**Gesture Recognition**

**Problem Statement**

Imagine you are working as a data scientist at a home electronics company which manufactures state of the art smart televisions. You want to develop a cool feature in the smart-TV that can ***recognize five different gestures performed by the user*** which will help users control the TV without using a remote.

**Gestures:**

* **Thumbs Up:** Increase the volume.
* **Thumbs Down:** Decrease the volume.
* **Left Swipe:** Jump backward 10 Sec.
* **Right Swipe:** Jump forward 10 Sec.
* **Stop:** Pause the movie.

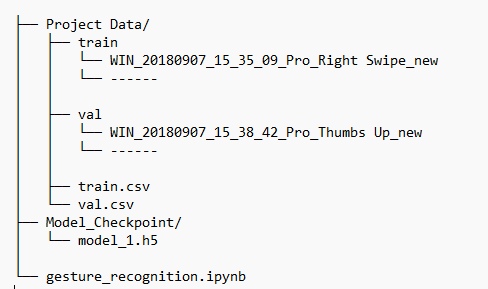
**Outline of the Project/Steps to be followed**

1. Step\_1: Importing Required Libraries to analyze the data and Train the model.
2. Step\_2: Data understanding and Data Visualization.
3. Step\_3: Data Preprocessing and custom Data Generator.
4. Step\_4: Model Building and Evaluation.
5. Method\_1: Using **Conv3D** architecture.
6. Method\_2: Using **Conv2D+LSTM** architecture.
7. Method\_3: Using **Conv2D+GRU** architecture.
8. Method\_4: Using **MobileNetV2+GRU** architecture.

**Custom Generator**

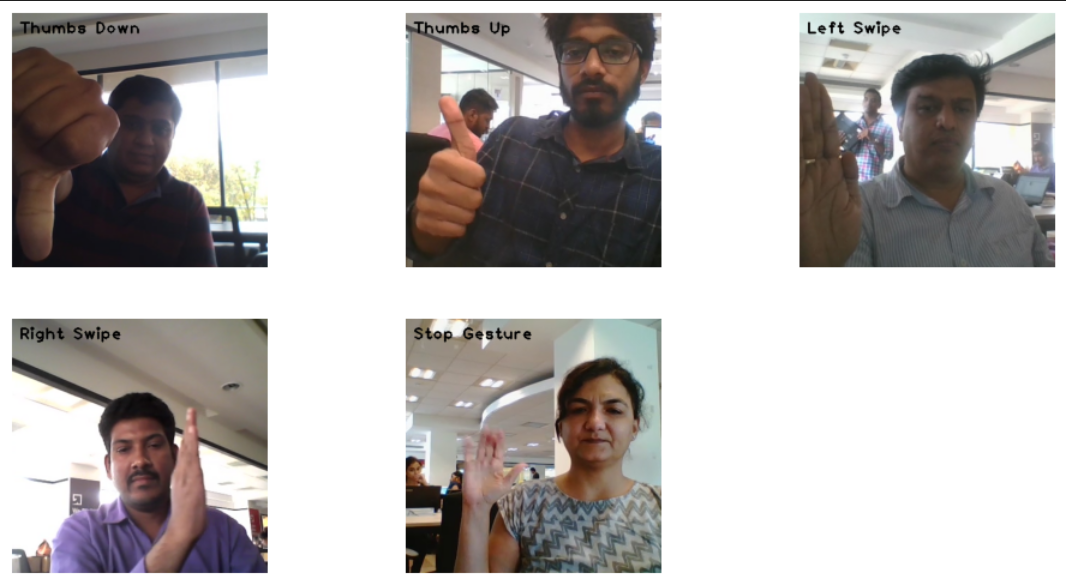
We need to create our own custom data generator to feed the data for model fitting into small batches of batch size 64. The problem with keras inbuilt data generator is not flexible enough to handle our use case.

File Structure should be look like as below to get use of custom data generator:



**Data Visualization** 

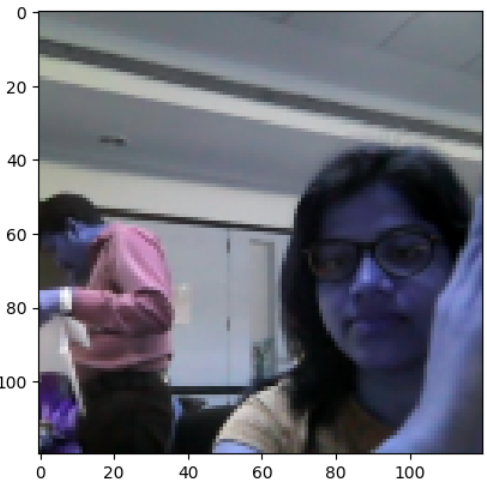
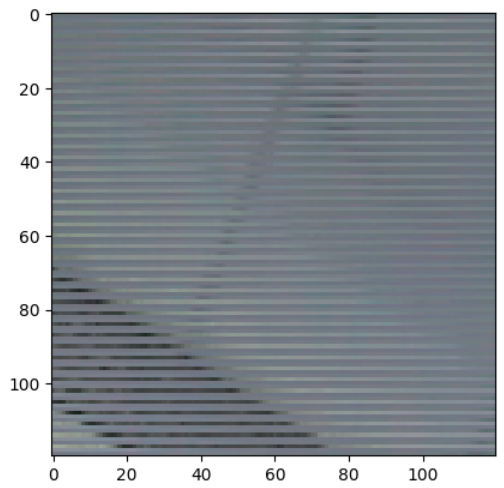
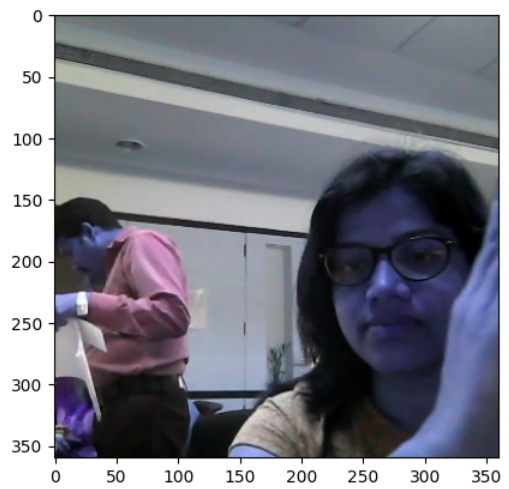
Here we get use of ***OpenCV (CV2)*** and ***matplotlib*** library to take a glance at our dataset. Cv2 for reading the path of image and convert the image into NumPy array, while matplotlib uses the plot function to plot the respective images.

  
 Fig: the above fig is representation of all the 5 classes/labels that we need to recognize.

**Data Preprocessing**

As we know our dataset containing varying shape of frames. Specifically, videos have two types of dimensions - either **360x360** or **120x160** depending on webcam to be used to record those videos. So, to handle this case we decide to resize all the frames to **120x120**.

After resizing our images/frames using ***NumPy*** module we’ve noticed that frames are not being resized properly and loosing too much information, hence we start hurdle and looking for better/robust module to help us with that problem and we came across one such library called ***skimage.***

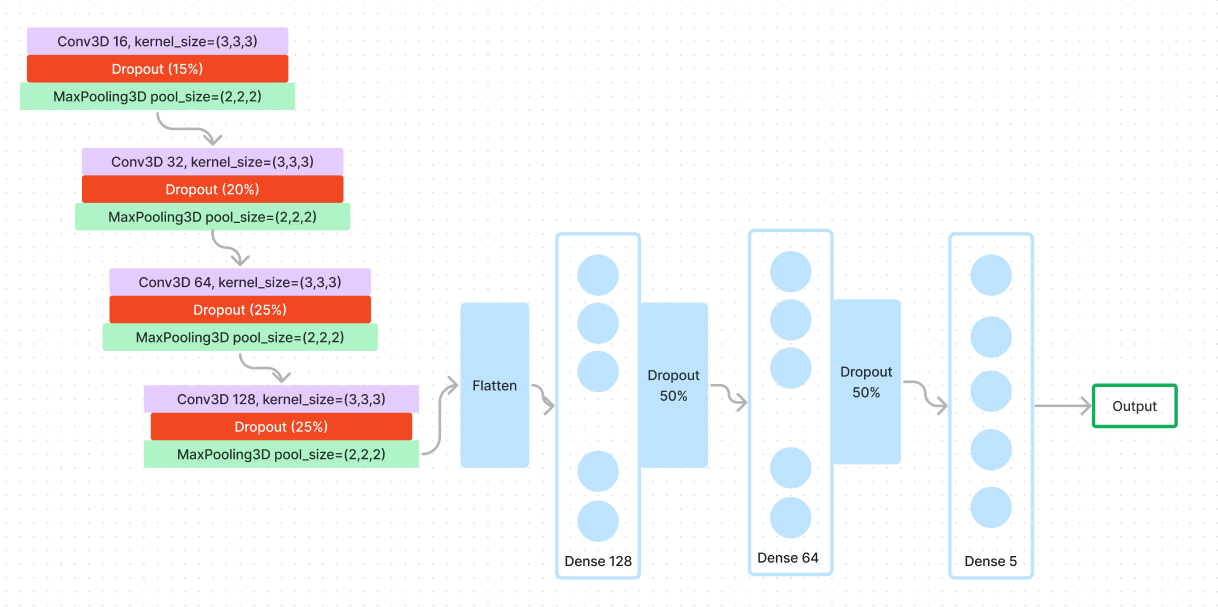
  
  
 Fig: Original Fig: using NumPy Fig: using Skimg

**Model Building**

1.Conv3D Architecture

We have decided to start with **Conv3D** model, like Conv2D it is more suitable to work with 3 dimensional or say temporal data which is in our case video classification (30 frames for each video class)

**Model Architecture**

  
 Fig: Architecture of Conv3D model

This architecture contains **four convolution layers of 16, 32, 64, 128** number of **kernels** respectively along with dropout and maxPooling layers. We choose this architecture as our base model, after fitting on train dataset and validate the model on validation set, we have got the below metrics:

**# No. Epochs 30**

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Max Categorical Accuracy** | **Loss** |
| **Train** | 49.219% | 1.0910 |
| **Validation** | 49.8% | 1.1549 |

2.Conv2D+LSTM

As per our finding on Conv3D model which is computationally more expensive and which has high parameter count can make the model more prone to overfitting. Hence, we will next choose **Conv2D** with conjunction of **LSTM** (long short-term memory networks) architecture to build our model. Which may solve training time problem with conv3D.

**Model Architecture**

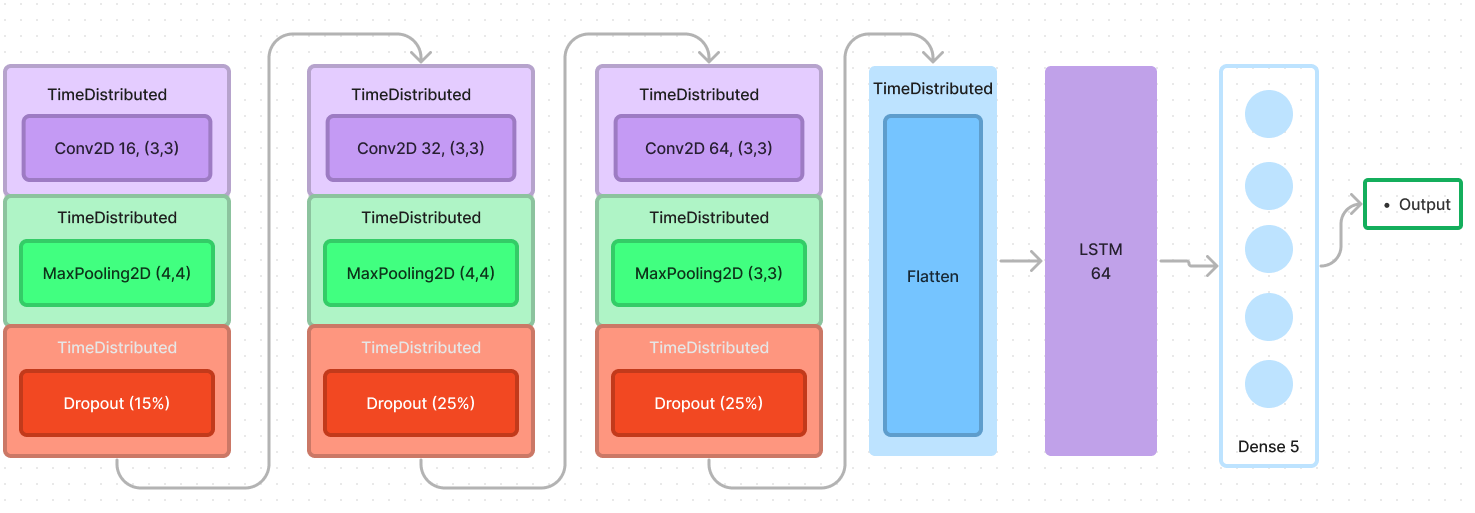


Fig: Architecture of Conv2D+LSTM model

This architecture contains **Three 2D convolution layers of 16, 32, 64** number of **kernels** respectively along with dropout and maxPooling layers which are wrapped with **TimeDistribution** layer (used for making recurrent network) after flattening the output of this layer we will then connect it to **LSTM** layer with **64** units which then directly fed the output into **Dense** layer with **5 Neurons.** after fitting the model on train dataset and validate the model on validation set, we have got the below metrics (here we are introducing callbacks to play around with EarlyStopping/LR Schedular): 

**# No. Epochs 30 (Early Stopping at Epoch 13)**

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Max Categorical Accuracy** | **Loss** |
| **Train** | 65.48% | 0.8753 |
| **Validation** | 41.406% | 1.3622 |

3.Conv2D+GRU

From the previous model we can say that **LSTMs** are prone to the problem of vanishing or exploding gradients. Gradient vanishing occurs when the gradients diminish as they propagate back in time, Gradient exploding occurs when the gradients grow exponentially, leading to **unstable training**. Hence, we move forward with **GRU** (Gated Recurrent Unit) which is better alternative to LSTM. As it offers **Faster Training** and **Inference, Less Prone to Overfitting.**

**Model Architecture**

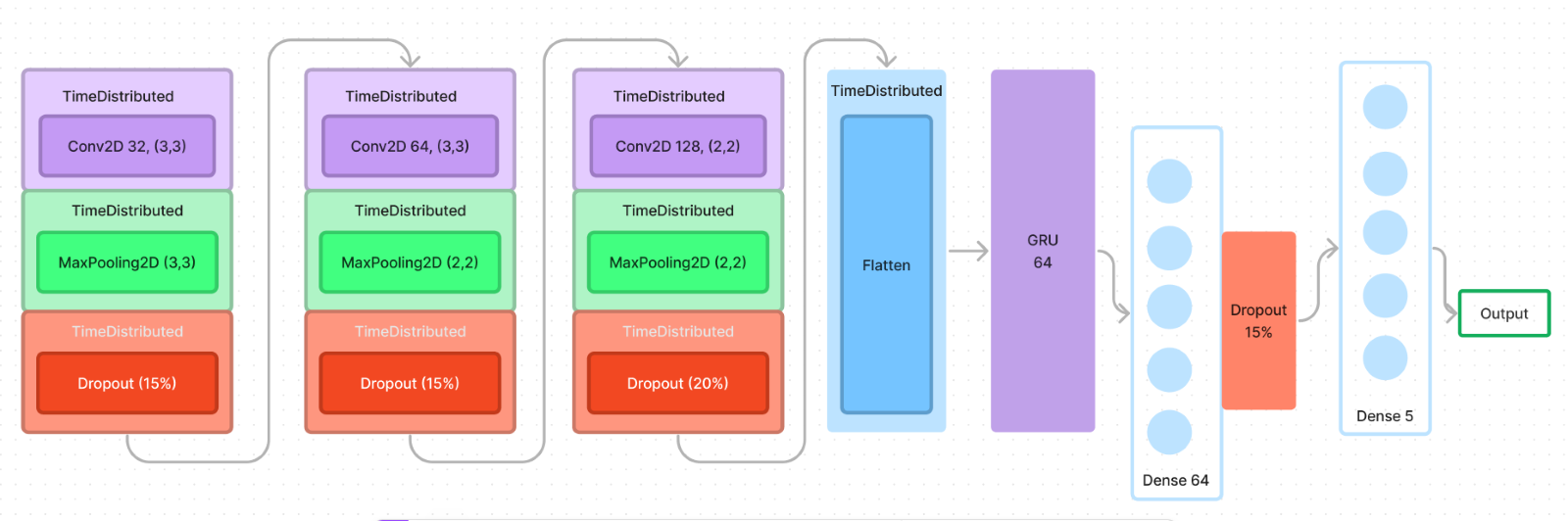


Fig: Architecture of Conv2D+GRU model

This architecture contains **Three 2D convolution layers of 32, 64, 128 number** of **kernels** respectively along with dropout and maxPooling layers which are wrapped with **TimeDistribution** layer (used for making recurrent network) after flattening the output of this layer we will then connect it to **GRU** layer with **64** units which then directly fed the output into **Dense** layer with **64 Neurons** and output from this Neurons connected to final **Dense** layer with **5** Neurons which uses **SoftMax** as activation function to predict the output**.** after fitting the model on train dataset and validate the model on validation set, we have got the below metrics (here we are using callbacks to play around with EarlyStopping/LR Schedular):

**# No. Epochs 40 (Early Stopping at Epoch 37)**

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Max Categorical Accuracy** | **Loss** |
| **Train** | 99.01 % | 0.0461 |
| **Validation** | 77.34 % | 0.8785 |

4.MobileNetV2+GRU

Above model **Conv2D+GRU** performs out of the box, but are not satisfied with its validation categorical accuracy. We then try to implement **Transfer Learning** concept with pre-trained model called **MobileNetV2** in conjunction with GRU. Before choosing this model, we had tried the combination of **VGG16** and **GRU** but VGG16 has lot of parameters and we can’t afford it terms of **training and inference time.**

**Model Architecture**

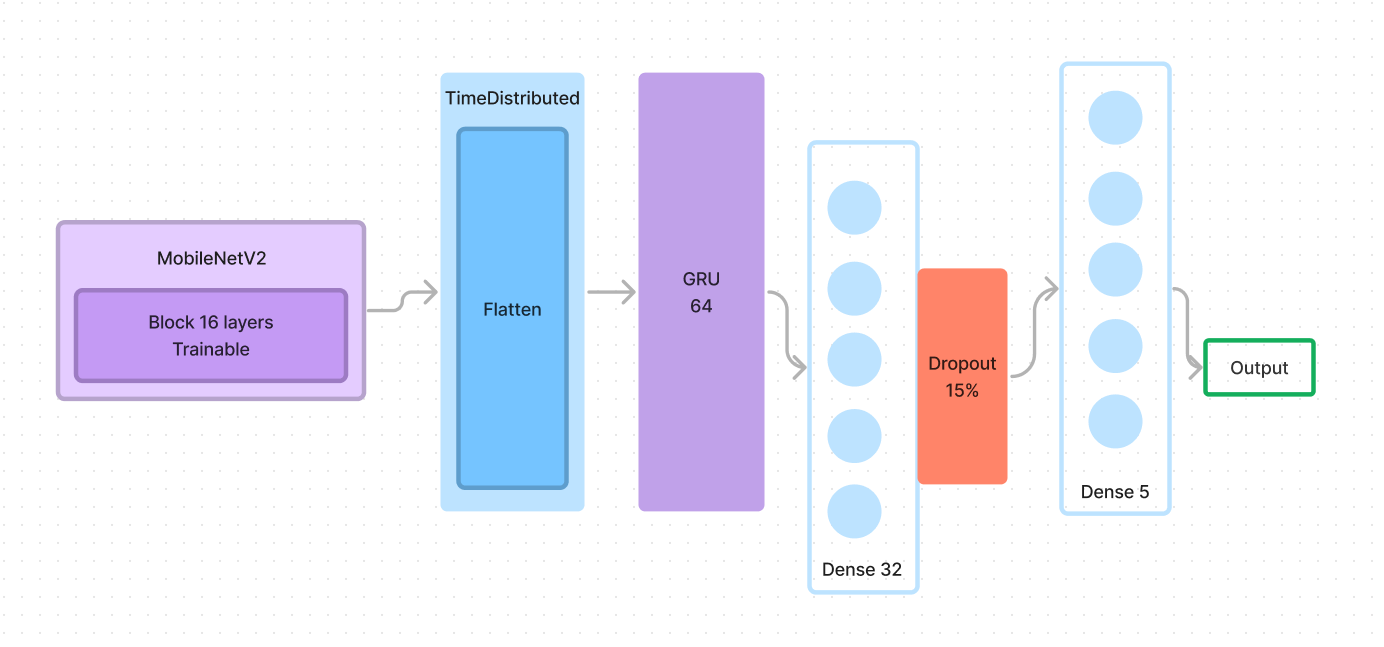


  Fig: Architecture of MobileNetV2+GRU model

**# No. Epochs 25**

|  |  |  |
| --- | --- | --- |
| **Dataset** | **Max Categorical Accuracy** | **Loss** |
| **Train** | 100 % | 0.0042 |
| **Validation** | 81.25 % | 0.5644 |

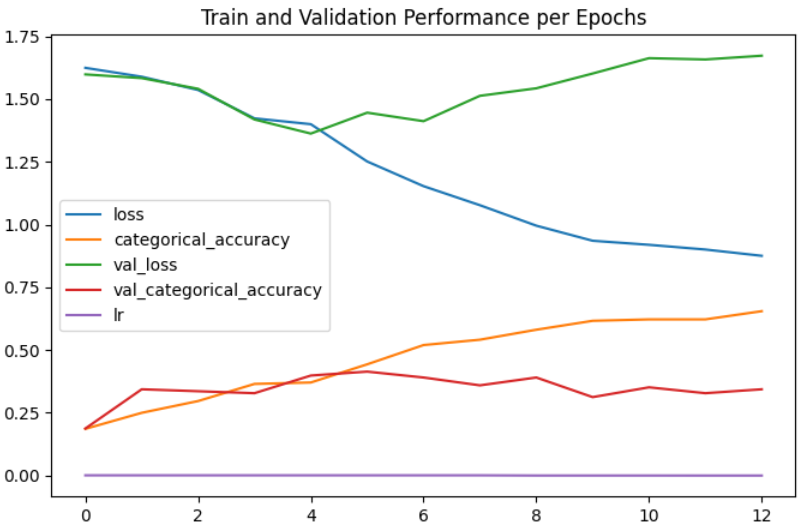
As we done with all the experiment building up different kind of architecture and comparing their performances. We come to an end of the conclusion that we choose the last model **MobileNetV2+GRU** which gave the better accuracy score of **81.25%** within small number of epochs. We can still increase the accuracy and reduce the validation loss by **hyper parameter tuning**.

**Overall Model Performance**

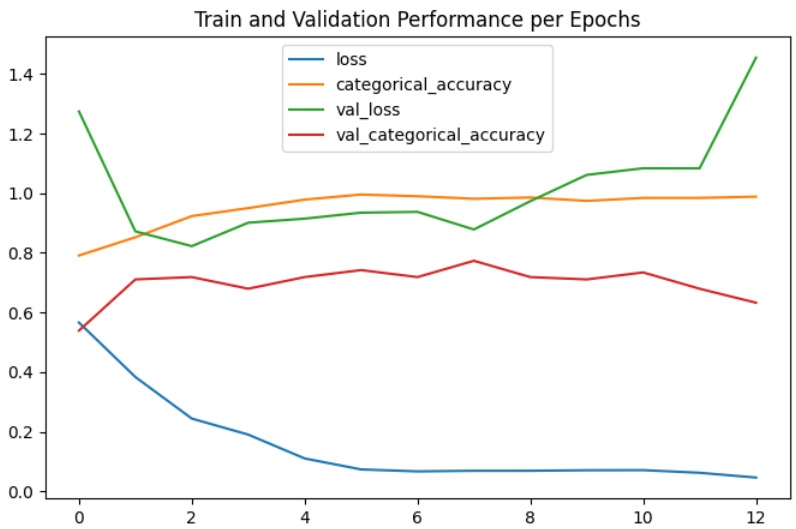
Conv3D (Base Model)



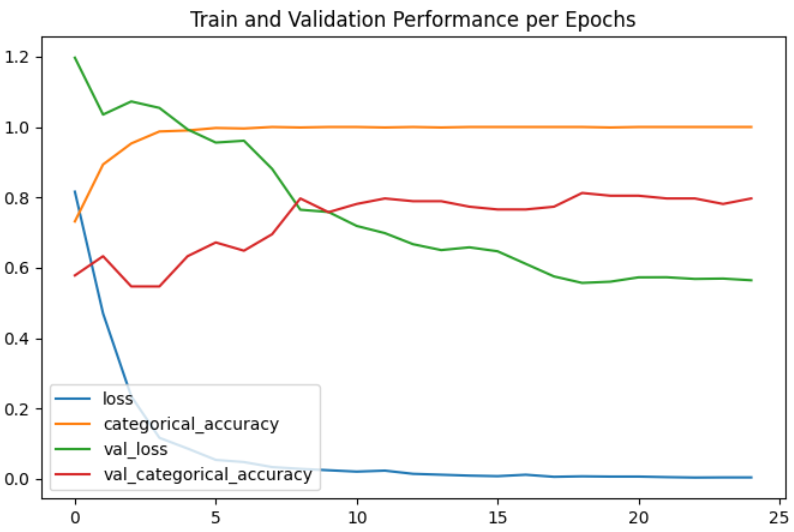
Conv2D+LSTM



Conv2D+GRU



MobileNetV2+GRU 



------------------------------- THE END -------------------------------