Assignment 3 Report

Harsh Vora
CWID : A20445400
CS512 – F20

November 09, 2020

1. Problem Statement

- To Implement Convolutional Neural Networks to classify images of numbers in the MNIST dataset as either even or odd.
- To construct and train CNN.
- Perform Hyper-parameter Tuning by evaluating different variations of the basic network and measure performance.
- At last write a program to use the pretrained model and the CNN model should classify the image as Even or Odd.

2. Proposed Solution

- I constructed a CNN Network with a combination of several convolutional layers, max pooling, a Dropout layer, Flatten and two fully connected layers.
- After that I trained the network and recorded the training and validation loss and accuracy and plot the graphs for training and validation loss as well as training and validation accuracy.
- In addition to this I also recorded the loss and accuracy values of the final training step.
- After that I performed Hyper parameter tuning by changing the network architecture, stride parameters, changed optimizer and loss functions, added batch and layer normalization, Used different weight Initializers and changed various parameters like dropout, learning rate, number of filters and number of epochs.
- After analyzing all the models and finding out the best I retrained the best model and evaluated that model on the testing subset.
- For the last step I loaded the best model and took input of a handwritten image and prepared the image for the model that I constructed, the preparation includes Resizing the image, transforming grayscale image to binary image and also used GaussianBlur() and adaptiveThreshold().
- Then used the best CNN model to classify the image as even or odd.

3. Implementation Details

1.Construct and train CNN:

- I loaded the MNIST dataset by using the mnist.load_data() function and split the data into training, validation & testing subsets of 55,000/10,000/5,000.

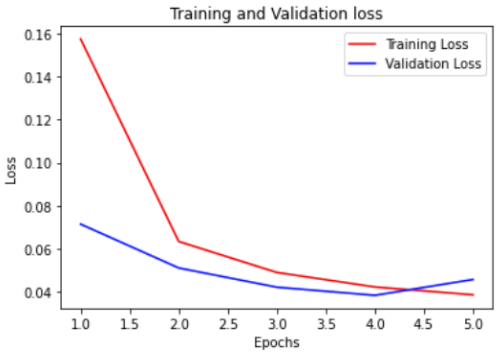
- I took 5000 images from the training subset and 5000 images from the testing subset to form the 10000 images validation subset by using np.vstack() function
- After that I converted the digit labels into odd and even labels and discarded the original digit labels and performed to_categorical() on train, test & validation labels subsets.
- This will convert labels into binary i.e. 0 or 1. We can assume 0 for even digits and 1 for odd digits.
- After this I constructed a model by taking two convolutional layers with pooling, a dropout layer, and two fully connected layers.
- After that I compiled the model by choosing optimizer as rmsprop and loss as categorical_crossentropy.

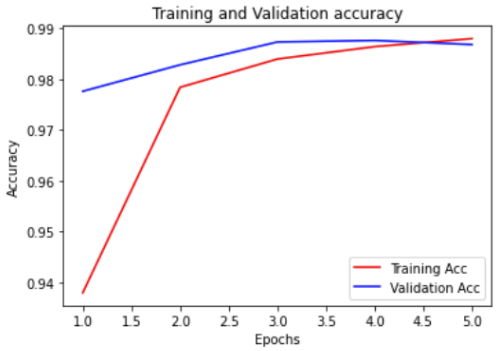
2.Hyper-parameter Tuning:

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

Added 1 more convolutional layer and applied MaxPooling for that Conv2D layer and trained the model for 5 epochs and batch size 64.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 64)          36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 1, 1, 64)          0
_________________________________________________________________
dropout (Dropout)            (None, 1, 1, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 128)               8320
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 258
=================================================================
Total params: 64,322
Trainable params: 64,322
Non-trainable params: 0
_________________________________________________________________
```



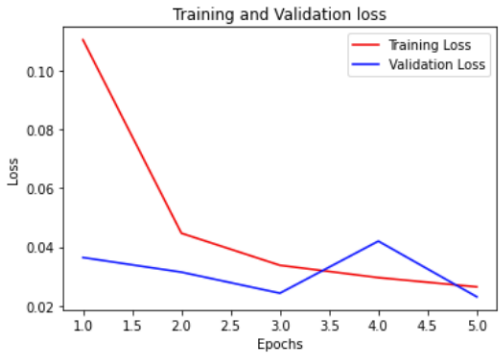


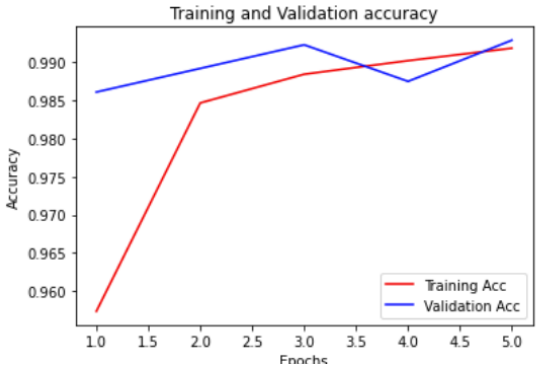
I got the Accuracy = 98.68 %

(b) **Changing the receptive field and stride parameters.**

Added strides=(1,1) in two convolutional layers and applied padding='same' in one Conv2D layer.

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_3 (Conv2D)            (None, 28, 28, 32)        320
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 32)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 12, 12, 64)        18496
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 4, 4, 64)          36928
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 2, 2, 64)          0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2, 2, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 128)               32896
_________________________________________________________________
dense_3 (Dense)              (None, 2)                 258
=================================================================
Total params: 88,898
Trainable params: 88,898
Non-trainable params: 0
_________________________________________________________________
```



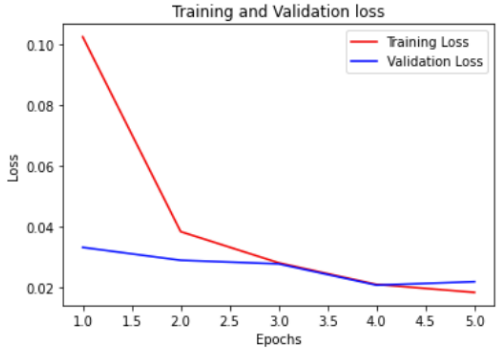


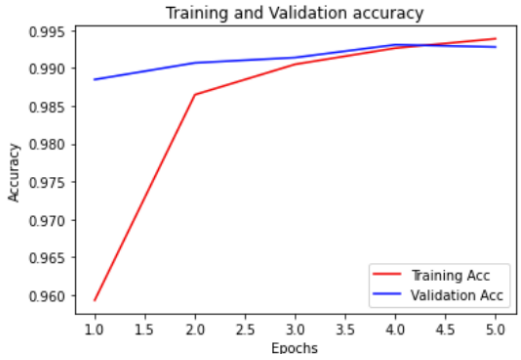
I got the Accuracy = 99.28 %

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

I changed the optimizer and the loss function I was using Optimizer=rmsprop and Loss=categorical_crossentropy so I changed it to Optimizer=adam and Loss=binary_crossentropy and also Optimizer=SGD
and Loss=binary_crossentropy

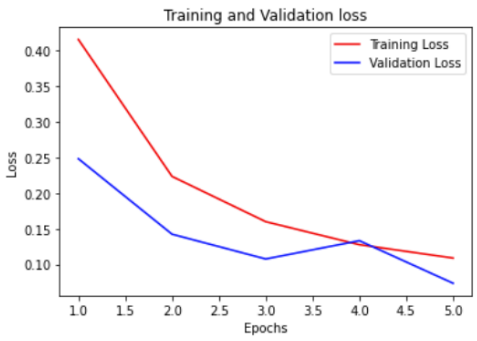
```
# Compile network
model.compile( optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])
```

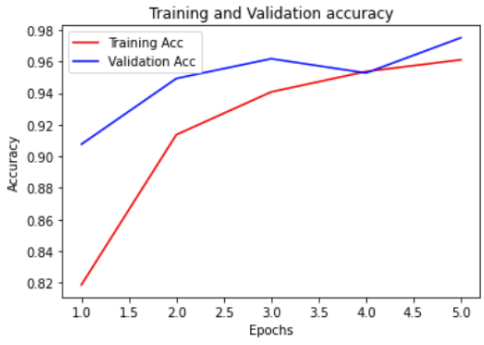




I got accuracy = 99.27%

```
# Compile network
model.compile( optimizer = "SGD", loss = 'binary_crossentropy', metrics = ['accuracy'])
```





I got accuracy = 97.50%

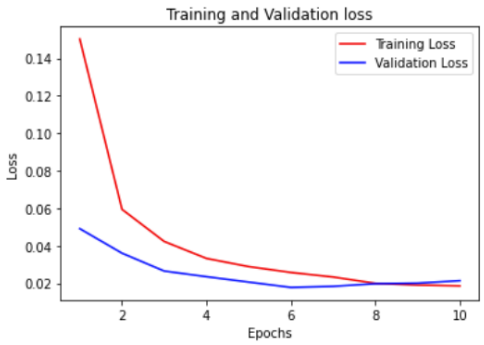
(d) **Changing various parameters (e.g dropout, learning rate, number of filters, number of epochs)**

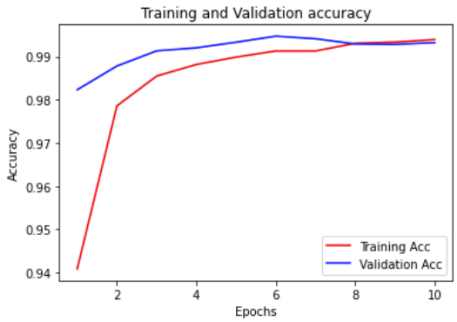
I changed the parameters like dropout initially I was taking dropout(0.4) now I am taking (0.6) and also adding the learning rate as 0.001 for adam optimizer and increasing the epochs to 10 and batch size to 128.

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu',input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.6))
model.add(layers.Flatten())
model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dense(2,activation='softmax'))
model.summary()
```

```
# Compile network
opt = Adam(lr=0.001)
model.compile( optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

```
# Training the model

history = model.fit(train_images_subs, train_labels_subs, epochs = 10, batch_size= 128, validation_data=(train_val, train_lab_val))
```



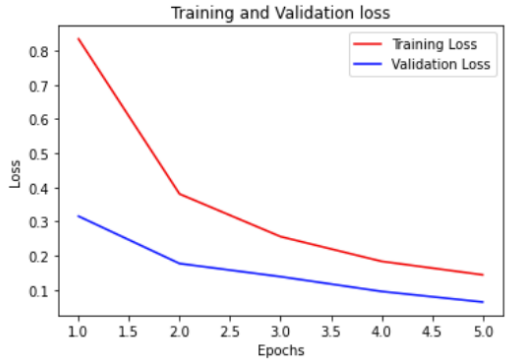


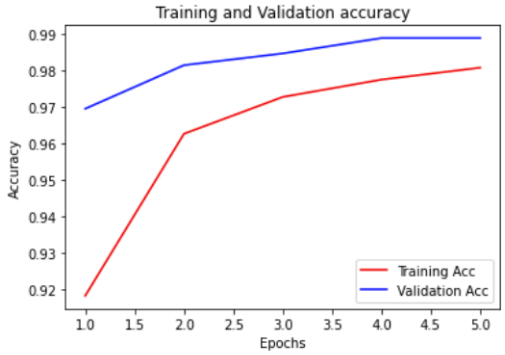
I got accuracy = 99.32%

(e) **Adding batch and layer normalization**

I added batch normalization after every layer and got a decent accuracy.

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu',input_shape=(28,28,1)))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.BatchNormalization())
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.BatchNormalization())
model.add(layers.Dropout(0.6))
model.add(layers.BatchNormalization())
model.add(layers.Flatten())
model.add(layers.BatchNormalization())
model.add(layers.Dense(128,activation='relu'))
model.add(layers.BatchNormalization())
model.add(layers.Dense(2,activation='softmax'))
model.add(layers.BatchNormalization())

model.summary()
```



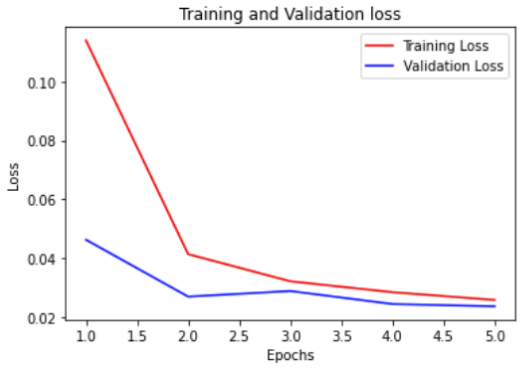


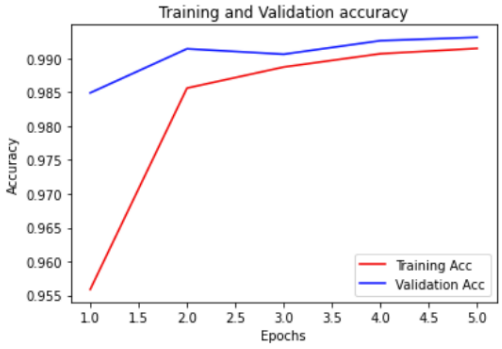
I got accuracy = 98.87%

(f) **Using different weight initializer**

I used Xavier as weight initializer in the Conv2D layer.

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu', kernel_initializer=xavier,input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3),activation='relu', kernel_initializer=xavier))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.4))
model.add(layers.Flatten())
model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dense(2,activation='softmax'))

model.summary()
```





I got accuracy = 99.30%

3. Inference

I am using the pretrained model to perform classification on handwritten digit image.
The code accepts an input image of a handwritten digit. Assuming each image contains one digit.
After that I am doing basic image processing to prepare the image for CNN.
Such as Resizing the image to fit mode size requirement.
Transforming the grayscale image to binary image and also used the GaussianBlur() and adaptiveThreshold() for binary thresholding.

```
resized_img = cv2.resize(img, (28,28), interpolation = cv2.INTER_AREA)
print('Resized Dimensions : ',resized_img.shape)
```

```
Resized Dimensions :  (28, 28, 3)
```

+ Code + Text

```
grayscale_img = cv2.cvtColor(resized_img, cv2.COLOR_BGR2GRAY)
blur_img = cv2.GaussianBlur(grayscale_img, (3,3), 0)
req_img = cv2.adaptiveThreshold(blur_img, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY,5,2)
```

Finally, I am using the pretrained model to classify the binary image as Even or Odd.

```
req_img = req_img.astype( 'float32' ) / 255
req_img = req_img.reshape((1, 28, 28,1))
```

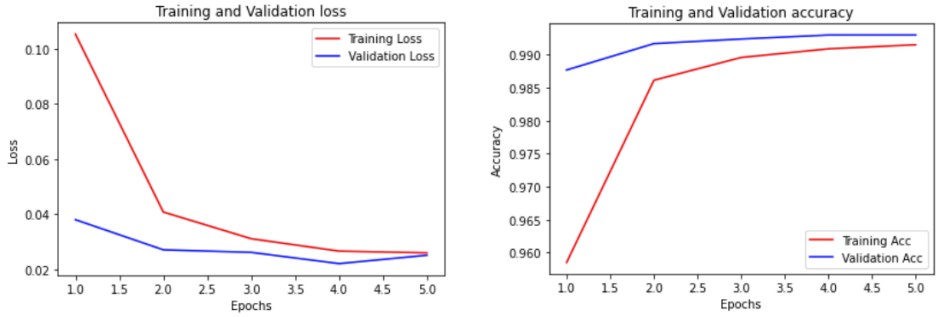
```
result = new_model.predict(req_img)
print(result)
if result[0,0] > result[0,1]:
  print("The number is Even")
else:
  print("The number is Odd")
```

4. Results and Discussion

1. Construct and train CNN

Initially I used the model consisting of 2 Conv2D layers along with MaxPooling applied to both the layers then I added a Dropout of 0.4 and after using Flatten() I added 2 Dense Layers Dense(128) and Dense(2) respectively and used rmsprop as the optimizer and loss as categorical_crossentropy with epochs = 5 and batch_size = 64.

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3), activation='relu',input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.4))
model.add(layers.Flatten())
model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dense(2,activation='softmax'))

model.summary()
```



I got the accuracy = 99.29%

This is the last epoch's accuracy and loss.

```
[618] print("Accuracy at the last epoch:")
      print("Training Accuracy:", acc[-1])
      print("Validation Accuracy:", val_acc[-1])
```

```
Accuracy at the last epoch:
Training Accuracy: 0.991527259349823
Validation Accuracy: 0.9929999709129333
```

```
[619] print("Loss at the last epoch:")
      print("Training Loss:", loss[-1])
      print("Validation Loss:", val_loss[-1])
```

```
Loss at the last epoch:
Training Loss: 0.025903886184096336
Validation Loss: 0.02504638396203518
```

I got this result after running the normal model on the test images (i.e. Final Training Results)

```
# Training the model

history = model.fit(train_images_subs, train_labels_subs, epochs = 5, batch_size= 64)
results = model.evaluate(test_images_subs,test_label_subs)
print(results)
```

```
Epoch 1/5
860/860 [==============================] - 3s 4ms/step - loss: 0.1065 - accuracy: 0.9586
Epoch 2/5
860/860 [==============================] - 3s 4ms/step - loss: 0.0389 - accuracy: 0.9872
Epoch 3/5
860/860 [==============================] - 3s 3ms/step - loss: 0.0316 - accuracy: 0.9900
Epoch 4/5
860/860 [==============================] - 3s 4ms/step - loss: 0.0266 - accuracy: 0.9913
Epoch 5/5
860/860 [==============================] - 3s 3ms/step - loss: 0.0252 - accuracy: 0.9919
157/157 [==============================] - 0s 2ms/step - loss: 0.0914 - accuracy: 0.9846
[0.09137596189975739, 0.9846000075340271]
```

```
print(results)
model.save("cv_model.h5")
```

```
[0.09137596189975739, 0.9846000075340271]
```

This is the accuracy I am getting from the final training i.e. 98.46%

2. Hyper-parameter Tuning

As seen above in the Implementation details 2 part I got all the accuracy and as seen above the accuracy of the first model without any hyperparameter training.

Models	Normal Model	3.2.(a)	3.2.(b)	3.2.(c.1)	3.2.(c.2)	3.2.(d)	3.2.(e)	3.2.(f)
Accuracy	99.29%	98.68%	99.28%	99.27%	97.50%	99.32%	98.87%	99.30%

As you can see almost most of the models give similar kind of accuracy but there is a fraction of difference in the model's accuracy. So according to me the best model out of the all is the model(3.2.d) where parameters like dropout is changed to (0.6) and also the learning rate is set to 0.001 for adam optimizer and increasing the epochs to 10 and batch size to 128.

So Applying the 3.2.(d) model to the testing subset to get the accuracy on the testing subset and got the results.

```
model = models.Sequential()
model.add(layers.Conv2D(32,(3,3),activation='relu',input_shape=(28,28,1)))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Conv2D(64,(3,3),activation='relu'))
model.add(layers.MaxPooling2D((2,2)))
model.add(layers.Dropout(0.6))
model.add(layers.Flatten())
model.add(layers.Dense(128,activation='relu'))
model.add(layers.Dense(2,activation='softmax'))
model.summary()
```

```
Model: "sequential_20"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_40 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_38 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_41 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_39 (MaxPooling (None, 5, 5, 64)          0
_________________________________________________________________
dropout_18 (Dropout)         (None, 5, 5, 64)          0
_________________________________________________________________
flatten_18 (Flatten)         (None, 1600)              0
_________________________________________________________________
dense_36 (Dense)             (None, 128)               204928
_________________________________________________________________
dense_37 (Dense)             (None, 2)                 258
=================================================================
Total params: 224,002
Trainable params: 224,002
Non-trainable params: 0
```

```
# Compile network
opt = Adam(lr= 0.001)
model.compile( optimizer = opt, loss = 'binary_crossentropy', metrics = ['accuracy'])
```

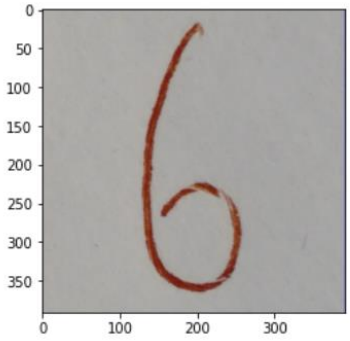
```
# Training the model

history = model.fit(train_images_subs, train_labels_subs, epochs = 10, batch_size= 128)
results = model.evaluate(test_images_subs,test_label_subs)
print(results)
```

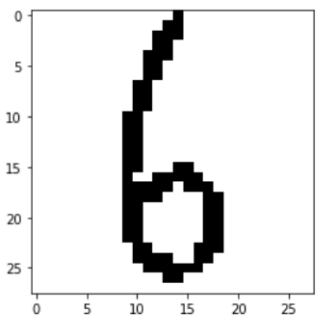
```
Epoch 1/10
430/430 [==============================] - 2s 4ms/step - loss: 0.1535 - accuracy: 0.9399
Epoch 2/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0585 - accuracy: 0.9795
Epoch 3/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0423 - accuracy: 0.9854
Epoch 4/10
430/430 [==============================] - 2s 5ms/step - loss: 0.0330 - accuracy: 0.9886
Epoch 5/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0280 - accuracy: 0.9902
Epoch 6/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0260 - accuracy: 0.9909
Epoch 7/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0238 - accuracy: 0.9916
Epoch 8/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0209 - accuracy: 0.9930
Epoch 9/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0196 - accuracy: 0.9928
Epoch 10/10
430/430 [==============================] - 2s 4ms/step - loss: 0.0182 - accuracy: 0.9934
157/157 [==============================] - 0s 2ms/step - loss: 7.5864 - accuracy: 0.9790
[7.586421012878418, 0.9789999723434448]
```

I got the accuracy of final testing data as 97.89%

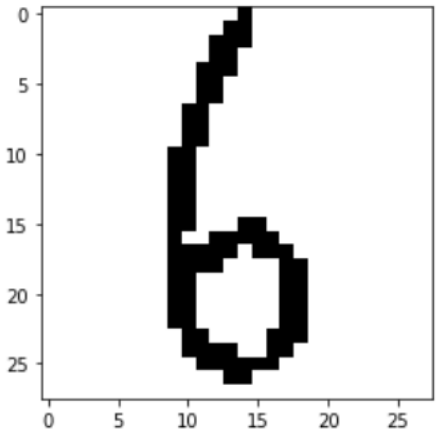
3. Inference

Displaying Binary Image and Original Image in separate Windows.





```
<matplotlib.image.AxesImage at 0x7f5de90604a8>
```

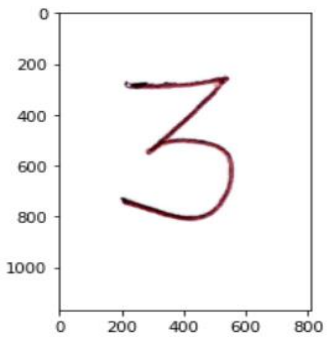


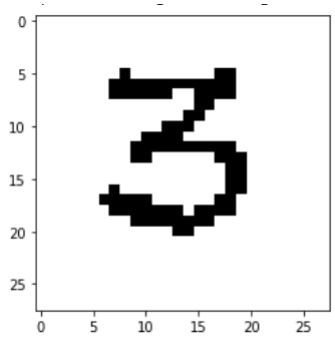
```
req_img = req_img.astype( 'float32' ) / 255
req_img = req_img.reshape((1, 28, 28,1))
```

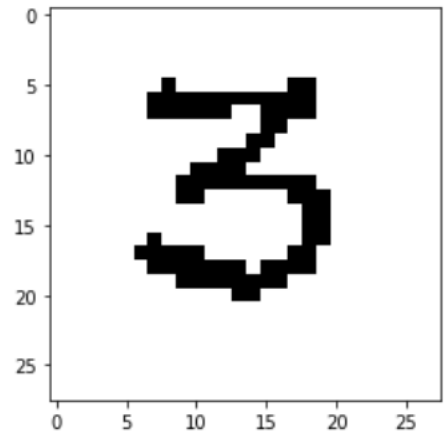
```
result = new_model.predict(req_img)
print(result)
if result[0,0] > result[0,1]:
  print("The number is Even")
else:
  print("The number is Odd")
```

```
WARNING:tensorflow:11 out of the last 11 calls to <function
[[0.97094125 0.0290588 ]]
The number is Even
```

Odd Number Detection








```
req_img = req_img.astype( 'float32' ) / 255
req_img = req_img.reshape((1, 28, 28,1))
```

```
result = new_model.predict(req_img)
print(result)
if result[0,0] > result[0,1]:
  print("The number is Even")
else:
  print("The number is Odd")
```

```
WARNING:tensorflow:11 out of the last 11 calls to <function Model.make
[[0.06081068 0.9391893 ]]
The number is Odd
```