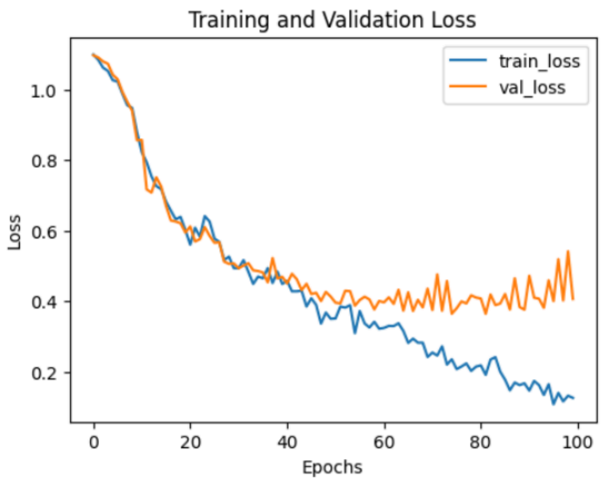
Because we used val_loss and patience = 15 as the based model, so that we will change the parameters combination with other possibilities, which are val_loss and patience = 10, val_loss and patience = 20, val_accuracy and patience = 10, val_accuracy and patience = 15 and val_accuracy and patience = 20.

Val_loss graphs



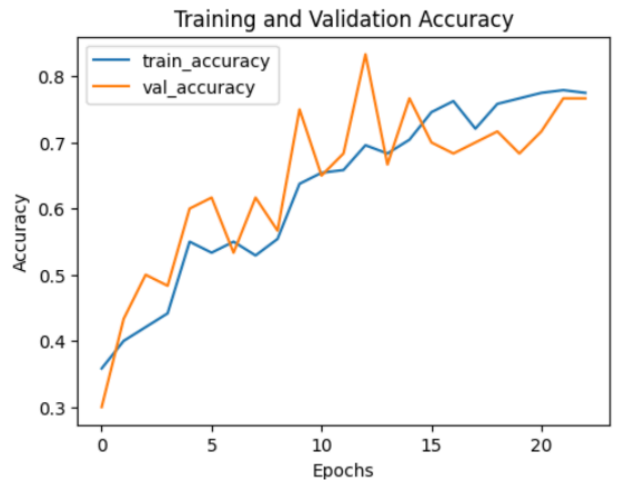


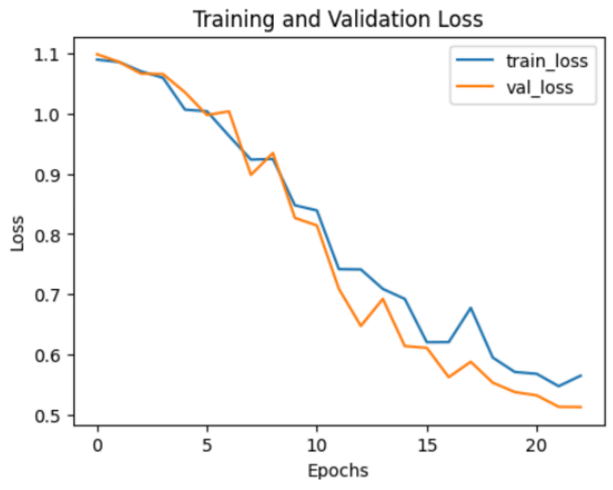


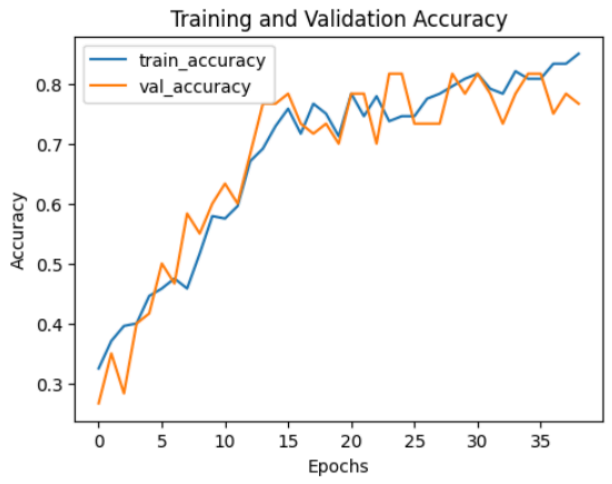


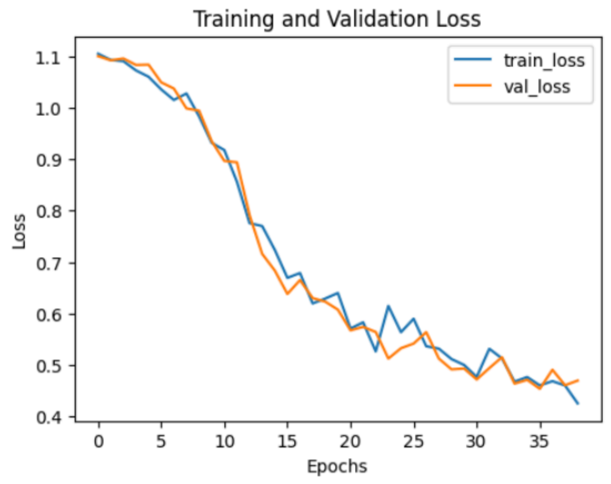
Based on these two graphs, because of the patience is different, when patience is low it only trained the model for 83 epochs. To avoid the difference in loss is too large, for the second graph we can see that after 80 epochs the loss in validation is increasing and the difference between training and validation is increasing.

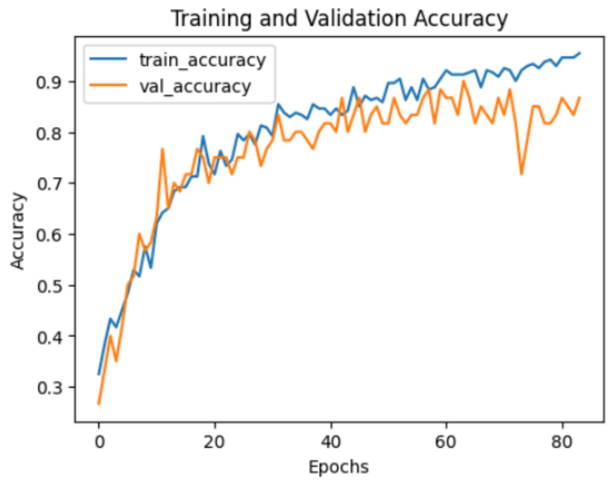
Val_accuracy graphs

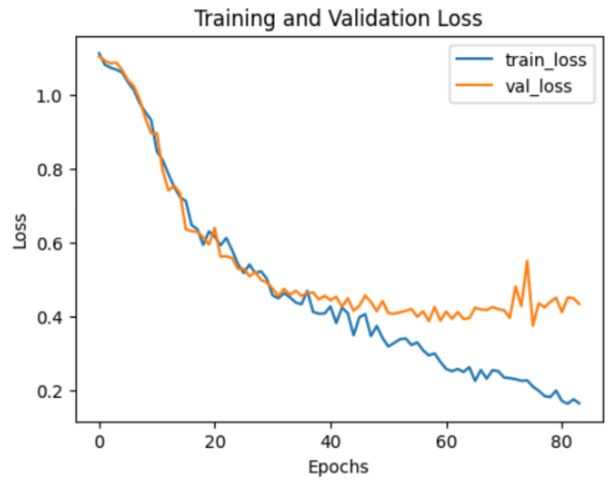












For the val_accuracy, the first two graphs only ran 23 and 38 epochs. This is because its patience is quiet low and cannot achieve the performance before stopping the training. The only one can be acceptable is the last one, with patience = 20. It ran 82 epochs, but with the same situation as val_loss, the difference between train and validation increased at the same time.

Overall, we could say that the best one is the val_loss with high patience so that can make the training model be completed and also check the trend between loss and accuracy.