Deep Learning & Neural Networks

Sports Classification Using CNN Models

Team: SC_H38

Preprocessing Techniques:

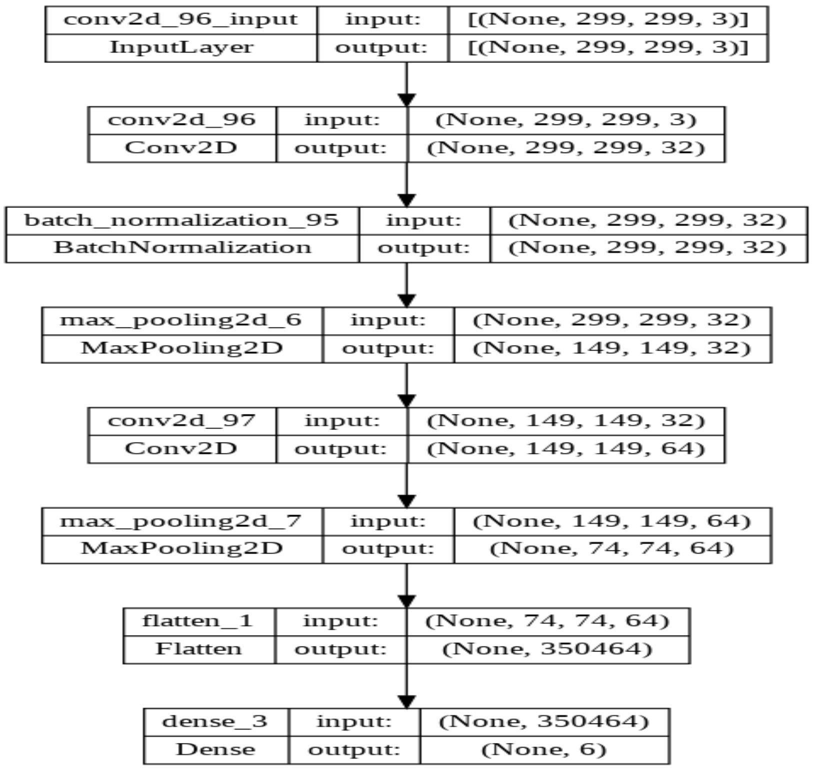
- All images were resized (will be resized) to the same size depending on the model preferred input in order to get the most optimal accuracy. This was achieved through using the Image Data Generator provided by Keras.
- All Images extensions were converted to PNG format in order to avoid data conflictions. This was achieved through using Image Data Generator provided by Keras.

Augmentation Techniques:

- Data was augmented through multiple techniques such as changing image's rotation range, ws Range, hs Range, shear Range, zoom Range, Horizontal flip and fill mode. All these parameters were randomized with the production of each batch of augmented data. It was noticed that the models' results tend to increase when more data is augmented and feed to the network.
- Throughout experimenting on the dataset satisfying results were achieved when the data was augmented to reach around 22,000 image and further augmentation didn't had a strong impact on the results.

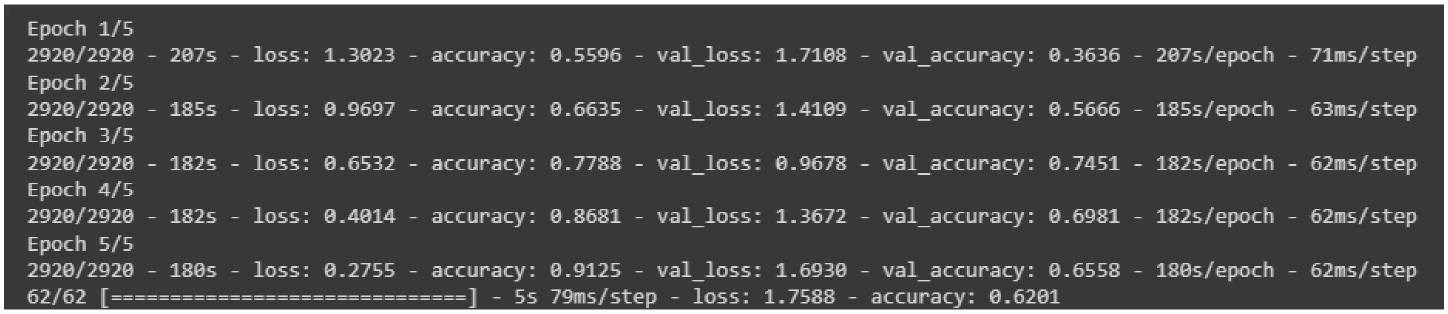
Models Used and obtained results:

- Basic CNN:


conv2d_96_input | input: | [(None, 299, 299, 3)]
InputLayer | output: | [(None, 299, 299, 3)]
conv2d_96 | input: | (None, 299, 299, 3)
Conv2D | output: | (None, 299, 299, 32)
batch_normalization_95 | input: | (None, 299, 299, 32)
BatchNormalization | output: | (None, 299, 299, 32)
max_pooling2d_6 | input: | (None, 299, 299, 32)
MaxPooling2D | output: | (None, 149, 149, 32)
conv2d_97 | input: | (None, 149, 149, 32)
Conv2D | output: | (None, 149, 149, 64)
max_pooling2d_7 | input: | (None, 149, 149, 64)
MaxPooling2D | output: | (None, 74, 74, 64)
flatten_1 | input: | (None, 74, 74, 64)
Flatten | output: | (None, 350464)
dense_3 | input: | (None, 350464)
Dense | output: | (None, 6)

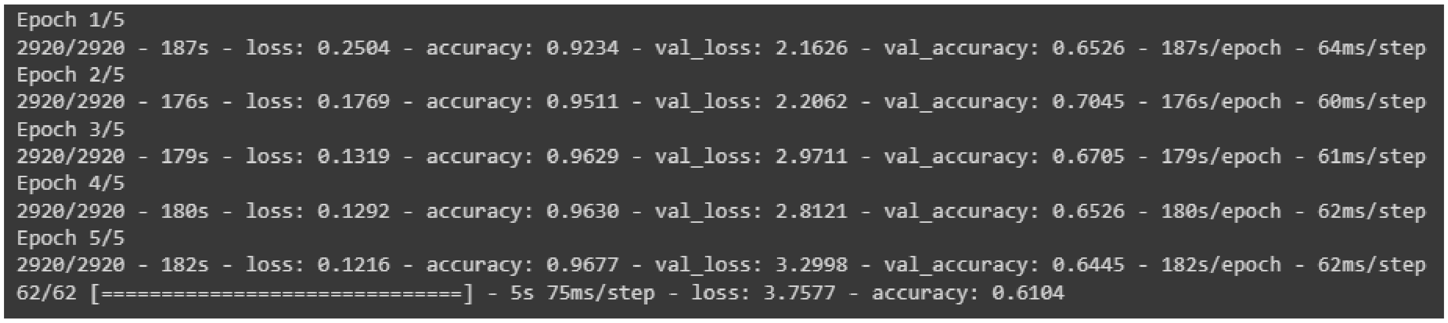
Figure(a): Basic CNN Model Architecture

On testing the model with 29,200 images on 10 epochs we get accuracy 62% on the first five epochs where each epoch takes an average of 183s/epoch and validation accuracy that reaches up to 74%.



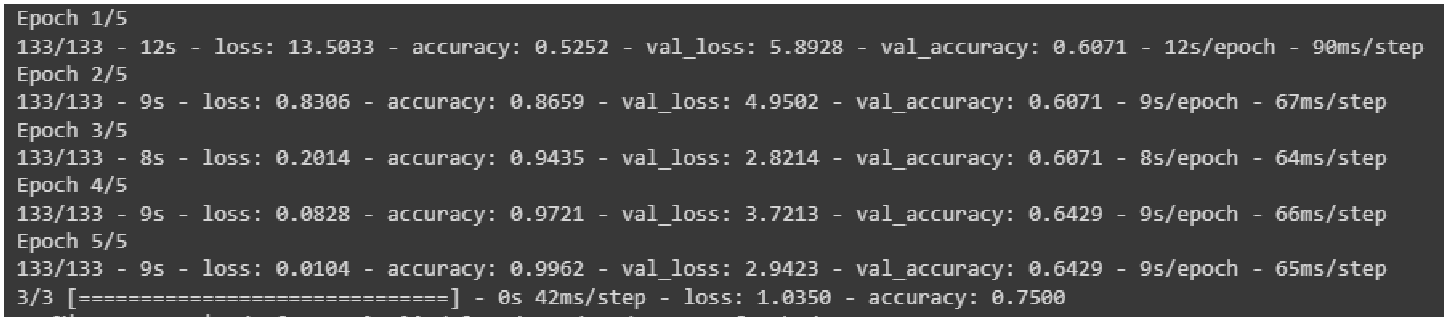
```
Epoch 1/5
2920/2920 - 207s - loss: 1.3023 - accuracy: 0.5596 - val_loss: 1.7108 - val_accuracy: 0.3636 - 207s/epoch - 71ms/step
Epoch 2/5
2920/2920 - 185s - loss: 0.9697 - accuracy: 0.6635 - val_loss: 1.4109 - val_accuracy: 0.5666 - 185s/epoch - 63ms/step
Epoch 3/5
2920/2920 - 182s - loss: 0.6532 - accuracy: 0.7788 - val_loss: 0.9678 - val_accuracy: 0.7451 - 182s/epoch - 62ms/step
Epoch 4/5
2920/2920 - 182s - loss: 0.4014 - accuracy: 0.8681 - val_loss: 1.3672 - val_accuracy: 0.6981 - 182s/epoch - 62ms/step
Epoch 5/5
2920/2920 - 180s - loss: 0.2755 - accuracy: 0.9125 - val_loss: 1.6930 - val_accuracy: 0.6558 - 180s/epoch - 62ms/step
62/62 [==============================] - 5s 79ms/step - loss: 1.7588 - accuracy: 0.6201
```

Where on the next five epochs the accuracy reaches 61% and maximum validation accuracy becomes 70%.



```
Epoch 1/5
2920/2920 - 187s - loss: 0.2504 - accuracy: 0.9234 - val_loss: 2.1626 - val_accuracy: 0.6526 - 187s/epoch - 64ms/step
Epoch 2/5
2920/2920 - 176s - loss: 0.1769 - accuracy: 0.9511 - val_loss: 2.2062 - val_accuracy: 0.7045 - 176s/epoch - 60ms/step
Epoch 3/5
2920/2920 - 179s - loss: 0.1319 - accuracy: 0.9629 - val_loss: 2.9711 - val_accuracy: 0.6705 - 179s/epoch - 61ms/step
Epoch 4/5
2920/2920 - 180s - loss: 0.1292 - accuracy: 0.9630 - val_loss: 2.8121 - val_accuracy: 0.6526 - 180s/epoch - 62ms/step
Epoch 5/5
2920/2920 - 182s - loss: 0.1216 - accuracy: 0.9677 - val_loss: 3.2998 - val_accuracy: 0.6445 - 182s/epoch - 62ms/step
62/62 [==============================] - 5s 75ms/step - loss: 3.7577 - accuracy: 0.6104
```

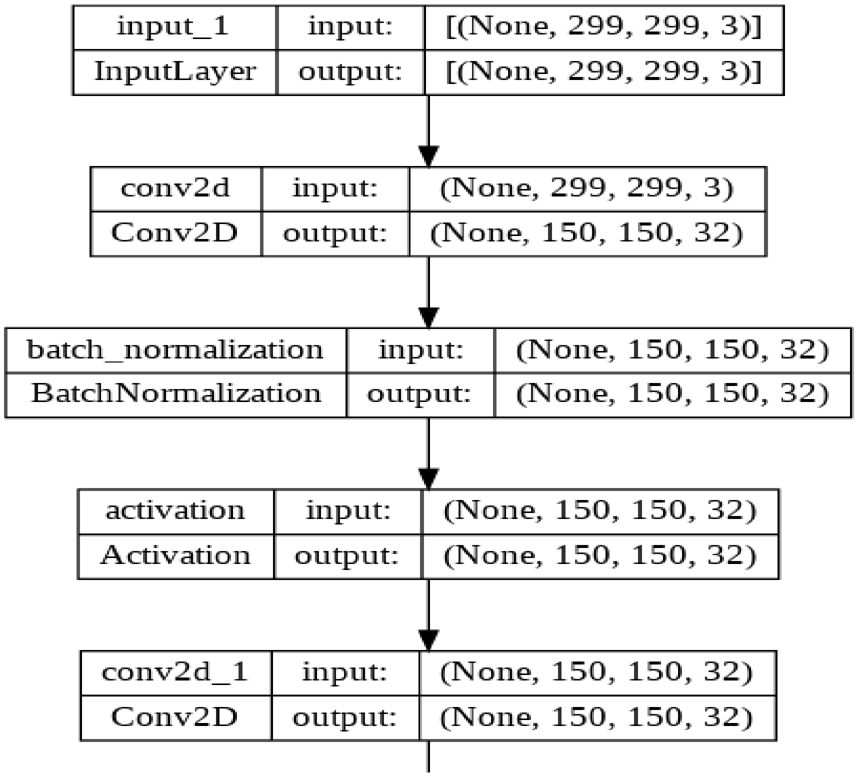
Where on 1327 images the model scores accuracy of 75% with maximum validation accuracy of 64%.



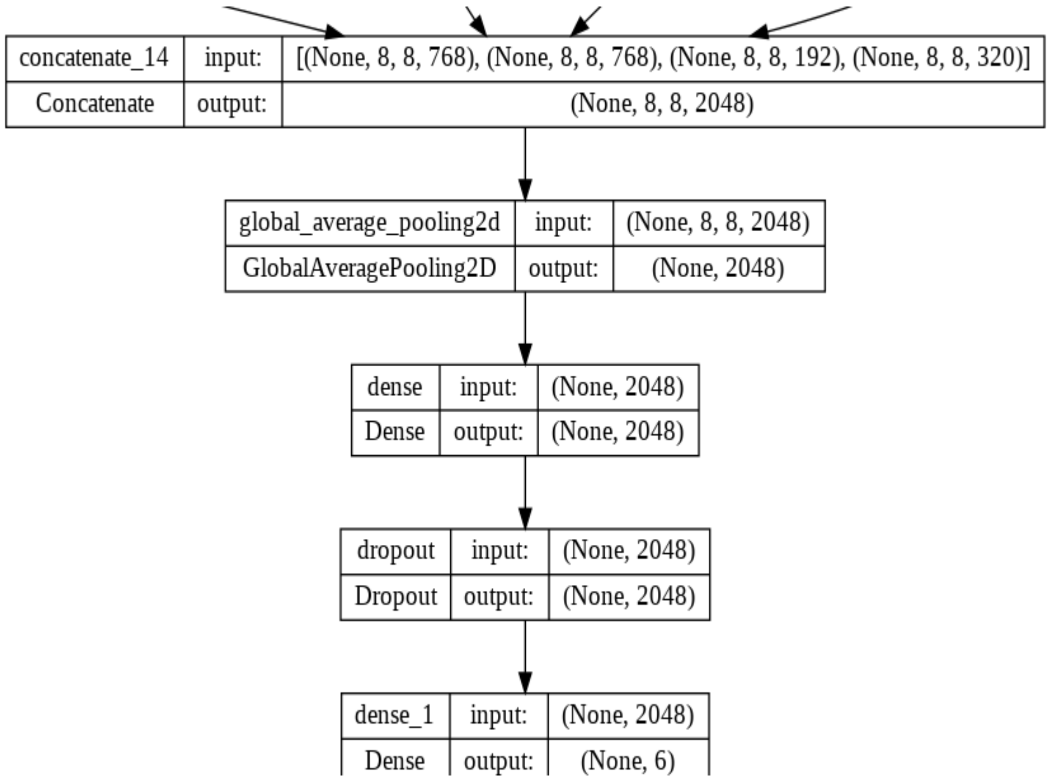
```
Epoch 1/5
133/133 - 12s - loss: 13.5033 - accuracy: 0.5252 - val_loss: 5.8928 - val_accuracy: 0.6071 - 12s/epoch - 90ms/step
Epoch 2/5
133/133 - 9s - loss: 0.8306 - accuracy: 0.8659 - val_loss: 4.9502 - val_accuracy: 0.6071 - 9s/epoch - 67ms/step
Epoch 3/5
133/133 - 8s - loss: 0.2014 - accuracy: 0.9435 - val_loss: 2.8214 - val_accuracy: 0.6071 - 8s/epoch - 64ms/step
Epoch 4/5
133/133 - 9s - loss: 0.0828 - accuracy: 0.9721 - val_loss: 3.7213 - val_accuracy: 0.6429 - 9s/epoch - 66ms/step
Epoch 5/5
133/133 - 9s - loss: 0.0104 - accuracy: 0.9962 - val_loss: 2.9423 - val_accuracy: 0.6429 - 9s/epoch - 65ms/step
3/3 [==============================] - 0s 42ms/step - loss: 1.0350 - accuracy: 0.7500
```

So, we can conculde that the basic CNN model works best on small dataset due to it’s small number of parameters which is roughly 2,122,246.

- Inception V3:

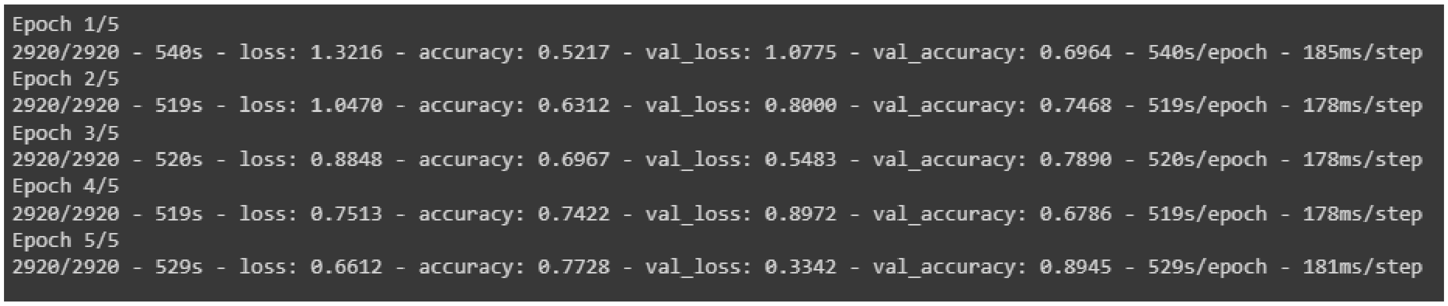


Figure(b): Inception v3 Model Architecture head



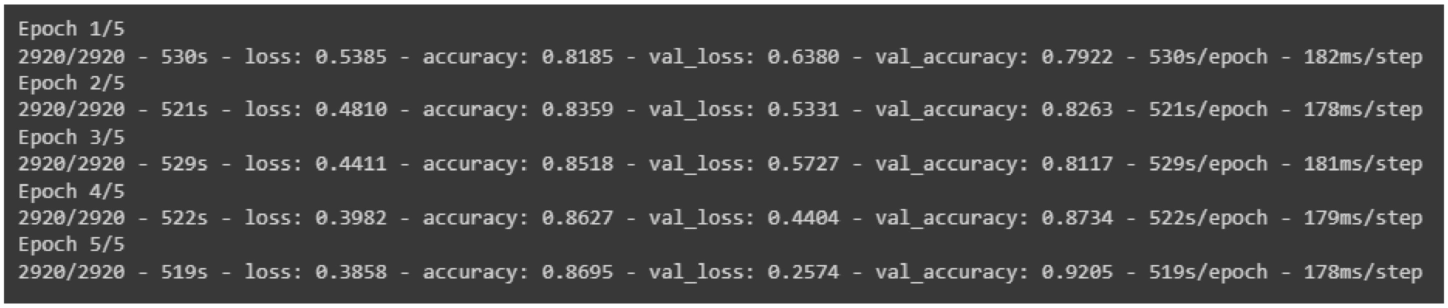
Figure(c): Inception v3 Model Architecture End

On testing the model with 29,200 images on 10 epochs we get accuracy 77% on the first five epochs where each epoch takes an average of 524s/epoch and validation accuracy that reaches up to 89%.



```
Epoch 1/5
2920/2920 - 540s - loss: 1.3216 - accuracy: 0.5217 - val_loss: 1.0775 - val_accuracy: 0.6964 - 540s/epoch - 185ms/step
Epoch 2/5
2920/2920 - 519s - loss: 1.0470 - accuracy: 0.6312 - val_loss: 0.8000 - val_accuracy: 0.7468 - 519s/epoch - 178ms/step
Epoch 3/5
2920/2920 - 520s - loss: 0.8848 - accuracy: 0.6967 - val_loss: 0.5483 - val_accuracy: 0.7890 - 520s/epoch - 178ms/step
Epoch 4/5
2920/2920 - 519s - loss: 0.7513 - accuracy: 0.7422 - val_loss: 0.8972 - val_accuracy: 0.6786 - 519s/epoch - 178ms/step
Epoch 5/5
2920/2920 - 529s - loss: 0.6612 - accuracy: 0.7728 - val_loss: 0.3342 - val_accuracy: 0.8945 - 529s/epoch - 181ms/step
```

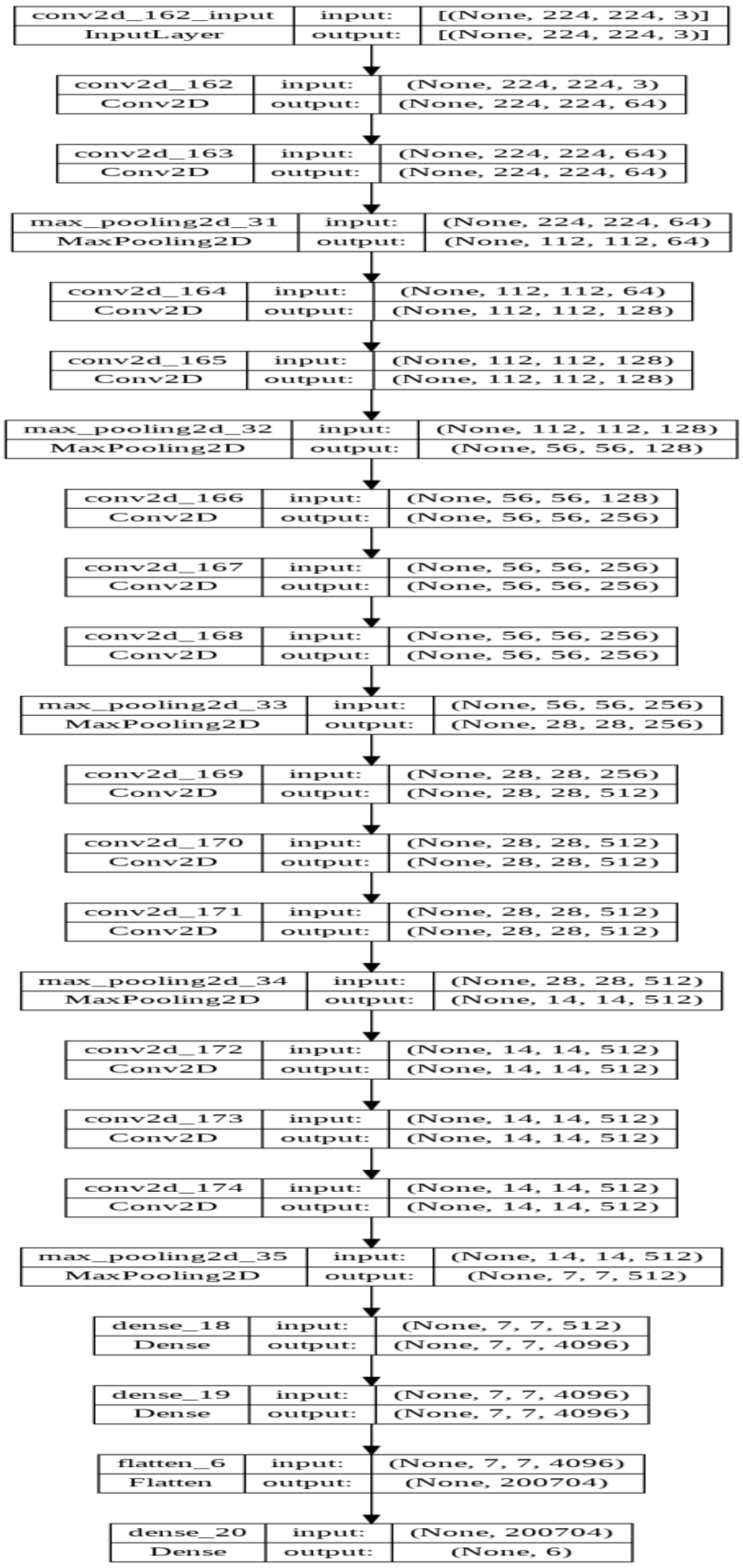
Where on the next five epochs the accuracy reaches 86% and maximum validation accuracy becomes 92%.



```
Epoch 1/5
2920/2920 - 530s - loss: 0.5385 - accuracy: 0.8185 - val_loss: 0.6380 - val_accuracy: 0.7922 - 530s/epoch - 182ms/step
Epoch 2/5
2920/2920 - 521s - loss: 0.4810 - accuracy: 0.8359 - val_loss: 0.5331 - val_accuracy: 0.8263 - 521s/epoch - 178ms/step
Epoch 3/5
2920/2920 - 529s - loss: 0.4411 - accuracy: 0.8518 - val_loss: 0.5727 - val_accuracy: 0.8117 - 529s/epoch - 181ms/step
Epoch 4/5
2920/2920 - 522s - loss: 0.3982 - accuracy: 0.8627 - val_loss: 0.4404 - val_accuracy: 0.8734 - 522s/epoch - 179ms/step
Epoch 5/5
2920/2920 - 519s - loss: 0.3858 - accuracy: 0.8695 - val_loss: 0.2574 - val_accuracy: 0.9205 - 519s/epoch - 178ms/step
```

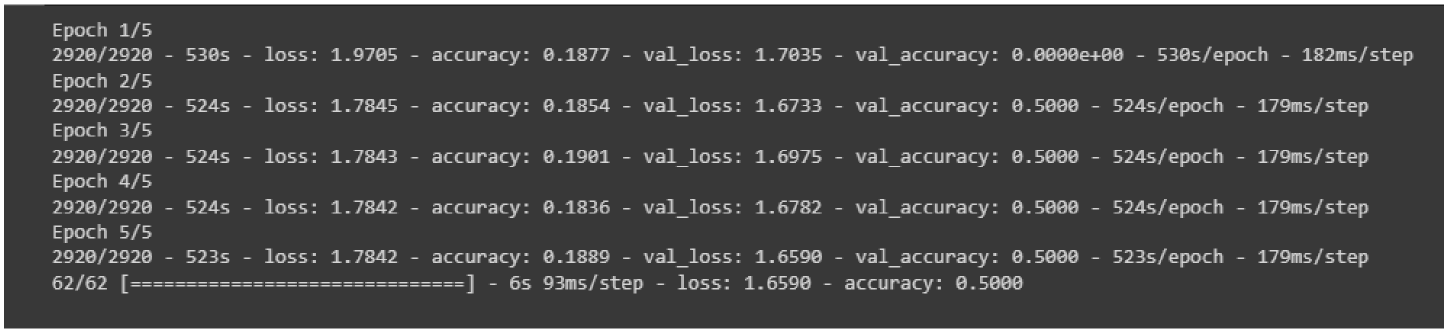
After repeating the whole process for another 5x4 epochs we achieve accuracy of 95% and thus we decide to save the model as it reaches an 84% test score.

- VGG16 Model:


conv2d_162_input | InputLayer | input: [(None, 224, 224, 3)] | output: [(None, 224, 224, 3)]
conv2d_162 | Conv2D | input: (None, 224, 224, 3) | output: (None, 224, 224, 64)
conv2d_163 | Conv2D | input: (None, 224, 224, 64) | output: (None, 224, 224, 64)
max_pooling2d_31 | MaxPooling2D | input: (None, 224, 224, 64) | output: (None, 112, 112, 64)
conv2d_164 | Conv2D | input: (None, 112, 112, 64) | output: (None, 112, 112, 128)
conv2d_165 | Conv2D | input: (None, 112, 112, 128) | output: (None, 112, 112, 128)
max_pooling2d_32 | MaxPooling2D | input: (None, 112, 112, 128) | output: (None, 56, 56, 128)
conv2d_166 | Conv2D | input: (None, 56, 56, 128) | output: (None, 56, 56, 256)
conv2d_167 | Conv2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
conv2d_168 | Conv2D | input: (None, 56, 56, 256) | output: (None, 56, 56, 256)
max_pooling2d_33 | MaxPooling2D | input: (None, 56, 56, 256) | output: (None, 28, 28, 256)
conv2d_169 | Conv2D | input: (None, 28, 28, 256) | output: (None, 28, 28, 512)
conv2d_170 | Conv2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
conv2d_171 | Conv2D | input: (None, 28, 28, 512) | output: (None, 28, 28, 512)
max_pooling2d_34 | MaxPooling2D | input: (None, 28, 28, 512) | output: (None, 14, 14, 512)
conv2d_172 | Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
conv2d_173 | Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
conv2d_174 | Conv2D | input: (None, 14, 14, 512) | output: (None, 14, 14, 512)
max_pooling2d_35 | MaxPooling2D | input: (None, 14, 14, 512) | output: (None, 7, 7, 512)
dense_18 | Dense | input: (None, 7, 7, 512) | output: (None, 7, 7, 4096)
dense_19 | Dense | input: (None, 7, 7, 4096) | output: (None, 7, 7, 4096)
flatten_6 | Flatten | input: (None, 7, 7, 4096) | output: (None, 200704)
dense_20 | Dense | input: (None, 200704) | output: (None, 6)

Figure(D): VGG16 Model Architecture

On testing the model with 29,200 images on 10 epochs we get accuracy 19% on the first five epochs where each epoch takes an average of 524s/epoch and validation accuracy that reaches up to 50%.



```
Epoch 1/5
2920/2920 - 530s - loss: 1.9705 - accuracy: 0.1877 - val_loss: 1.7035 - val_accuracy: 0.0000e+00 - 530s/epoch - 182ms/step
Epoch 2/5
2920/2920 - 524s - loss: 1.7845 - accuracy: 0.1854 - val_loss: 1.6733 - val_accuracy: 0.5000 - 524s/epoch - 179ms/step
Epoch 3/5
2920/2920 - 524s - loss: 1.7843 - accuracy: 0.1901 - val_loss: 1.6975 - val_accuracy: 0.5000 - 524s/epoch - 179ms/step
Epoch 4/5
2920/2920 - 524s - loss: 1.7842 - accuracy: 0.1836 - val_loss: 1.6782 - val_accuracy: 0.5000 - 524s/epoch - 179ms/step
Epoch 5/5
2920/2920 - 523s - loss: 1.7842 - accuracy: 0.1889 - val_loss: 1.6590 - val_accuracy: 0.5000 - 523s/epoch - 179ms/step
62/62 [==============================] - 6s 93ms/step - loss: 1.6590 - accuracy: 0.5000
```

After using those 3 models we can conclude that the best model that was used is inceptionV3 as it works well in both situation (small or large dataset).