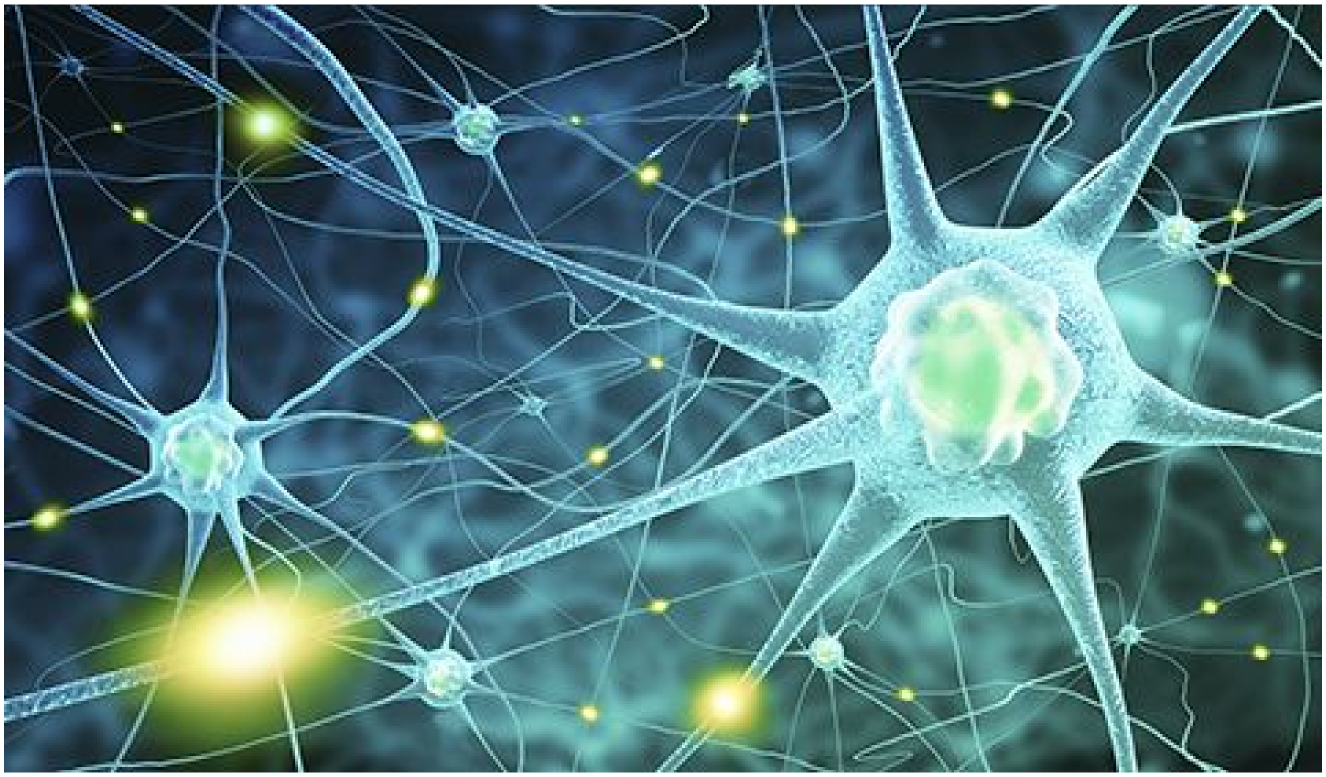
Assignment 3

Computer Vision - CS512



Akshay R

A20442409

Problem Statement:

1. Construct and train CNN:

(a) Load the MNIST set and split it into training/validation/testing subsets of 55,000/10,000/5,000 examples respectively.

(b) Convert the digit labels into odd/even labels and discard the original digit labels. You may not use the original digit labels for training.

(c) Construct a network with two convolution layers with pooling, a dropout layer, and two fully connected layers. It is up to you to select the number of units in each layer.

(d) Select the appropriate loss function, optimization algorithm (and its parameters), and evaluation metric.

(e) Train the network and record the training and validation loss and accuracy measures.

(f) Plot the training and validation loss as a function of epochs. Plot the accuracy as a function of epochs. In addition, report the loss and accuracy values of the final training step.

2. Hyper-parameter Tuning:

Evaluate different variations of the basic network as described below and measure performance. In your report, discuss your variations, compare the results you obtain and attempt to draw conclusions.

Evaluate at least the following aspects:

(a) Changing the network architecture (e.g. number of layers and/or organization of layers).

(b) Changing the receptive field and stride parameters.

(c) Changing optimizer and loss function (e.g. Adam optimizer).

(d) Changing various parameters (e.g. dropout, learning rate, number of filters, number of epochs).

(e) Adding batch and layer normalization.

(f) Using different weight initializers (e.g. Xavier, He).

(g) Evaluate the best validation model on the testing subset and report the results.

3. Inference:

Write a program to use your pretrained custom CNN. The program should do the following: (a) Accept as input an image of a handwritten digit. Assume each image contains one digit.

(b) Using OpenCV do some basic image preprocessing to prepare the image for your CNN: Resize the image to fit your model's image size requirement; Transform the grayscale image to a binary image (consider using the GaussianBlur() and adaptiveThreshold(), or any other type of binary thresholding that performs well); Display the original image and binary image in two separate windows.

(c) Using your CNN classify the binary image (even/odd).

Proposed Solution:

Constructing and Hyperparameter tuning is executed first and each model is saved. The model with the best test accuracy and loss is then loaded to test the classification.

Implementation and methodology:

MNIST data set is first loaded from keras.datasets library. The obtained images are split into 60000 training images and 10000 testing images by default. To obtain a custom split, I first merged training and testing dataset into a single numpy array and then split it into 55000 training 10000 validation and 5000 testing images using the sklearn.train_test_split function. By default we obtain 10 labels for 10 digits which is converted into odd or even labels by using a simple if else statement and checking if the number is divisible by zero.

The obtained data set is standardized by subtracting the mean and dividing the standard deviation. The obtained datasets for training will be of the shape (x, 28, 28). This is then converted to (x, 28, 28, 1) to use in the keras architecture.

A CNN model is then built using the keras.sequential models. The two CNN layers have a filter of size (5, 5) and have 32 and 64 filters. The model is compiled with a loss of categorical cross entropy and a RMSprop optimizer. This model is then fit for 10 epochs to obtain the training validation and test accuracy and loss. Graphs are plotted to show Validation accuracy and loss, Training accuracy and loss vs epochs.

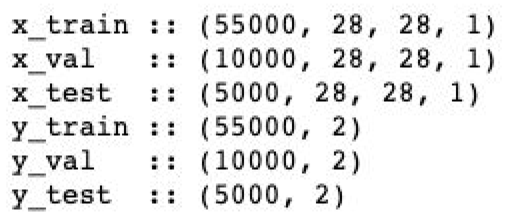
To tune the hyperparameters the model is altered in the following ways:

- The network architecture is first changed to have 3 layers instead of two, with filter size (3, 3) and with 32, 32, 32 number of filters. The model accuracy and loss are plotted and recorded.
- The basic model is then changed by using a filter of size (3, 3) and a stride of 2. The model accuracy and loss are plotted and recorded.
- The basic model is now changed by using Adam optimizer and a categorical hinge loss. The model accuracy and loss are plotted and recorded.
- The basic model is now changed by increasing the number of epochs to 20 and reducing the learning rate to 1e-6. The model accuracy and loss are plotted and recorded.
- The basic model is now changed by adding BatchNormalization at every layer. The model accuracy and loss are plotted and recorded.
- The basic model is now changed by initializing the weights of the fully connected network to orthogonal. The model accuracy and loss are plotted and recorded.

The best model is selected from the above custom models based on the accuracy and loss values obtained from the above experiments. This model is then loaded and provided with an odd number and an even number image to validate the classification.

Results and discussion:

Data split:



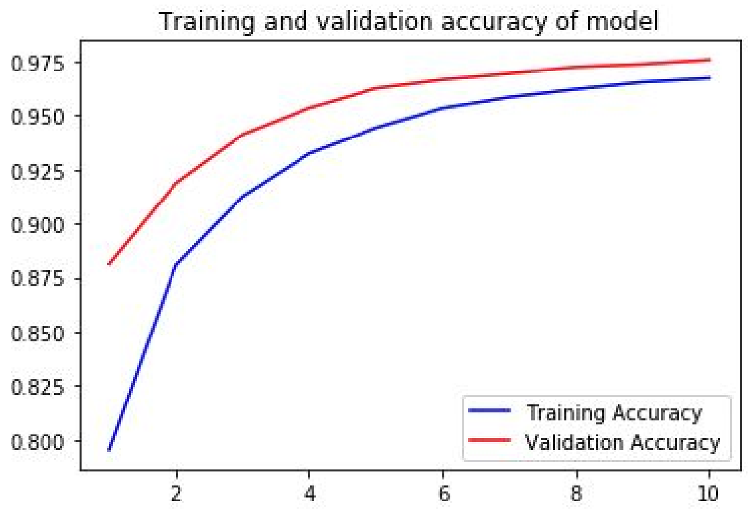
```
x_train :: (55000, 28, 28, 1)
x_val   :: (10000, 28, 28, 1)
x_test  :: (5000, 28, 28, 1)
y_train :: (55000, 2)
y_val   :: (10000, 2)
y_test  :: (5000, 2)
```

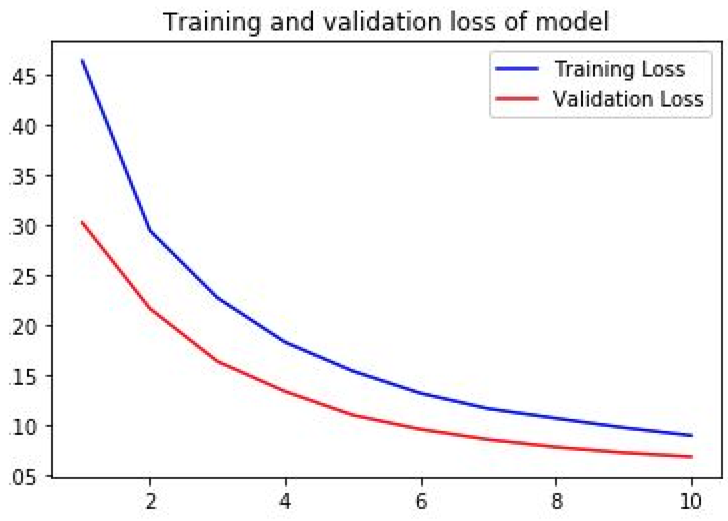
Basic model:

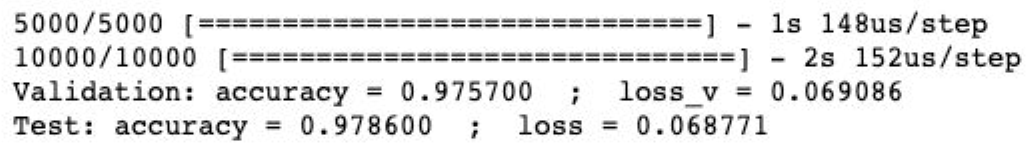
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 24, 24, 32)        832
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 4, 4, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               524800
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 1026
=================================================================
Total params: 577,922
Trainable params: 577,922
Non-trainable params: 0
```

Basic model results:

```
The accuracy of the last step while training:  0.96729094
The loss of the last step while training:  0.09027373099530285
```



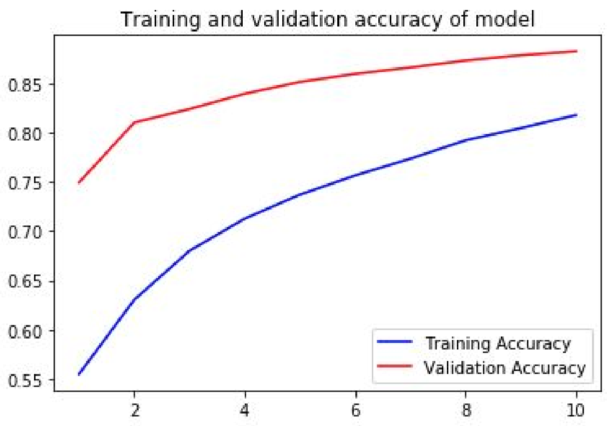


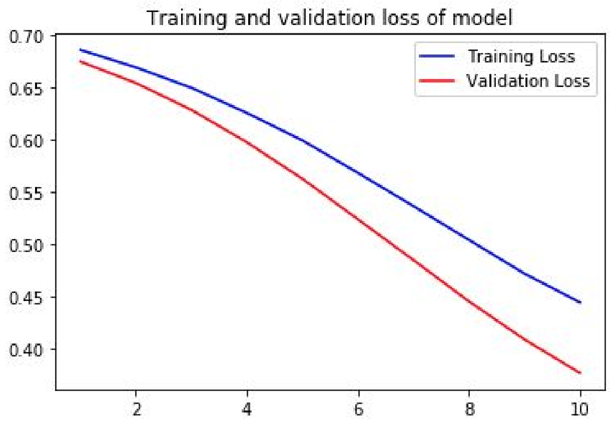


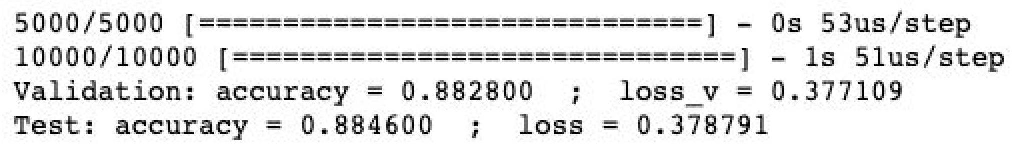
```
5000/5000 [==============================] - 1s 148us/step
10000/10000 [==============================] - 2s 152us/step
Validation: accuracy = 0.975700  ;  loss_v = 0.069086
Test: accuracy = 0.978600  ;  loss = 0.068771
```

Results:

Custom model 1:

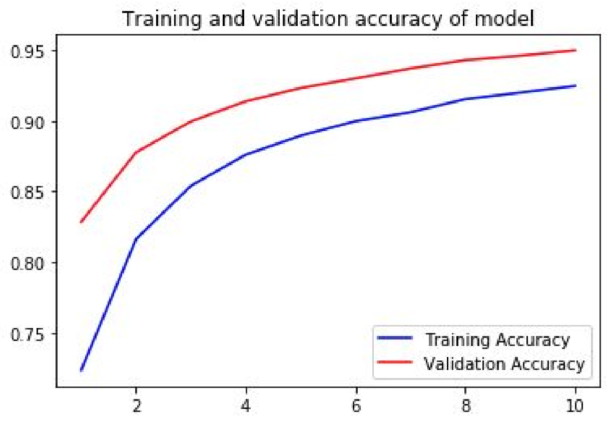


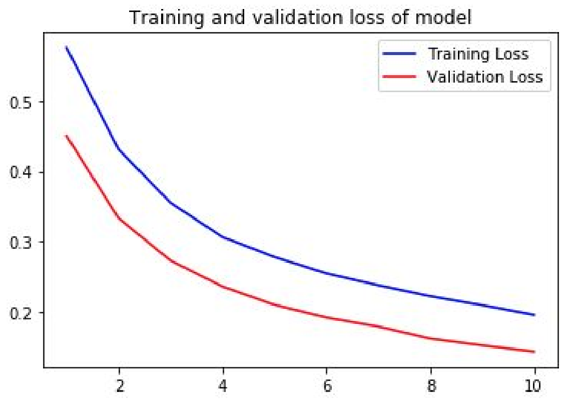


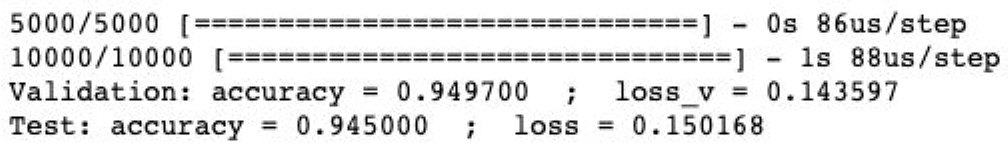


```
5000/5000 [==============================] - 0s 53us/step
10000/10000 [==============================] - 1s 51us/step
Validation: accuracy = 0.882800  ;  loss_v = 0.377109
Test: accuracy = 0.884600  ;  loss = 0.378791
```

Custom model 2:

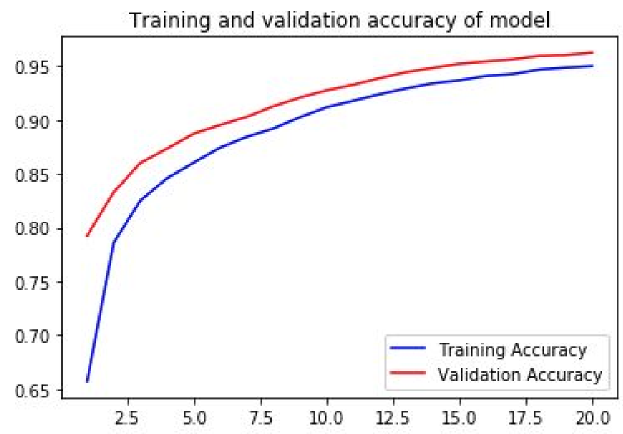


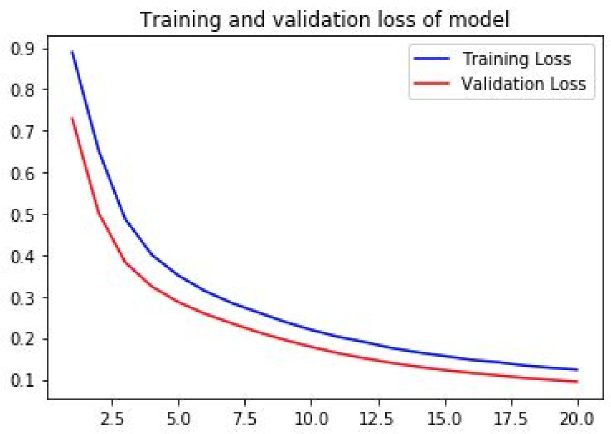


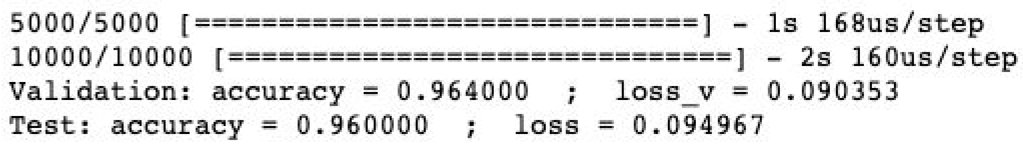


```
5000/5000 [==============================] - 0s 86us/step
10000/10000 [==============================] - 1s 88us/step
Validation: accuracy = 0.949700  ;  loss_v = 0.143597
Test: accuracy = 0.945000  ;  loss = 0.150168
```

Custom model 3:

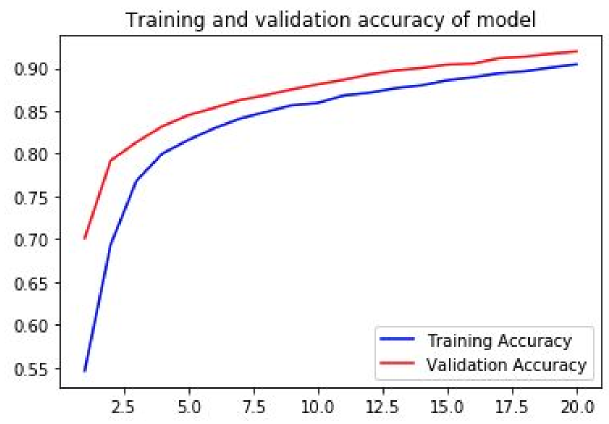


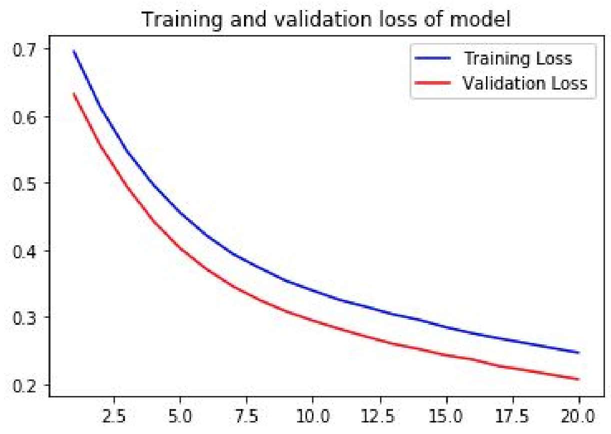


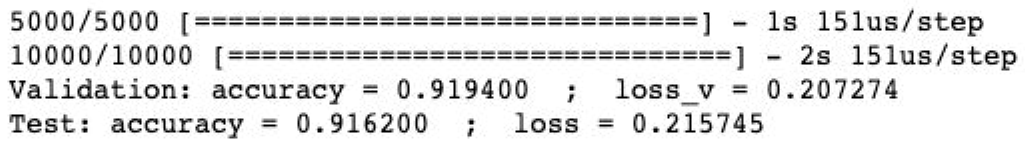


```
5000/5000 [==============================] - 1s 168us/step
10000/10000 [==============================] - 2s 160us/step
Validation: accuracy = 0.964000  ;  loss_v = 0.090353
Test: accuracy = 0.960000  ;  loss = 0.094967
```

Custom model 4:

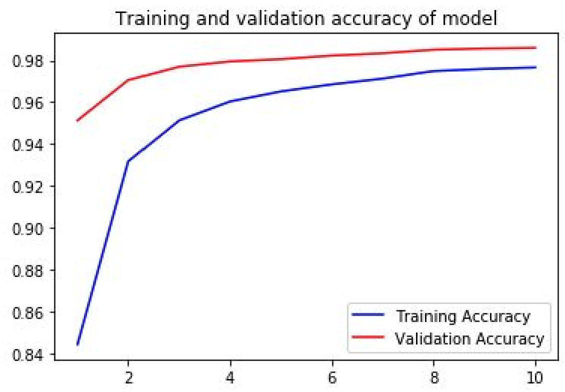


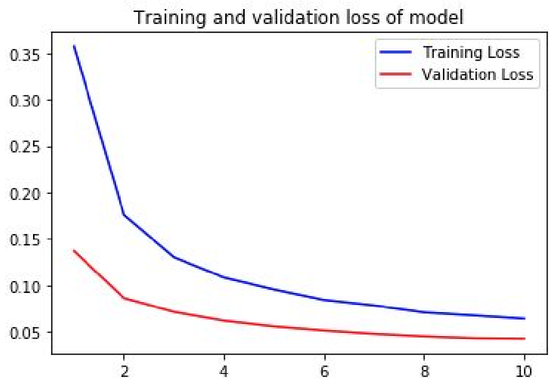


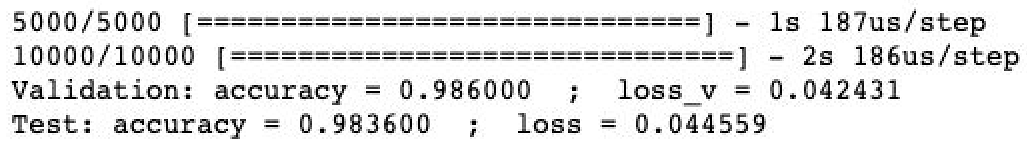


```
5000/5000 [==============================] - 1s 151us/step
10000/10000 [==============================] - 2s 151us/step
Validation: accuracy = 0.919400  ;  loss_v = 0.207274
Test: accuracy = 0.916200  ;  loss = 0.215745
```

Custom model 5:

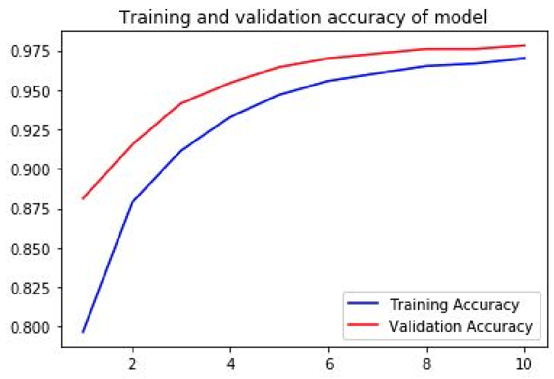


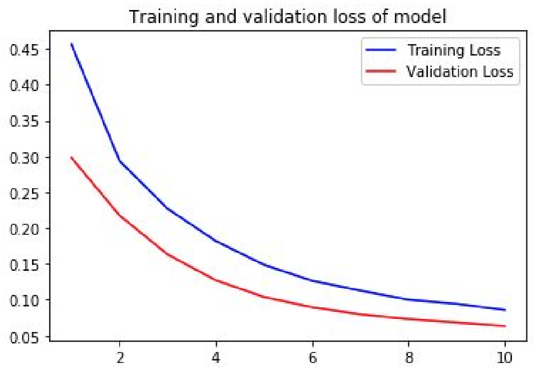


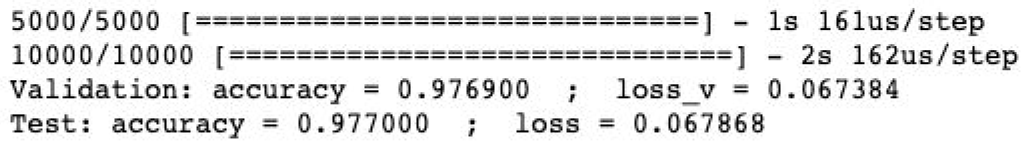


```
5000/5000 [==============================] - 1s 187us/step
10000/10000 [==============================] - 2s 186us/step
Validation: accuracy = 0.986000  ;  loss_v = 0.042431
Test: accuracy = 0.983600  ;  loss = 0.044559
```

Custom model 6:







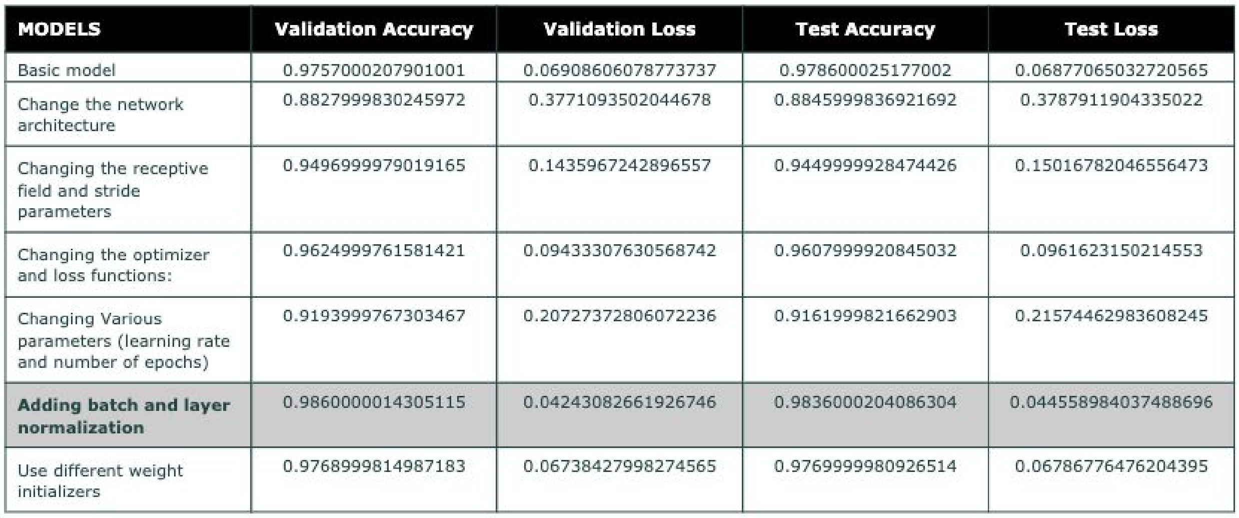
```
5000/5000 [==============================] - 1s 161us/step
10000/10000 [==============================] - 2s 162us/step
Validation: accuracy = 0.976900  ;  loss_v = 0.067384
Test: accuracy = 0.977000  ;  loss = 0.067868
```

From the above results it can be inferred that:

- Test accuracy has reduced to 94% when a third layer is added. It can be seen that the validation and test accuracy and loss are converging at a slower pace. This can be due to the reason that the network has to learn more number of parameters compared to the basic model.

- Test accuracy has reduced to 94% by changing the receptive field to (3, 3) instead of (5, 5) and a stride of 2 has been used. This can be interpreted by saying that the model has learnt less compared to the (5, 5) receptive field. There is a tradeoff in time vs accuracy, as this model runs faster compared to the basic model due to the stride and change in receptive field.
- Changing the optimizer and loss function has reduced the test accuracy to 96%. Categorical_sparse loss function was used and Adam optimizer was used which led to this result. Which implies that RMSprop and categorical_crossentropy as doing a better job.
- Changing various parameters like learning rate and number of epochs las lead to a slower convergence. A lower learning rate makes the learning slower and hence slower convergence.
- Adding batch and layer normalization resulted in an increase in the test accuracy to 98.3%. This can be interpreted as the model being more general towards data. The layer and batch normalization layers normalize increasing gradients between layers. Hence leading to more accurate learning of weights.
- Finally an orthogonal initialization of weights decreased the test accuracy to 97.6 which is nearly identical to the basic model. Orthogonal initialization resolves the effects of too small and extremely large gradients. From the results we can see that there were no such effects in the basic model.

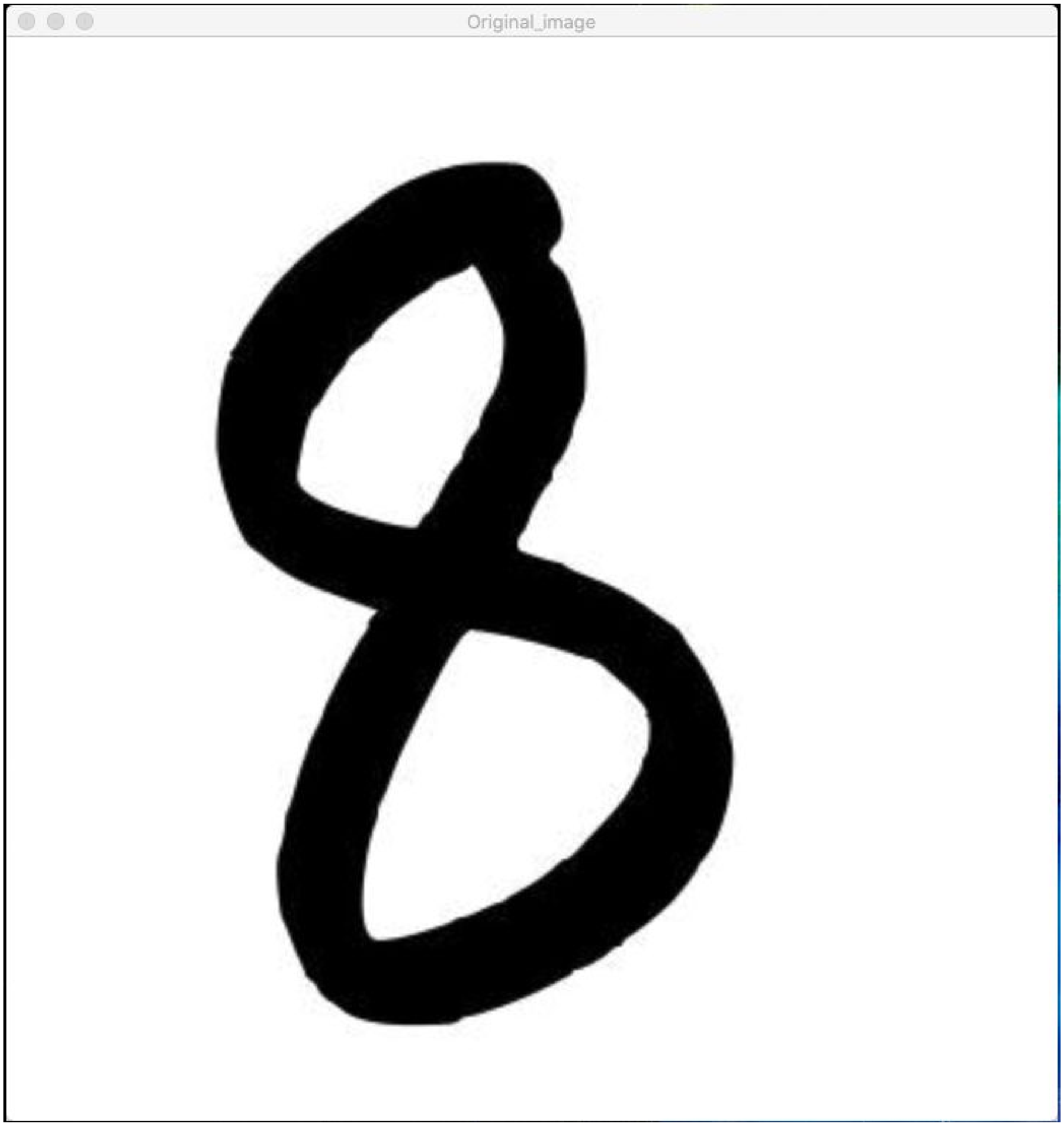
Final Results and validating results:

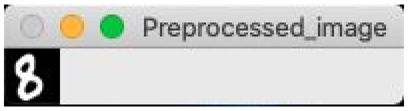


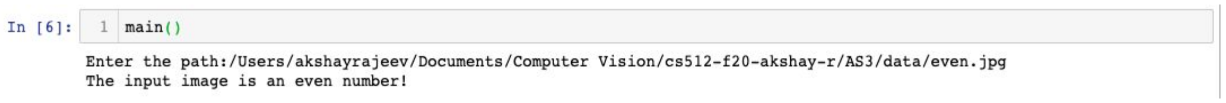
MODELS	Validation Accuracy	Validation Loss	Test Accuracy	Test Loss
Basic model	0.9757000207901001	0.06908606078773737	0.978600025177002	0.06877065032720565
Change the network architecture	0.8827999830245972	0.3771093502044678	0.8845999836921692	0.3787911904335022
Changing the receptive field and stride parameters	0.9496999979019165	0.1435967242896557	0.9449999928474426	0.15016782046556473
Changing the optimizer and loss functions:	0.9624999761581421	0.09433307630568742	0.9607999920845032	0.0961623150214553
Changing Various parameters (learning rate and number of epochs)	0.9193999767303467	0.20727372806072236	0.9161999821662903	0.21574462983608245
Adding batch and layer normalization	0.9860000014305115	0.04243082661926746	0.9836000204086304	0.044558984037488696
Use different weight initializers	0.9768999814987183	0.06738427998274565	0.9769999980926514	0.06786776476204395

The Adding batch and layer normalization (Model 5) was used to validate the classification.

First an pre-processed image of digit 8 was passed through the loaded model the out of the program is as follows:

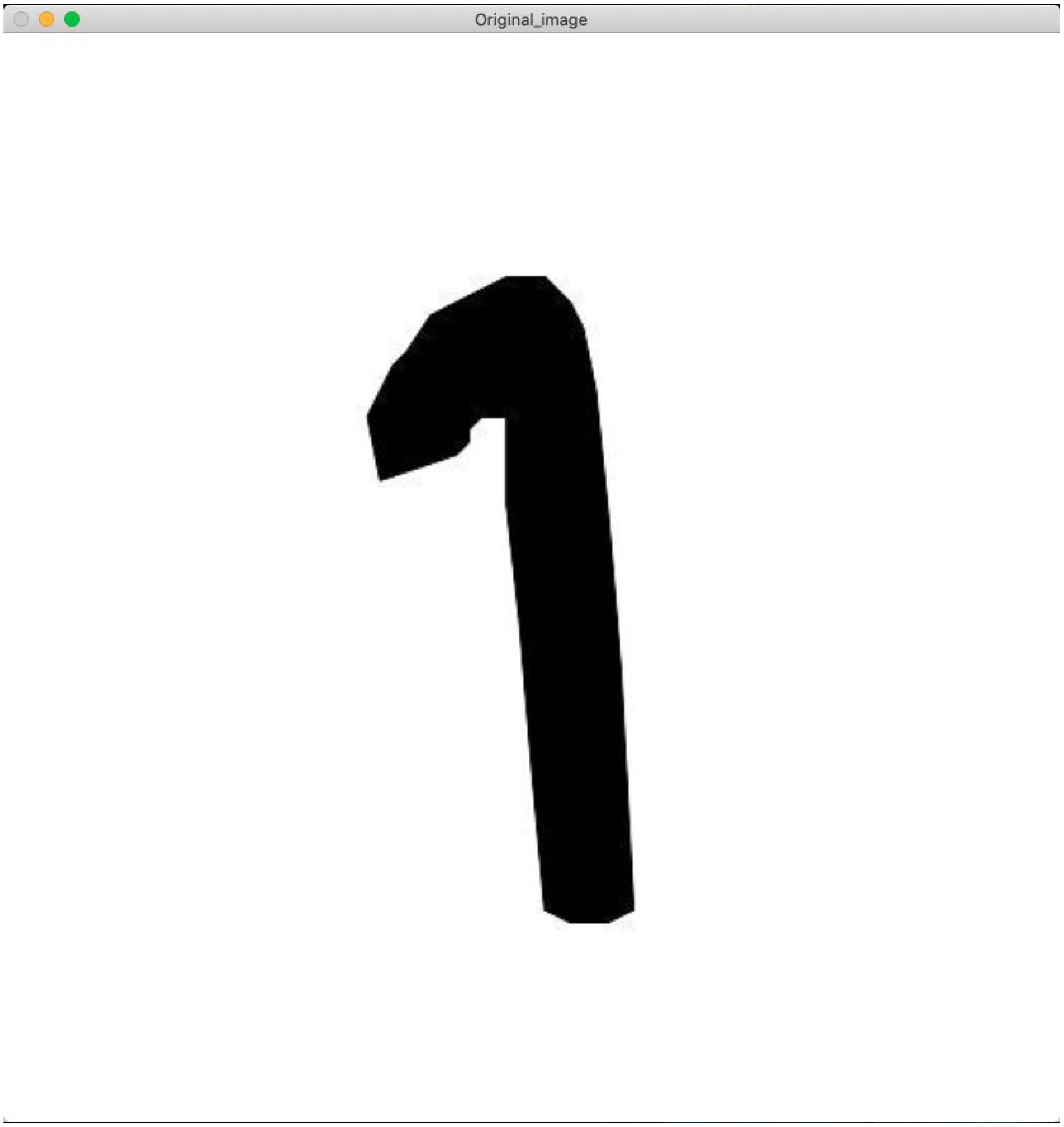


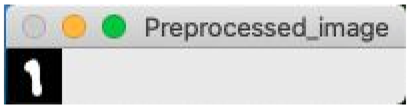


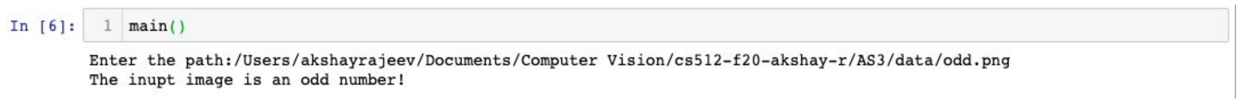


```
In [6]:  main()
Enter the path:/Users/akshayrajeev/Documents/Computer Vision/cs512-f20-akshay-r/AS3/data/even.jpg
The input image is an even number!
```

It correctly classified the image as an even number. Next a pre-processed image of the digit 1 was passed through the pre loaded model the result of this was:







```
In [6]:  main()

Enter the path:/Users/akshayrajeev/Documents/Computer Vision/cs512-f20-akshay-r/AS3/data/odd.png
The inupt image is an odd number!
```

The program correctly classified both images as odd and even respectively.

References:

- https://smerity.com/articles/2016/orthogonal_init.html
- https://plotly.com/python/table/
- https://www.geeksforgeeks.org/python-membership-identity-operators-not-not/
- https://keras.io/api/losses/hinge_losses/#hinge-class
- https://www.codegrepper.com/code-examples/python/cv2.error%3A+OpenCV%28
4.4.0%29+%2Fprivate%2Fvar%2Ffolders%2Fnz%2Fvv4_9tw56nv9k3tkvyszvwg800
00gn%2FT%2Fpip-req-build-gi6lxw0x%2Fopencv%2Fmodules%2Fimgcodecs%2Fs
rc%2Floadsave.cpp%3A738%3A+error%3A+%28-215%3AAssertion+failed%29+%2
1_img.empty%28%29+in+function+%27imwrite%27