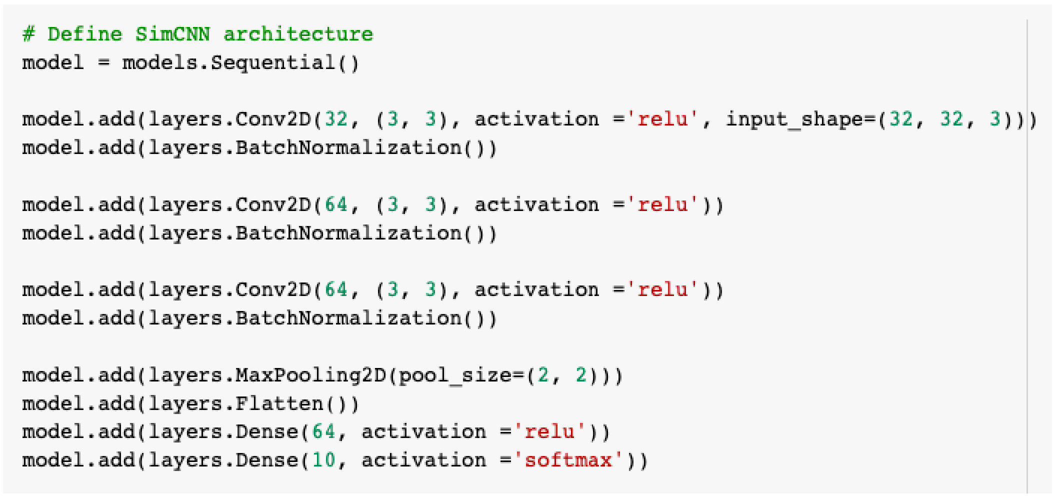
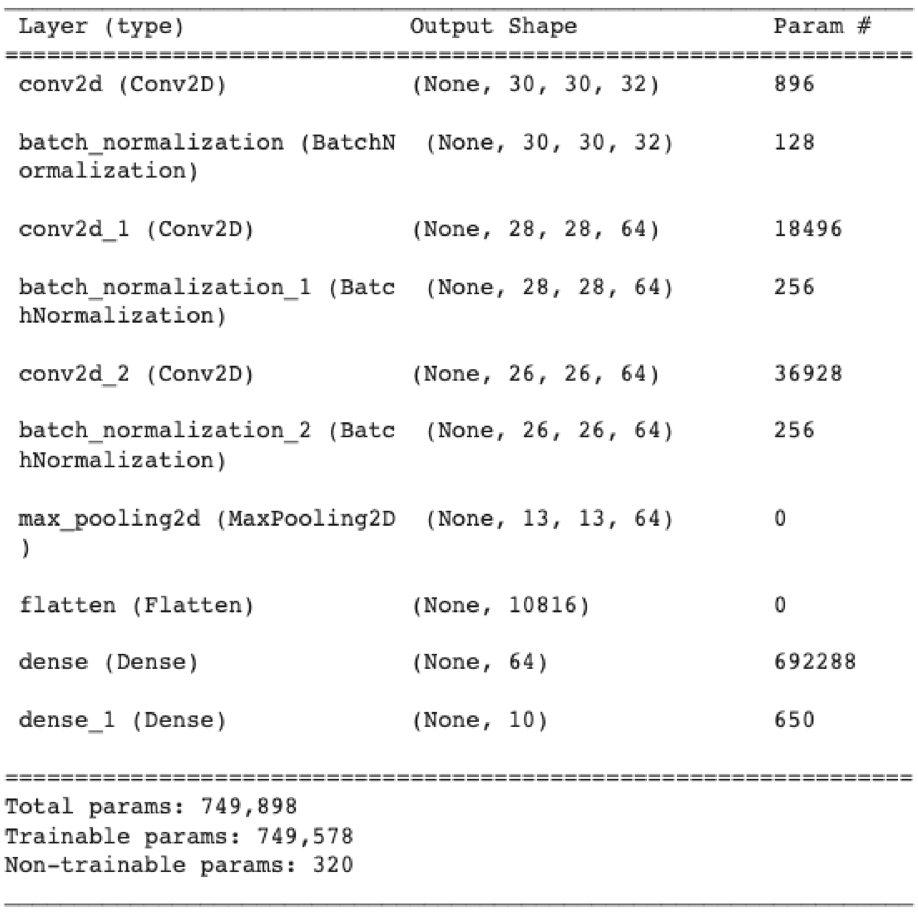
Simple CNN Model architecture:

The model has a convolutional layer with 32 filters of size 3x3, each with a 3x3 kernel and ReLU activation. A batch normalization layer has been added to help stabilize the learning process and reduce overfitting. Two more convolution layers with 64 filters, ReLu activation, and batch normalisation are added before adding a max pooling layer of size 2x2. The output is flattened and passed through a dense layer with 64 neurons and ReLU activation. Finally, the result is passed through a dense layer with ten neurons and SoftMax activation, representing the ten classes of the CIFAR-10 dataset.



```python
# Define SimCNN architecture
model = models.Sequential()

model.add(layers.Conv2D(32, (3, 3), activation ='relu', input_shape=(32, 32, 3)))
model.add(layers.BatchNormalization())

model.add(layers.Conv2D(64, (3, 3), activation ='relu'))
model.add(layers.BatchNormalization())

model.add(layers.Conv2D(64, (3, 3), activation ='relu'))
model.add(layers.BatchNormalization())

model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(64, activation ='relu'))
model.add(layers.Dense(10, activation ='softmax'))
```

Fig 1: SimCNN architecture used in the code

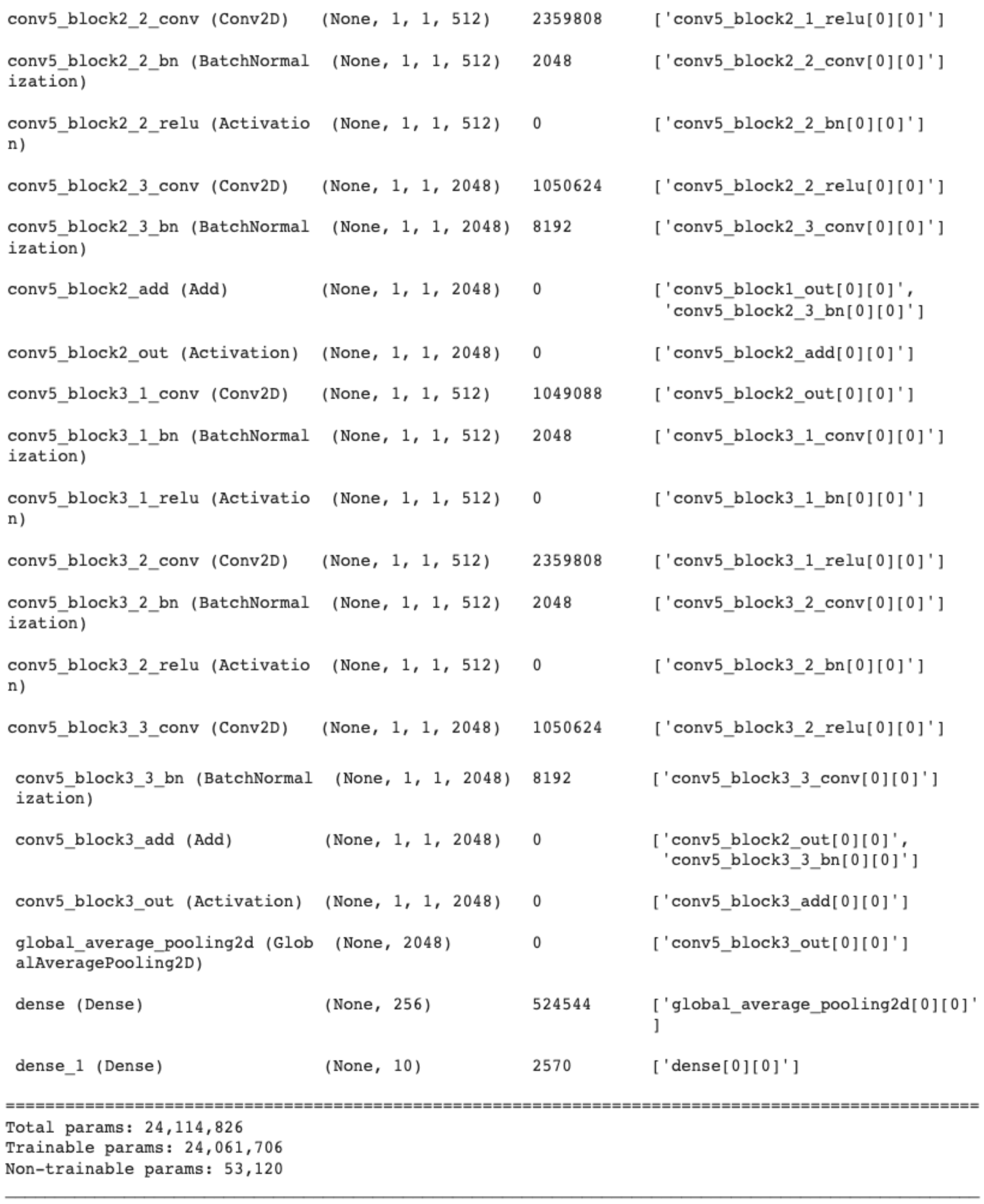


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        896

 batch_normalization (BatchN  (None, 30, 30, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 28, 28, 64)        18496

 batch_normalization_1 (Batc  (None, 28, 28, 64)       256
 hNormalization)

 conv2d_2 (Conv2D)           (None, 26, 26, 64)        36928

 batch_normalization_2 (Batc  (None, 26, 26, 64)       256
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 13, 13, 64)       0
 )

 flatten (Flatten)           (None, 10816)             0

 dense (Dense)               (None, 64)                692288

 dense_1 (Dense)             (None, 10)                650

=================================================================
Total params: 749,898
Trainable params: 749,578
Non-trainable params: 320
_________________________________________________________________
```

Fig 2: Model Summary of SimCNN network

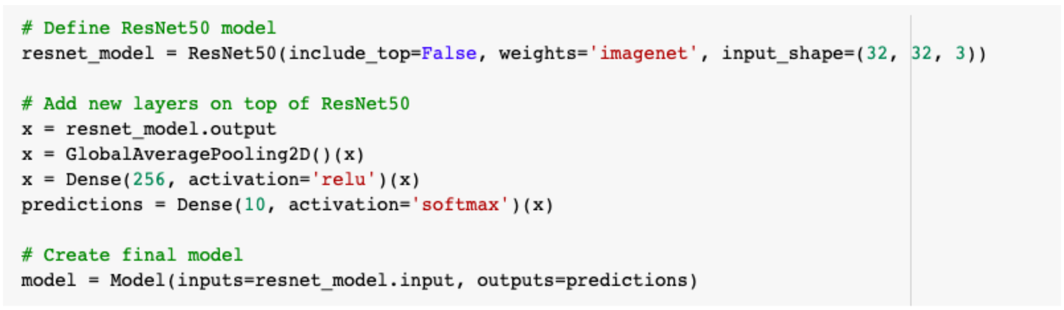
<u>ResNet50 Model architecture:</u>

A ResNet50 model pre-trained on the ImageNet dataset is modified and used in this code. New layers are then added to the ResNet50 model to adapt the input dataset, which has smaller 32x32 images.
The GlobalAveragePooling2D() layer is added to reduce the spatial dimensions of the output feature maps to a single value per channel. The resulting feature vector is passed through a Dense layer with 256 neurons and a ReLu activation function.
Finally, we add a Dense layer with ten neurons and a SoftMax activation function to produce the final class probabilities for the ten classes in the dataset.



```
conv5_block2_2_conv (Conv2D)   (None, 1, 1, 512)    2359808     ['conv5_block2_1_relu[0][0]']

conv5_block2_2_bn (BatchNormal  (None, 1, 1, 512)   2048        ['conv5_block2_2_conv[0][0]']
ization)

conv5_block2_2_relu (Activatio  (None, 1, 1, 512)   0           ['conv5_block2_2_bn[0][0]']
n)

conv5_block2_3_conv (Conv2D)   (None, 1, 1, 2048)   1050624     ['conv5_block2_2_relu[0][0]']

conv5_block2_3_bn (BatchNormal  (None, 1, 1, 2048)  8192        ['conv5_block2_3_conv[0][0]']
ization)

conv5_block2_add (Add)         (None, 1, 1, 2048)   0           ['conv5_block1_out[0][0]',
                                                                 'conv5_block2_3_bn[0][0]']

conv5_block2_out (Activation)  (None, 1, 1, 2048)   0           ['conv5_block2_add[0][0]']

conv5_block3_1_conv (Conv2D)   (None, 1, 1, 512)    1049088     ['conv5_block2_out[0][0]']

conv5_block3_1_bn (BatchNormal  (None, 1, 1, 512)   2048        ['conv5_block3_1_conv[0][0]']
ization)

conv5_block3_1_relu (Activatio  (None, 1, 1, 512)   0           ['conv5_block3_1_bn[0][0]']
n)

conv5_block3_2_conv (Conv2D)   (None, 1, 1, 512)    2359808     ['conv5_block3_1_relu[0][0]']

conv5_block3_2_bn (BatchNormal  (None, 1, 1, 512)   2048        ['conv5_block3_2_conv[0][0]']
ization)

conv5_block3_2_relu (Activatio  (None, 1, 1, 512)   0           ['conv5_block3_2_bn[0][0]']
n)

conv5_block3_3_conv (Conv2D)   (None, 1, 1, 2048)   1050624     ['conv5_block3_2_relu[0][0]']

conv5_block3_3_bn (BatchNormal  (None, 1, 1, 2048)  8192        ['conv5_block3_3_conv[0][0]']
ization)

conv5_block3_add (Add)         (None, 1, 1, 2048)   0           ['conv5_block2_out[0][0]',
                                                                 'conv5_block3_3_bn[0][0]']

conv5_block3_out (Activation)  (None, 1, 1, 2048)   0           ['conv5_block3_add[0][0]']

global_average_pooling2d (Glob  (None, 2048)        0           ['conv5_block3_out[0][0]']
alAveragePooling2D)

dense (Dense)                  (None, 256)          524544      ['global_average_pooling2d[0][0]'
                                                                ]

dense_1 (Dense)                (None, 10)           2570        ['dense[0][0]']

==================================================================================================
Total params: 24,114,826
Trainable params: 24,061,706
Non-trainable params: 53,120
__________________________________________________________________________________________________
```

Fig 3: ResNet50 model summary.



```python
# Define ResNet50 model
resnet_model = ResNet50(include_top=False, weights='imagenet', input_shape=(32, 32, 3))

# Add new layers on top of ResNet50
x = resnet_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(256, activation='relu')(x)
predictions = Dense(10, activation='softmax')(x)

# Create final model
model = Model(inputs=resnet_model.input, outputs=predictions)
```

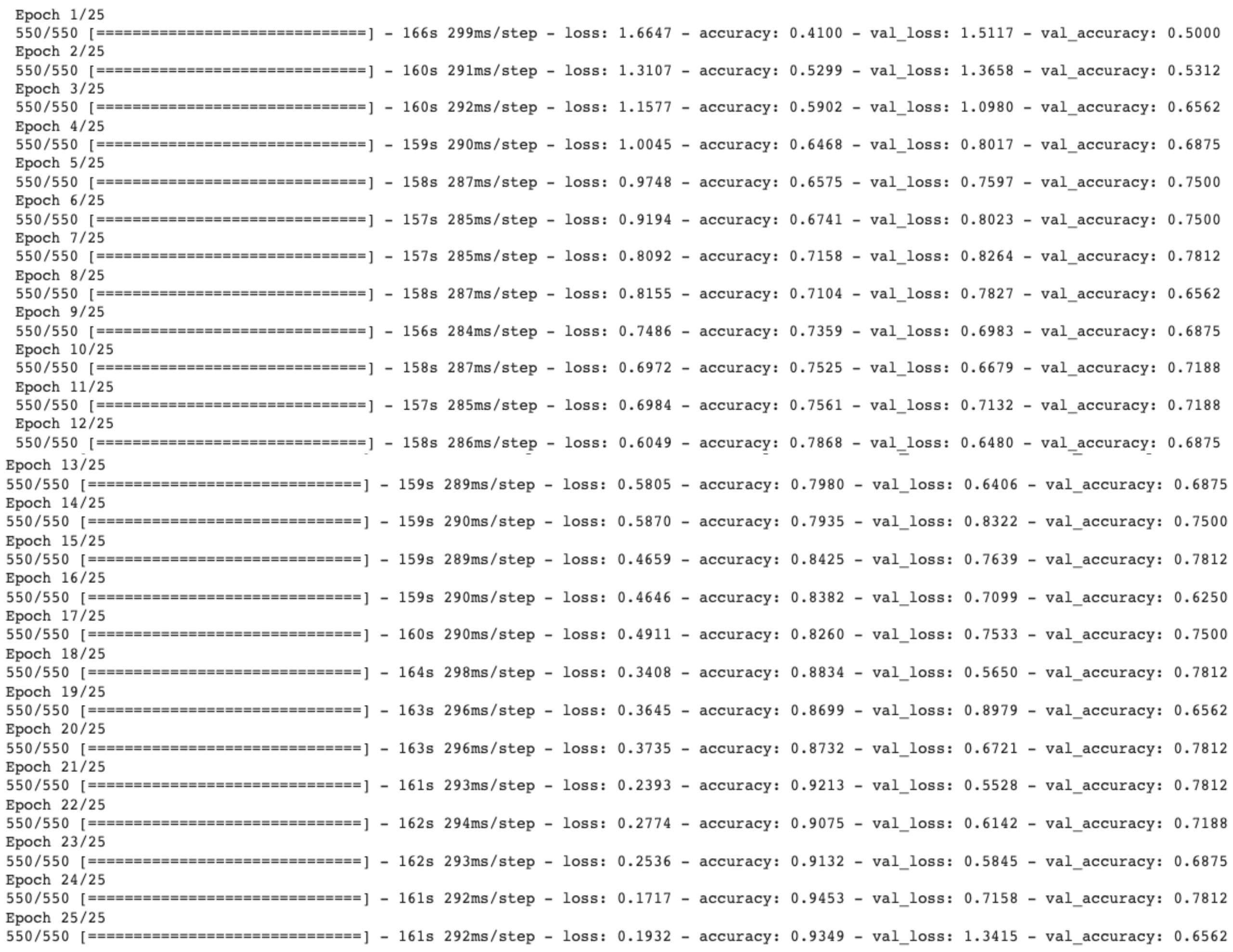
Fig 4: ResNet50 architecture

Parameters:

The following parameters were set for both the models.

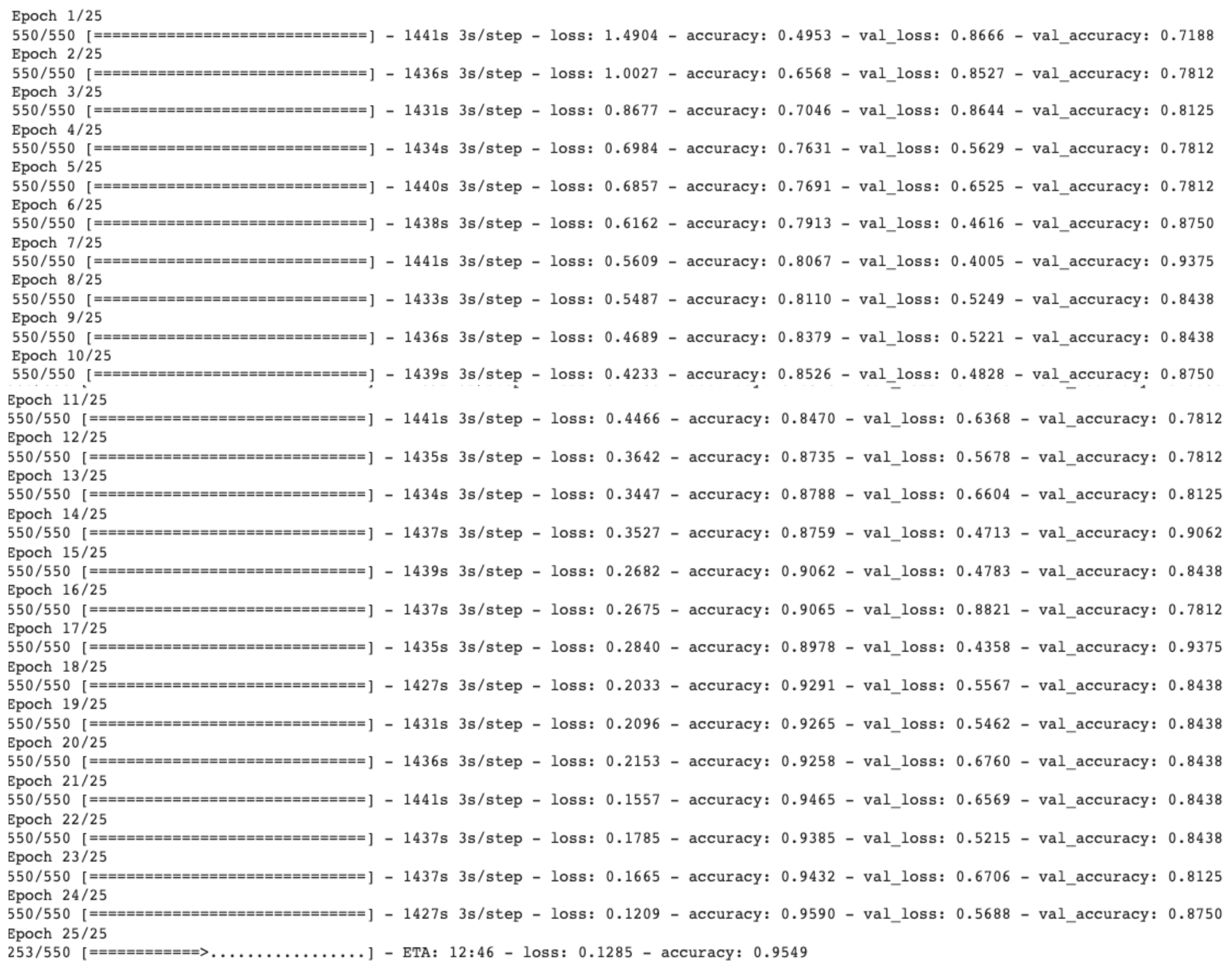
Parameter	Value
Batch Size	32
Steps per epoch	550
Epoch	25
Validation Steps	1
Optimiser	SGD

Training the model:



```
Epoch 1/25
550/550 [==============================] - 166s 299ms/step - loss: 1.6647 - accuracy: 0.4100 - val_loss: 1.5117 - val_accuracy: 0.5000
Epoch 2/25
550/550 [==============================] - 160s 291ms/step - loss: 1.3107 - accuracy: 0.5299 - val_loss: 1.3658 - val_accuracy: 0.5312
Epoch 3/25
550/550 [==============================] - 160s 292ms/step - loss: 1.1577 - accuracy: 0.5902 - val_loss: 1.0980 - val_accuracy: 0.6562
Epoch 4/25
550/550 [==============================] - 159s 290ms/step - loss: 1.0045 - accuracy: 0.6468 - val_loss: 0.8017 - val_accuracy: 0.6875
Epoch 5/25
550/550 [==============================] - 158s 287ms/step - loss: 0.9748 - accuracy: 0.6575 - val_loss: 0.7597 - val_accuracy: 0.7500
Epoch 6/25
550/550 [==============================] - 157s 285ms/step - loss: 0.9194 - accuracy: 0.6741 - val_loss: 0.8023 - val_accuracy: 0.7500
Epoch 7/25
550/550 [==============================] - 157s 285ms/step - loss: 0.8092 - accuracy: 0.7158 - val_loss: 0.8264 - val_accuracy: 0.7812
Epoch 8/25
550/550 [==============================] - 158s 287ms/step - loss: 0.8155 - accuracy: 0.7104 - val_loss: 0.7827 - val_accuracy: 0.6562
Epoch 9/25
550/550 [==============================] - 156s 284ms/step - loss: 0.7486 - accuracy: 0.7359 - val_loss: 0.6983 - val_accuracy: 0.6875
Epoch 10/25
550/550 [==============================] - 158s 287ms/step - loss: 0.6972 - accuracy: 0.7525 - val_loss: 0.6679 - val_accuracy: 0.7188
Epoch 11/25
550/550 [==============================] - 157s 285ms/step - loss: 0.6984 - accuracy: 0.7561 - val_loss: 0.7132 - val_accuracy: 0.7188
Epoch 12/25
550/550 [==============================] - 158s 286ms/step - loss: 0.6049 - accuracy: 0.7868 - val_loss: 0.6480 - val_accuracy: 0.6875
Epoch 13/25
550/550 [==============================] - 159s 289ms/step - loss: 0.5805 - accuracy: 0.7980 - val_loss: 0.6406 - val_accuracy: 0.6875
Epoch 14/25
550/550 [==============================] - 159s 290ms/step - loss: 0.5870 - accuracy: 0.7935 - val_loss: 0.8322 - val_accuracy: 0.7500
Epoch 15/25
550/550 [==============================] - 159s 289ms/step - loss: 0.4659 - accuracy: 0.8425 - val_loss: 0.7639 - val_accuracy: 0.7812
Epoch 16/25
550/550 [==============================] - 159s 290ms/step - loss: 0.4646 - accuracy: 0.8382 - val_loss: 0.7099 - val_accuracy: 0.6250
Epoch 17/25
550/550 [==============================] - 160s 290ms/step - loss: 0.4911 - accuracy: 0.8260 - val_loss: 0.7533 - val_accuracy: 0.7500
Epoch 18/25
550/550 [==============================] - 164s 298ms/step - loss: 0.3408 - accuracy: 0.8834 - val_loss: 0.5650 - val_accuracy: 0.7812
Epoch 19/25
550/550 [==============================] - 163s 296ms/step - loss: 0.3645 - accuracy: 0.8699 - val_loss: 0.8979 - val_accuracy: 0.6562
Epoch 20/25
550/550 [==============================] - 163s 296ms/step - loss: 0.3735 - accuracy: 0.8732 - val_loss: 0.6721 - val_accuracy: 0.7812
Epoch 21/25
550/550 [==============================] - 161s 293ms/step - loss: 0.2393 - accuracy: 0.9213 - val_loss: 0.5528 - val_accuracy: 0.7812
Epoch 22/25
550/550 [==============================] - 162s 294ms/step - loss: 0.2774 - accuracy: 0.9075 - val_loss: 0.6142 - val_accuracy: 0.7188
Epoch 23/25
550/550 [==============================] - 162s 293ms/step - loss: 0.2536 - accuracy: 0.9132 - val_loss: 0.5845 - val_accuracy: 0.6875
Epoch 24/25
550/550 [==============================] - 161s 292ms/step - loss: 0.1717 - accuracy: 0.9453 - val_loss: 0.7158 - val_accuracy: 0.7812
Epoch 25/25
550/550 [==============================] - 161s 292ms/step - loss: 0.1932 - accuracy: 0.9349 - val_loss: 1.3415 - val_accuracy: 0.6562
```

Fig 5: SimCNN Training



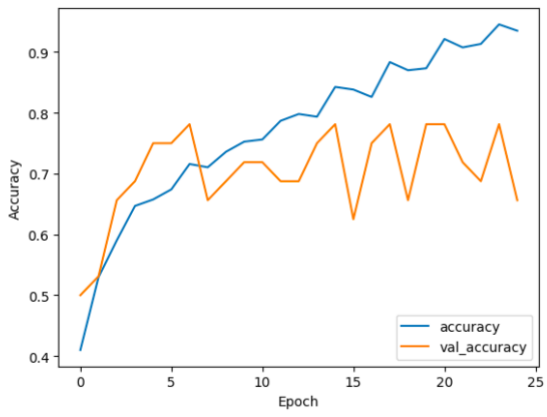
```
Epoch 1/25
550/550 [==============================] - 1441s 3s/step - loss: 1.4904 - accuracy: 0.4953 - val_loss: 0.8666 - val_accuracy: 0.7188
Epoch 2/25
550/550 [==============================] - 1436s 3s/step - loss: 1.0027 - accuracy: 0.6568 - val_loss: 0.8527 - val_accuracy: 0.7812
Epoch 3/25
550/550 [==============================] - 1431s 3s/step - loss: 0.8677 - accuracy: 0.7046 - val_loss: 0.8644 - val_accuracy: 0.8125
Epoch 4/25
550/550 [==============================] - 1434s 3s/step - loss: 0.6984 - accuracy: 0.7631 - val_loss: 0.5629 - val_accuracy: 0.7812
Epoch 5/25
550/550 [==============================] - 1440s 3s/step - loss: 0.6857 - accuracy: 0.7691 - val_loss: 0.6525 - val_accuracy: 0.7812
Epoch 6/25
550/550 [==============================] - 1438s 3s/step - loss: 0.6162 - accuracy: 0.7913 - val_loss: 0.4616 - val_accuracy: 0.8750
Epoch 7/25
550/550 [==============================] - 1441s 3s/step - loss: 0.5609 - accuracy: 0.8067 - val_loss: 0.4005 - val_accuracy: 0.9375
Epoch 8/25
550/550 [==============================] - 1433s 3s/step - loss: 0.5487 - accuracy: 0.8110 - val_loss: 0.5249 - val_accuracy: 0.8438
Epoch 9/25
550/550 [==============================] - 1436s 3s/step - loss: 0.4689 - accuracy: 0.8379 - val_loss: 0.5221 - val_accuracy: 0.8438
Epoch 10/25
550/550 [==============================] - 1439s 3s/step - loss: 0.4233 - accuracy: 0.8526 - val_loss: 0.4828 - val_accuracy: 0.8750
Epoch 11/25
550/550 [==============================] - 1441s 3s/step - loss: 0.4466 - accuracy: 0.8470 - val_loss: 0.6368 - val_accuracy: 0.7812
Epoch 12/25
550/550 [==============================] - 1435s 3s/step - loss: 0.3642 - accuracy: 0.8735 - val_loss: 0.5678 - val_accuracy: 0.7812
Epoch 13/25
550/550 [==============================] - 1434s 3s/step - loss: 0.3447 - accuracy: 0.8788 - val_loss: 0.6604 - val_accuracy: 0.8125
Epoch 14/25
550/550 [==============================] - 1437s 3s/step - loss: 0.3527 - accuracy: 0.8759 - val_loss: 0.4713 - val_accuracy: 0.9062
Epoch 15/25
550/550 [==============================] - 1439s 3s/step - loss: 0.2682 - accuracy: 0.9062 - val_loss: 0.4783 - val_accuracy: 0.8438
Epoch 16/25
550/550 [==============================] - 1437s 3s/step - loss: 0.2675 - accuracy: 0.9065 - val_loss: 0.8821 - val_accuracy: 0.7812
Epoch 17/25
550/550 [==============================] - 1435s 3s/step - loss: 0.2840 - accuracy: 0.8978 - val_loss: 0.4358 - val_accuracy: 0.9375
Epoch 18/25
550/550 [==============================] - 1427s 3s/step - loss: 0.2033 - accuracy: 0.9291 - val_loss: 0.5567 - val_accuracy: 0.8438
Epoch 19/25
550/550 [==============================] - 1431s 3s/step - loss: 0.2096 - accuracy: 0.9265 - val_loss: 0.5462 - val_accuracy: 0.8438
Epoch 20/25
550/550 [==============================] - 1436s 3s/step - loss: 0.2153 - accuracy: 0.9258 - val_loss: 0.6760 - val_accuracy: 0.8438
Epoch 21/25
550/550 [==============================] - 1441s 3s/step - loss: 0.1557 - accuracy: 0.9465 - val_loss: 0.6569 - val_accuracy: 0.8438
Epoch 22/25
550/550 [==============================] - 1437s 3s/step - loss: 0.1785 - accuracy: 0.9385 - val_loss: 0.5215 - val_accuracy: 0.8438
Epoch 23/25
550/550 [==============================] - 1437s 3s/step - loss: 0.1665 - accuracy: 0.9432 - val_loss: 0.6706 - val_accuracy: 0.8125
Epoch 24/25
550/550 [==============================] - 1427s 3s/step - loss: 0.1209 - accuracy: 0.9590 - val_loss: 0.5688 - val_accuracy: 0.8750
Epoch 25/25
253/550 [============>.................] - ETA: 12:46 - loss: 0.1285 - accuracy: 0.9549
```

Fig 6: ResNet 50 model training

The results indicate that the SimCNN model exhibits a training accuracy of 41% in the initial iteration, which progressively improves to reach 93.49% at the 25th iteration. Despite the lack of uniformity in the trajectory of this curve, it does not exhibit any significant fluctuations. Conversely, the validation accuracy steadily increases from 50% to 78% in the first seven iterations, after which it oscillates between 62% and 78%.

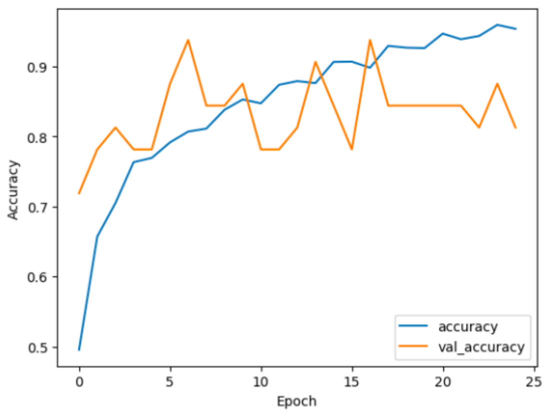
The ResNet50 model exhibits a training accuracy range of 50% to 95.49%. In contrast, the validation accuracy commences at 71% and experiences fluctuations throughout the 25 iterations, peaking at 93.75%. The model failed to complete all 550 steps in the final iteration, and the training process required approximately 11 hours to complete. Rather than restarting the training process, the model's testing efficiency was evaluated.

<matplotlib.legend.Legend at 0x7fd767fa2880>



(a)

<matplotlib.legend.Legend at 0x7fb170b36820>

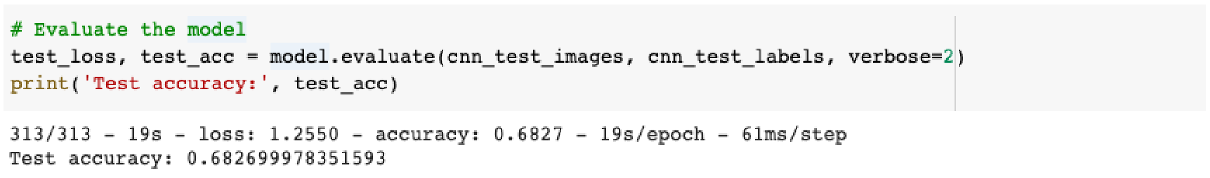


(b)

Fig 7: (a) Accuracy Distribution of SimCNN model (b) Accuracy Distribution of ResNet50 model.

Test Accuracy:

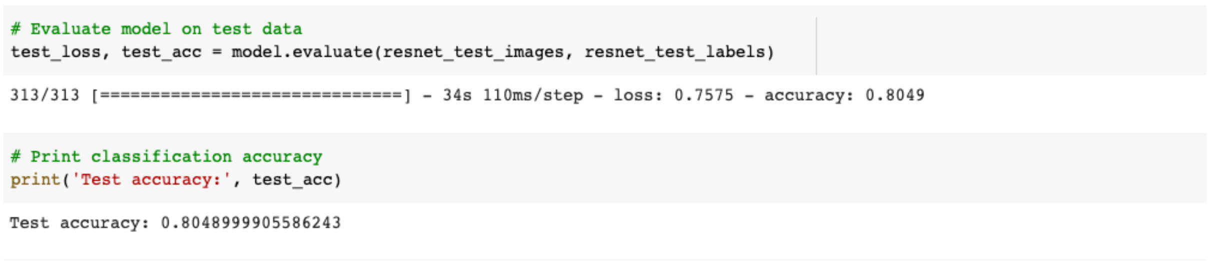
The performance of the SimCNN network and the ResNet50 model was evaluated by assessing their respective test accuracy. The SimCNN network yielded a test accuracy of 68.26%, whereas the ResNet 50 model exhibited a higher accuracy of 80.48%. As such, the ResNet50 model demonstrated superior performance on this particular dataset.



```
# Evaluate the model
test_loss, test_acc = model.evaluate(cnn_test_images, cnn_test_labels, verbose=2)
print('Test accuracy:', test_acc)
```

```
313/313 - 19s - loss: 1.2550 - accuracy: 0.6827 - 19s/epoch - 61ms/step
Test accuracy: 0.6826999783515 93
```

Fig 8: Test Accuracy of SimCNN model



```
# Evaluate model on test data
test_loss, test_acc = model.evaluate(resnet_test_images, resnet_test_labels)
```

```
313/313 [==============================] - 34s 110ms/step - loss: 0.7575 - accuracy: 0.8049
```

```
# Print classification accuracy
print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.8048999905586243
```

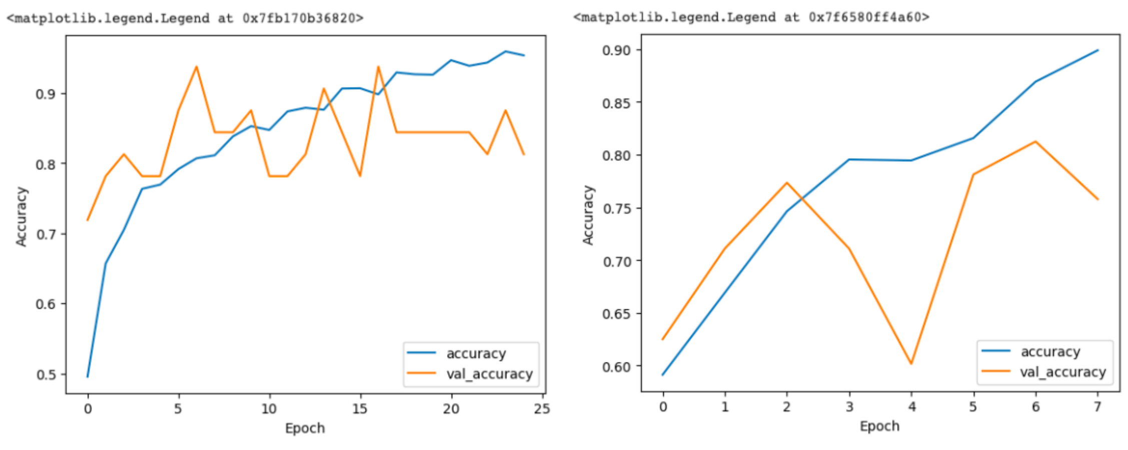
Fig 9: Test Accuracy of SimCNN model

<u>Comparing hyperparameters of ResNet50 Model:</u>

For the second part of the project, the hyperparameters for the ResNet50 model were modified to double their original values. Given that each epoch required approximately one hour of computation, the number of epochs was reduced to ten. However, due to the limited input data, the model could only complete seven epochs.

Parameter	Initial Values	Updated Values
Batch Size	32	64
Steps per epoch	550	1000
Epoch	25	10
Validation Steps	1	2
Optimiser	SGD	ADAM

Upon implementing the modified hyperparameters, the ResNet50 model exhibited an initial training accuracy of 59% and a validation accuracy of 62%. The model demonstrated a steady increase in training accuracy throughout training, eventually reaching a value of 86.93%. Similarly, the validation accuracy exhibited an upward trend, with a peak of 81.25%, briefly decreasing only once to 60.16% during the fifth iteration.



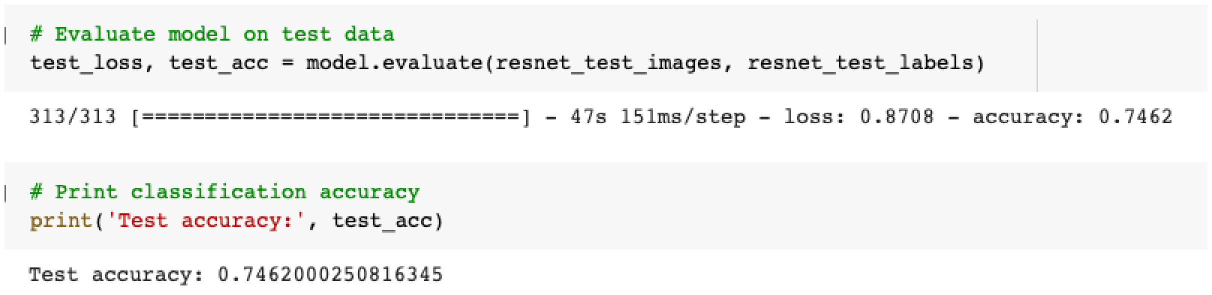
```
<matplotlib.legend.Legend at 0x7fb170b36820>
```

```
<matplotlib.legend.Legend at 0x7f6580ff4a60>
```

(a) (b)

Fig 10: (a) Accuracy Distribution of ResNet50 Model with intital hyerparameters (b)Accuracy Distribution of ResNet50 Model with modified hyperparameters.

The ResNet50 model with the updated hyperparameters yielded a test accuracy of 74%, which was lower than the original hyperparameters. It is essential to consider that the dataset or generator must be capable of producing at least as many batches as the product of steps_per_epoch and epochs. To identify the optimal hyperparameters, methods such as grid search are more reliable. However, this technique can be resource-intensive and time-consuming.



```
# Evaluate model on test data
test_loss, test_acc = model.evaluate(resnet_test_images, resnet_test_labels)
```

```
313/313 [==============================] - 47s 151ms/step - loss: 0.8708 - accuracy: 0.7462
```

```
# Print classification accuracy
print('Test accuracy:', test_acc)
```

```
Test accuracy: 0.7462000250816345
```

Fig 11: Test accuracy of ResNet50 Model with updated hyperparameters