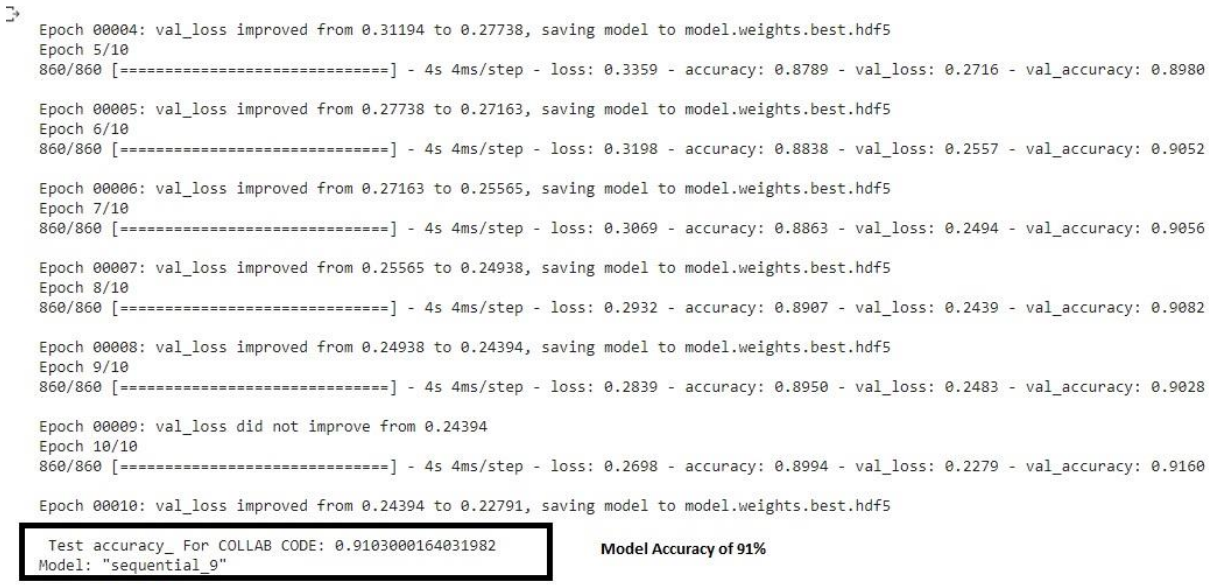
HW5 – Report

Achyuthan Unni Krishnan, Kishor Sabarish

1. Implementing the Neural Network from Blog Post

Upon implementing the network from the blog post we were able to achieve an accuracy of 91.03%. Training was done for 10 epochs.



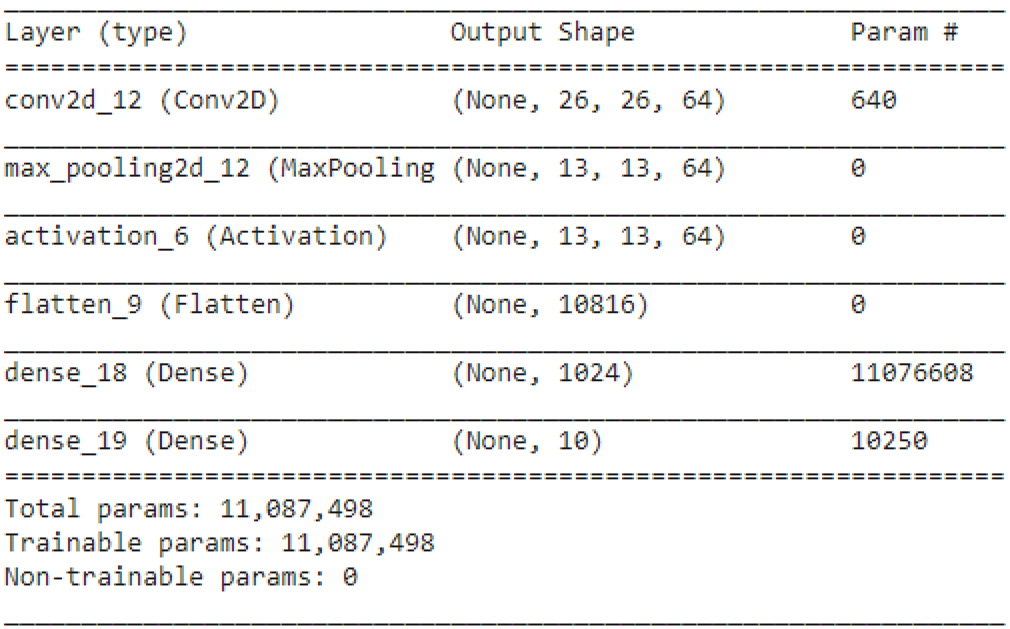
```
Epoch 00004: val_loss improved from 0.31194 to 0.27738, saving model to model.weights.best.hdf5
Epoch 5/10
860/860 [==============================] - 4s 4ms/step - loss: 0.3359 - accuracy: 0.8789 - val_loss: 0.2716 - val_accuracy: 0.8980

Epoch 00005: val_loss improved from 0.27738 to 0.27163, saving model to model.weights.best.hdf5
Epoch 6/10
860/860 [==============================] - 4s 4ms/step - loss: 0.3198 - accuracy: 0.8838 - val_loss: 0.2557 - val_accuracy: 0.9052

Epoch 00006: val_loss improved from 0.27163 to 0.25565, saving model to model.weights.best.hdf5
Epoch 7/10
860/860 [==============================] - 4s 4ms/step - loss: 0.3069 - accuracy: 0.8863 - val_loss: 0.2494 - val_accuracy: 0.9056

Epoch 00007: val_loss improved from 0.25565 to 0.24938, saving model to model.weights.best.hdf5
Epoch 8/10
860/860 [==============================] - 4s 4ms/step - loss: 0.2932 - accuracy: 0.8907 - val_loss: 0.2439 - val_accuracy: 0.9082

Epoch 00008: val_loss improved from 0.24938 to 0.24394, saving model to model.weights.best.hdf5
Epoch 9/10
860/860 [==============================] - 4s 4ms/step - loss: 0.2839 - accuracy: 0.8950 - val_loss: 0.2483 - val_accuracy: 0.9028

Epoch 00009: val_loss did not improve from 0.24394
Epoch 10/10
860/860 [==============================] - 4s 4ms/step - loss: 0.2698 - accuracy: 0.8994 - val_loss: 0.2279 - val_accuracy: 0.9160

Epoch 00010: val_loss improved from 0.24394 to 0.22791, saving model to model.weights.best.hdf5
 Test accuracy_ For COLLAB CODE: 0.9103000164031982
Model: "sequential_9"
```

Model Accuracy of 91%

Fig 1: The losses for the training of the model presented in the blog post.

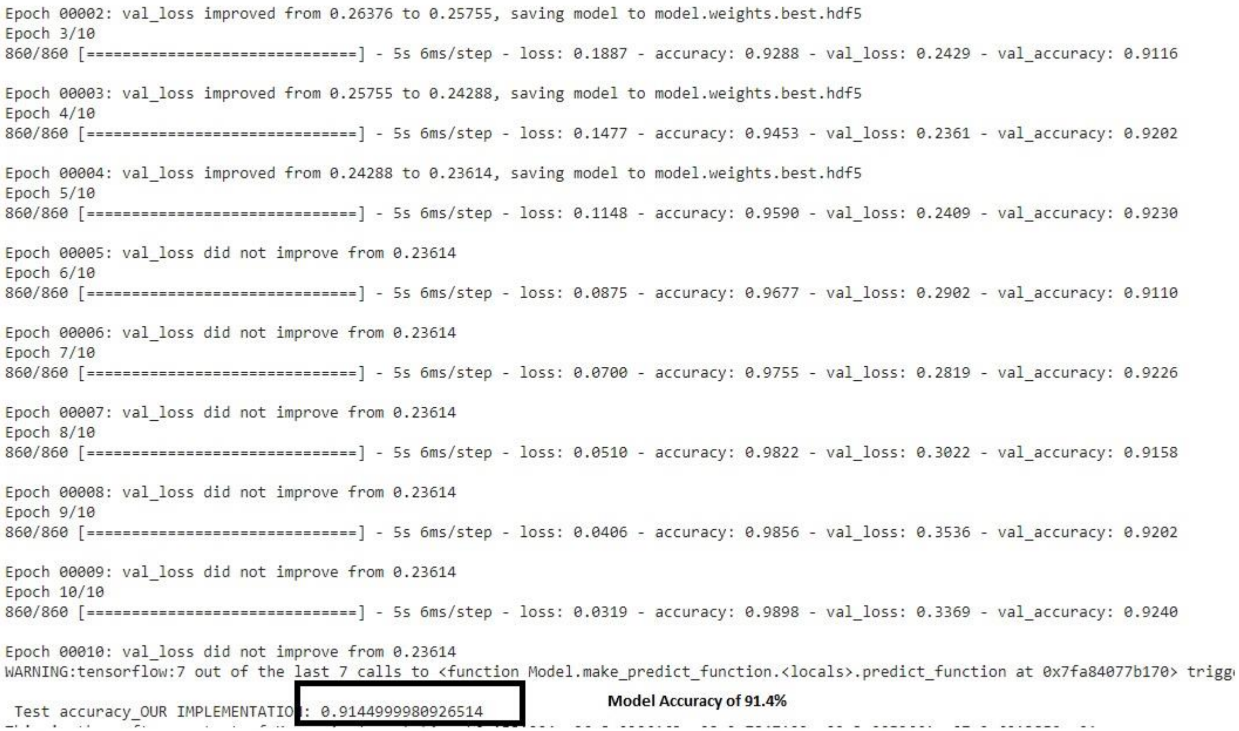
2. Representing A CNN as a Fully-Connected Neural Network

The model as required in the question was implemented. The model summary for the same is presented in the screenshot below. The accuracy was found to be 91.44% when trained for 10 epochs.



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_12 (Conv2D)           (None, 26, 26, 64)        640       
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 13, 13, 64)        0         
_________________________________________________________________
activation_6 (Activation)    (None, 13, 13, 64)        0         
_________________________________________________________________
flatten_9 (Flatten)          (None, 10816)             0         
_________________________________________________________________
dense_18 (Dense)             (None, 1024)              11076608  
_________________________________________________________________
dense_19 (Dense)             (None, 10)                10250     
=================================================================
Total params: 11,087,498
Trainable params: 11,087,498
Non-trainable params: 0
_________________________________________________________________
```

Fig 2: Model Summary of the model required in question 2.



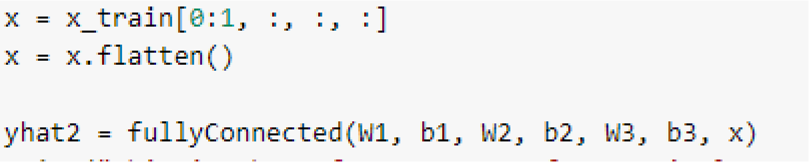
```
Epoch 00002: val_loss improved from 0.26376 to 0.25755, saving model to model.weights.best.hdf5
Epoch 3/10
860/860 [==============================] - 5s 6ms/step - loss: 0.1887 - accuracy: 0.9288 - val_loss: 0.2429 - val_accuracy: 0.9116

Epoch 00003: val_loss improved from 0.25755 to 0.24288, saving model to model.weights.best.hdf5
Epoch 4/10
860/860 [==============================] - 5s 6ms/step - loss: 0.1477 - accuracy: 0.9453 - val_loss: 0.2361 - val_accuracy: 0.9202

Epoch 00004: val_loss improved from 0.24288 to 0.23614, saving model to model.weights.best.hdf5
Epoch 5/10
860/860 [==============================] - 5s 6ms/step - loss: 0.1148 - accuracy: 0.9590 - val_loss: 0.2409 - val_accuracy: 0.9230

Epoch 00005: val_loss did not improve from 0.23614
Epoch 6/10
860/860 [==============================] - 5s 6ms/step - loss: 0.0875 - accuracy: 0.9677 - val_loss: 0.2902 - val_accuracy: 0.9110

Epoch 00006: val_loss did not improve from 0.23614
Epoch 7/10
860/860 [==============================] - 5s 6ms/step - loss: 0.0700 - accuracy: 0.9755 - val_loss: 0.2819 - val_accuracy: 0.9226

Epoch 00007: val_loss did not improve from 0.23614
Epoch 8/10
860/860 [==============================] - 5s 6ms/step - loss: 0.0510 - accuracy: 0.9822 - val_loss: 0.3022 - val_accuracy: 0.9158

Epoch 00008: val_loss did not improve from 0.23614
Epoch 9/10
860/860 [==============================] - 5s 6ms/step - loss: 0.0406 - accuracy: 0.9856 - val_loss: 0.3536 - val_accuracy: 0.9202

Epoch 00009: val_loss did not improve from 0.23614
Epoch 10/10
860/860 [==============================] - 5s 6ms/step - loss: 0.0319 - accuracy: 0.9898 - val_loss: 0.3369 - val_accuracy: 0.9240

Epoch 00010: val_loss did not improve from 0.23614
WARNING:tensorflow:7 out of the last 7 calls to <function Model.make_predict_function.<locals>.predict_function at 0x7fa84077b170> trigg

 Test accuracy_OUR IMPLEMENTATIO: 0.9144999980926514
```

Model Accuracy of 91.4%

Fig 3: The losses for the training of the model required in question 2.

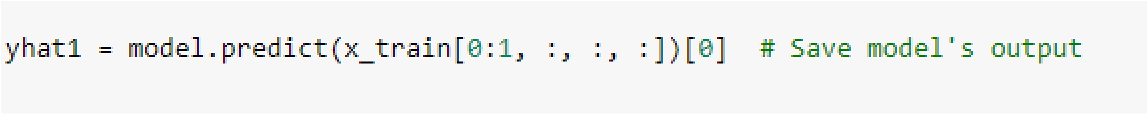
2.1 Transfer of Weights

The transfer of weights from the CNN from Keras to Fully-Connected Neural Network was successful as the softmax values for both the outputs were almost the same as seen in the screen shot below. The sample image was the first image from the training set which was extracted as seen in the screenshots of the code from our submission below.



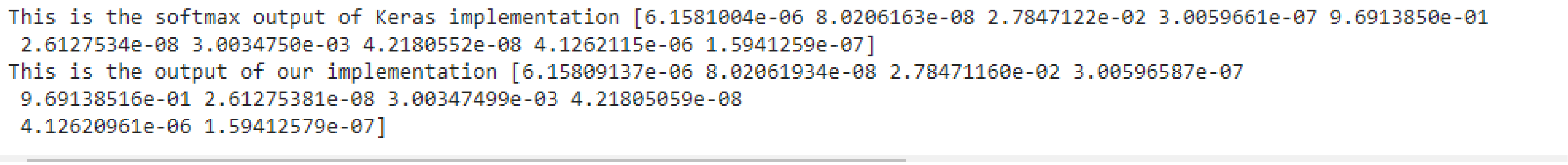
```
x = x_train[0:1, :, :, :]
x = x.flatten()

yhat2 = fullyConnected(W1, b1, W2, b2, W3, b3, x)
```

Fig 4: The first image fed as input to our Fully Connected Implementation.



```
yhat1 = model.predict(x_train[0:1, :, :, :])[0]  # Save model's output
```

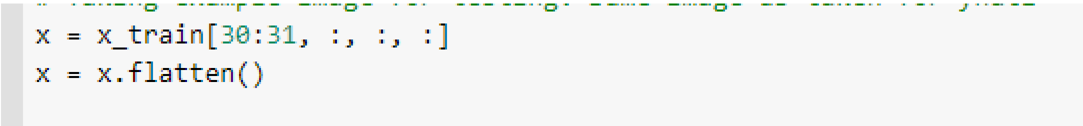
Fig 5: The first image fed as input to our Keras Implementation.



```
This is the softmax output of Keras implementation [6.1581004e-06 8.0206163e-08 2.7847122e-02 3.0059661e-07 9.6913850e-01
 2.6127534e-08 3.0034750e-03 4.2180552e-08 4.1262115e-06 1.5941259e-07]
This is the output of our implementation [6.15809137e-06 8.02061934e-08 2.78471160e-02 3.00596587e-07
 9.69138516e-01 2.61275381e-08 3.00347499e-03 4.21805059e-08
 4.12620961e-06 1.59412579e-07]
```

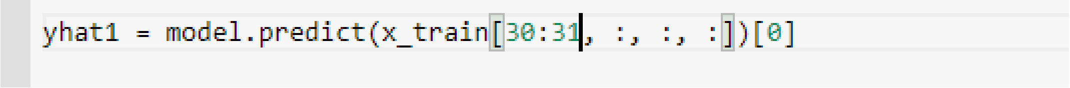
Fig 6: The first softmax output is for the Keras implementation and the second output is for the FCNN implementation.

Below is a demonstration for the 30th example from our training set to show that our implementation is indeed robust.



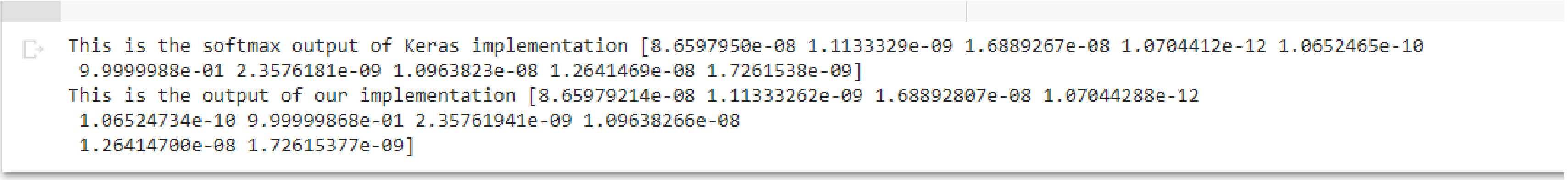
```
x = x_train[30:31, :, :, :]
x = x.flatten()
```

Fig 7: The 30th image fed as input to our Fully Connected Implementation.



```
yhat1 = model.predict(x_train[30:31, :, :, :])[0]
```

Fig 8: The 30th image fed as input to our Keras Implementation.



```
This is the softmax output of Keras implementation [8.6597950e-08 1.1133329e-09 1.6889267e-08 1.0704412e-12 1.0652465e-10
 9.9999988e-01 2.3576181e-09 1.0963823e-08 1.2641469e-08 1.7261538e-09]
This is the output of our implementation [8.65979214e-08 1.11333262e-09 1.68892807e-08 1.07044288e-12
 1.06524734e-10 9.99999868e-01 2.35761941e-09 1.09638266e-08
 1.26414700e-08 1.72615377e-09]
```

Fig 9: The first softmax output is for the Keras implementation and the second output is for the FCNN implementation.