Homework 1 – [Bhati –xxxxxxx36]

Screenshots of Coding Output

My ID = [redacted]36. So the last two digits are 3 and 6.

```
# filter out the training and test sets with these two digits - 3 and 6
x_train = x_train[(y_train == 3) | (y_train == 6)]
y_train = y_train[(y_train == 3) | (y_train == 6)]
x_test = x_test[(y_test == 3) | (y_test == 6)]
y_test = y_test[(y_test == 3) | (y_test == 6)]
```

```
# For binary classification modify labels: Label encoding [3 => 0 and 6 => 1]
y_train = np.where(y_train == 3, 0, 1)
y_test = np.where(y_test == 3, 0, 1)
```

In [13]:
```
# Pixel values range from 0 to 255. Normalize pixel values of train and test to the range [0, 1]
x_train, x_test = x_train / 255.0, x_test / 255.0
```

In [14]:
```
x_train[0] # Normalized x_train
```

```
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.17647059, 0.87058824,
 0.98823529, 0.98823529, 0.98823529, 0.98823529, 0.99215686,
 0.98823529, 0.98823529, 0.98823529, 0.69411765, 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.17647059, 0.8745098 ,
 0.99215686, 0.99215686, 0.99215686, 0.99215686, 1.        ,
 0.99215686, 0.99215686, 0.99215686, 0.99215686, 0.29019608,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
[0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        , 0.        , 0.12156863,
 0.48235294, 0.20392157, 0.17254902, 0.17254902, 0.17254902,
 0.17254902, 0.56078431, 0.98823529, 0.98823529, 0.29019608,
 0.        , 0.        , 0.        , 0.        , 0.        ,
 0.        , 0.        , 0.        ],
```

Neural Network for Binary Classification

[15]:
```
# Batch size and Learning rate are hyperparameters
batch_size = 32
learning_rate = 0.001
```

[16]:
```
model = Sequential()

model.add(Flatten(input_shape=(28,28))) ## 28x28 pixels are converted to 1D 784 input units.
model.add(Dense(128,activation='relu')) # 128 nodes with ReLU activation to add non-linearity.
model.add(Dense(1,activation='sigmoid')) ## 1 output node for binary classication using sigmoid.
```

[17]:
```
model.summary()

```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten (Flatten)           (None, 784)               0

 dense (Dense)               (None, 128)               100480

 dense_1 (Dense)             (None, 1)                 129

=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
_________________________________________________________________
```

Params in the second layer are 100480 because 784 inputs * 128 nodes(weights = 100352) + 128(baises) = 100480

Params in the last layer = 129 because 128(wts) + 1(bias) = 129

Binary_Crossentropy is used for binary classification and optimization is chosen as Adam

[18]:
```
# Compiling the model
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='binary_crossentropy',metrics=['accuracy'
```

Early stopping criteria based on validation loss

[19]:
```
# Early stopping criteria based on validation loss
# patience = Number of epochs with no improvement after which training will be stopped.
# restore_best_weights = Whether to restore model weights from the epoch with the best value of the monitored quantity.
early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

[20]:
```
# Training the model
history = model.fit(x_train, y_train, batch_size=batch_size, epochs=20, validation_split=0.2, callbacks=[early_stopping])
```

```
Epoch 1/20
302/302 [==============================] - 1s 2ms/step - loss: 0.0344 - accuracy: 0.9899 - val_loss: 0.0086 - val_accuracy: 0.9
975
Epoch 2/20
302/302 [==============================] - 0s 1ms/step - loss: 0.0098 - accuracy: 0.9969 - val_loss: 0.0055 - val_accuracy: 0.9
979
Epoch 3/20
302/302 [==============================] - 0s 2ms/step - loss: 0.0054 - accuracy: 0.9985 - val_loss: 0.0045 - val_accuracy: 0.9
979
Epoch 4/20
302/302 [==============================] - 0s 2ms/step - loss: 0.0026 - accuracy: 0.9997 - val_loss: 0.0021 - val_accuracy: 0.9
996
Epoch 5/20
302/302 [==============================] - 0s 1ms/step - loss: 0.0011 - accuracy: 0.9999 - val_loss: 0.0082 - val_accuracy: 0.9
967
Epoch 6/20
```

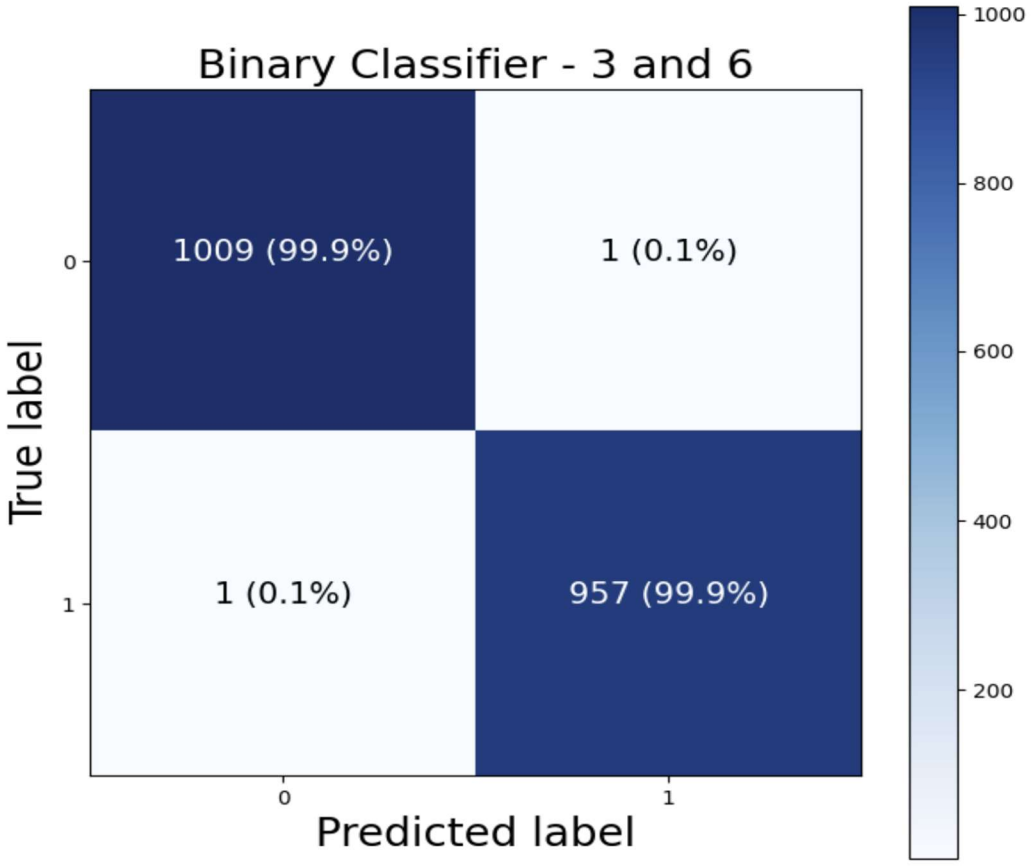
30]:
```
# Evaluate the model on the test data
test_loss, test_accuracy = model.evaluate(x_test, y_test)
print("Test Accuracy:", test_accuracy)
```

```
62/62 [==============================] - 0s 869us/step - loss: 0.0024 - accuracy: 0.9990
Test Accuracy: 0.9989837408065796
```

Construct a confusion matrix

28]:
```
conf_matrix = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(conf_matrix)
draw_confusion_matrix(y_test, y_pred,"Binary Classifier - 3 and 6")
```

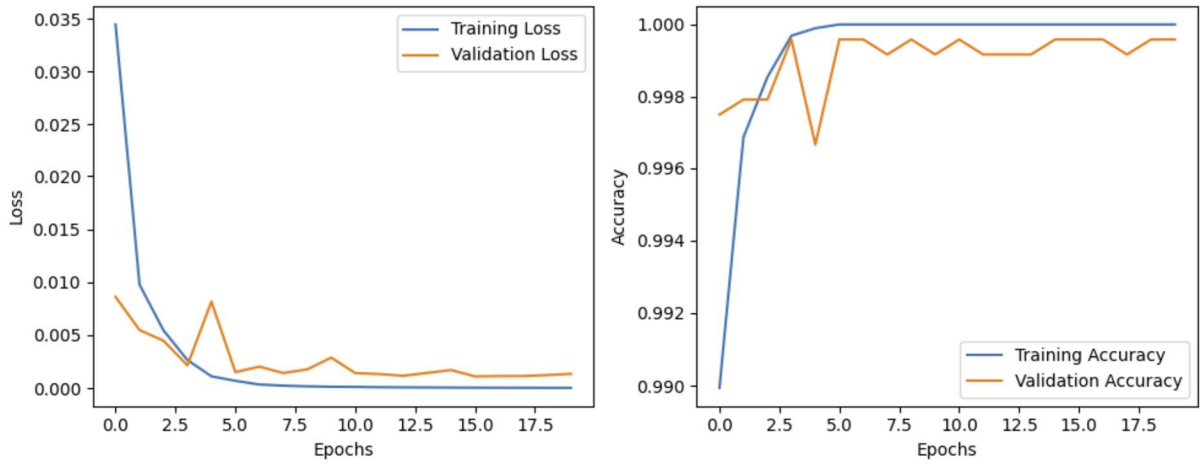
```
Confusion Matrix:
[[1009    1]
 [   1  957]]
```



Just two instances have been misclassified. 1 instance of class 3 has been misclassified as class 6(False Negative) and 1 instance of Class 6 has been misclassified as 3(False Positive)

Learning Curve - Loss and Accuracy

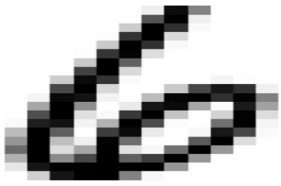
.]:
```python
# Present the learning curve
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```



Show some examples of predictions

:
```python

eg_indices = np.random.choice(len(x_test), 10) # 10 randomly selected samples
eg_images = x_test[eg_indices]
eg_labels = y_test[eg_indices]
eg_predictions = model.predict(eg_images)
for i in range(len(eg_images)):
    plt.figure()
    plt.imshow(eg_images[i], cmap='Greys')
    plt.title(f"True Label: {eg_labels[i]}, Predicted: {np.round(eg_predictions[i][0])}")
    plt.axis('off')
```

True Label: 1, Predicted: 1.0



True Label: 0, Predicted: 0.0

TABLE for 2-Class(Binary Class) Hyperparameters

Activation function – hidden layer	ReLU
Activation function - output layer	Sigmoid
Weight Initializer	Default -GlorotUniform
Number of hidden layers	1 (2 including output layer)
Neurons in hidden layers	128
Neurons in the output layer	1
Loss function	Binary_crossentropy
Optimizer	Adam
Number of epochs	20
Batch size	32
Learning rate	0.001
Evaluation Metric	Accuracy

B. Three different Weight Initializers. As per keras documentation there are multiple initializers such as He normal, GlorotNormal, RandomNormal, and GlorotUniform.

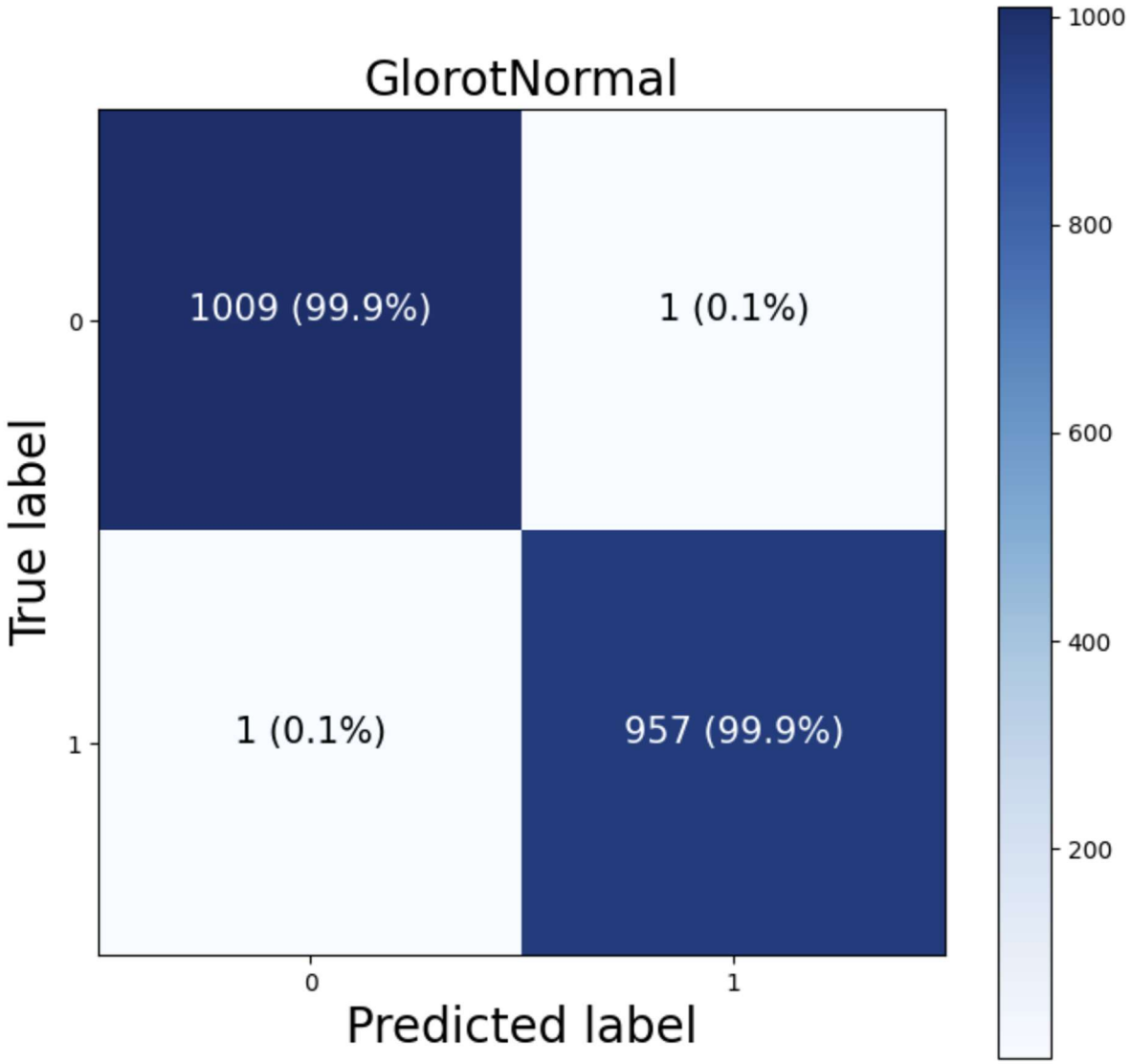
3]:
```
1 from tensorflow.keras.initializers import GlorotNormal, HeNormal, RandomNormal,GlorotUniform
2
```

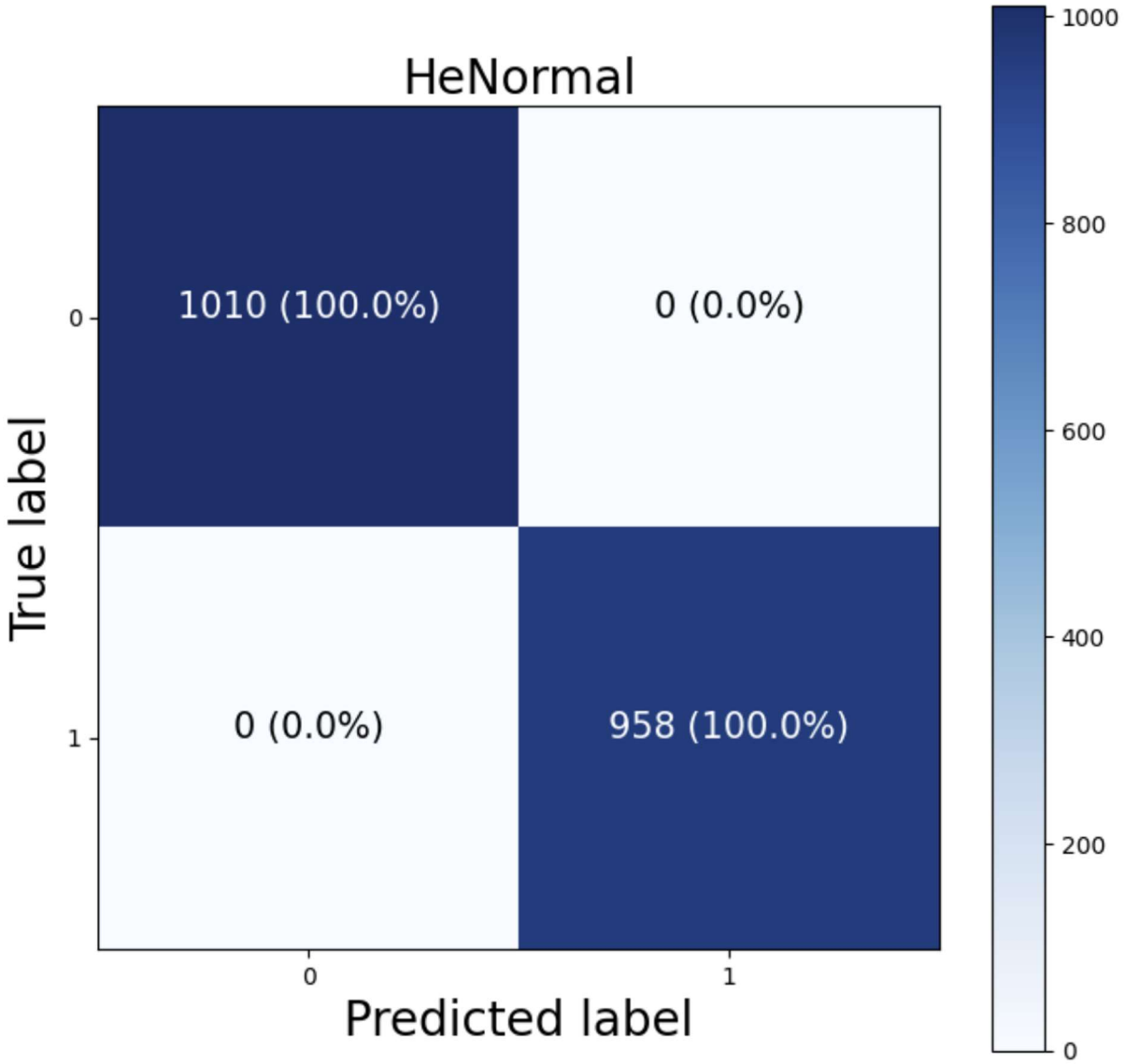
```python
# Lists to store results
learning_curves = []
accuracies = []
```

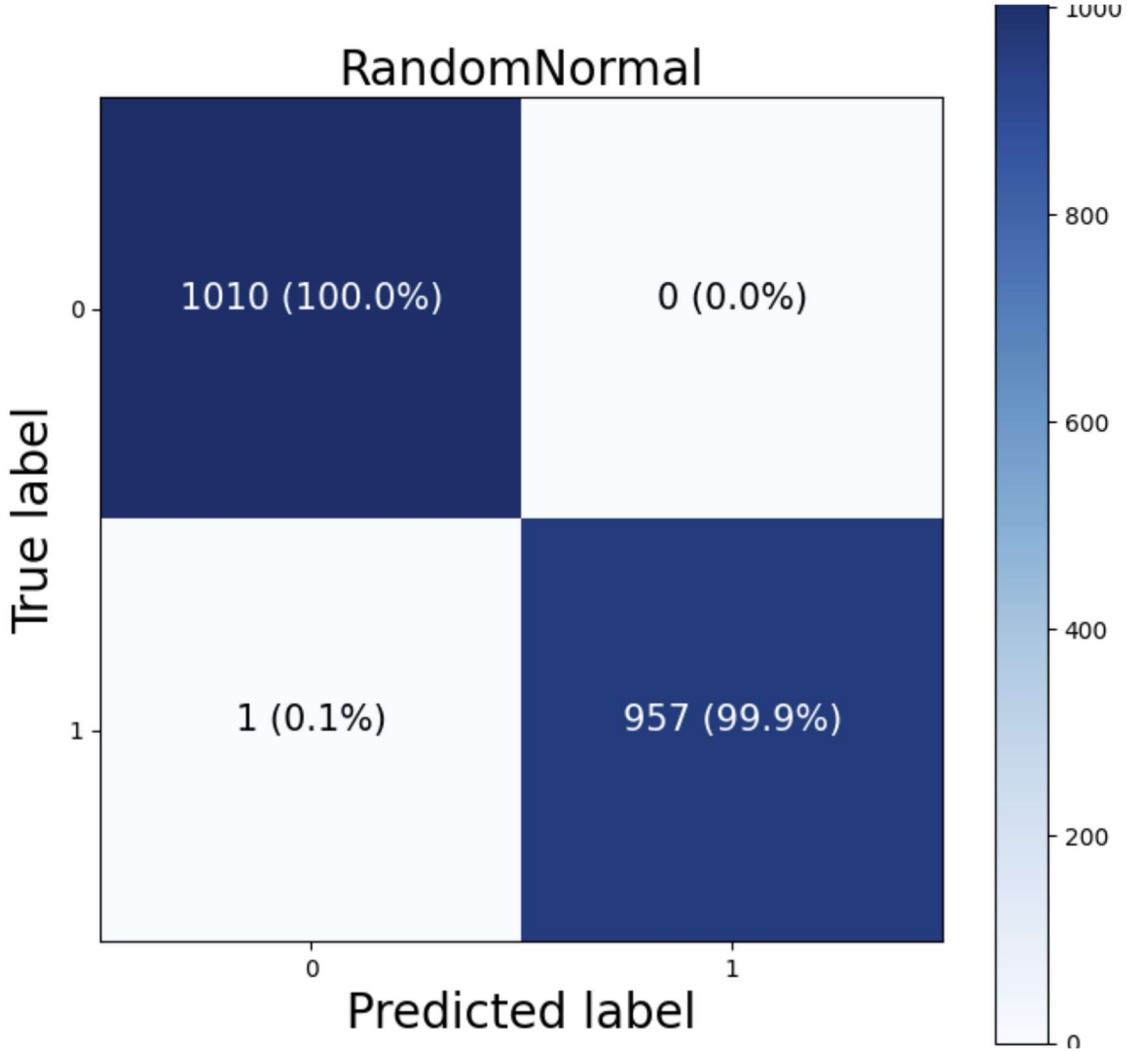
```python
labels = ["GlorotNormal", "HeNormal", "RandomNormal","GlorotUniform"]
i=0
# Build and train models for each initializer
for initializer in initializers:

    model = Sequential([
        Flatten(input_shape=(28, 28)),
        Dense(128, activation='relu', kernel_initializer=initializer),
        Dense(1, activation='sigmoid')
    ])
    model.summary()
    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='binary_crossentropy', metrics=['accu

    # Early stopping criteria based on validation loss
    # patience = Number of epochs with no improvement after which training will be stopped.
    # restore_best_weights = Whether to restore model weights from the epoch with the best value of the monitored quantity.
    early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)

     # Train the model
    history = model.fit(x_train, y_train, batch_size=batch_size, epochs=20, validation_split=0.2, verbose=0,callbacks=[early_

    # Evaluate on test data
    y_pred = np.round(model.predict(x_test))
    accuracy = accuracy_score(y_test, y_pred)
    confusion = confusion_matrix(y_test, y_pred)
    draw_confusion_matrix(y_test, y_pred,labels[i])

    #confusion_matrices.append(confusion)
    learning_curves.append(history.history)
    accuracies.append(accuracy)
    i=i+1
```

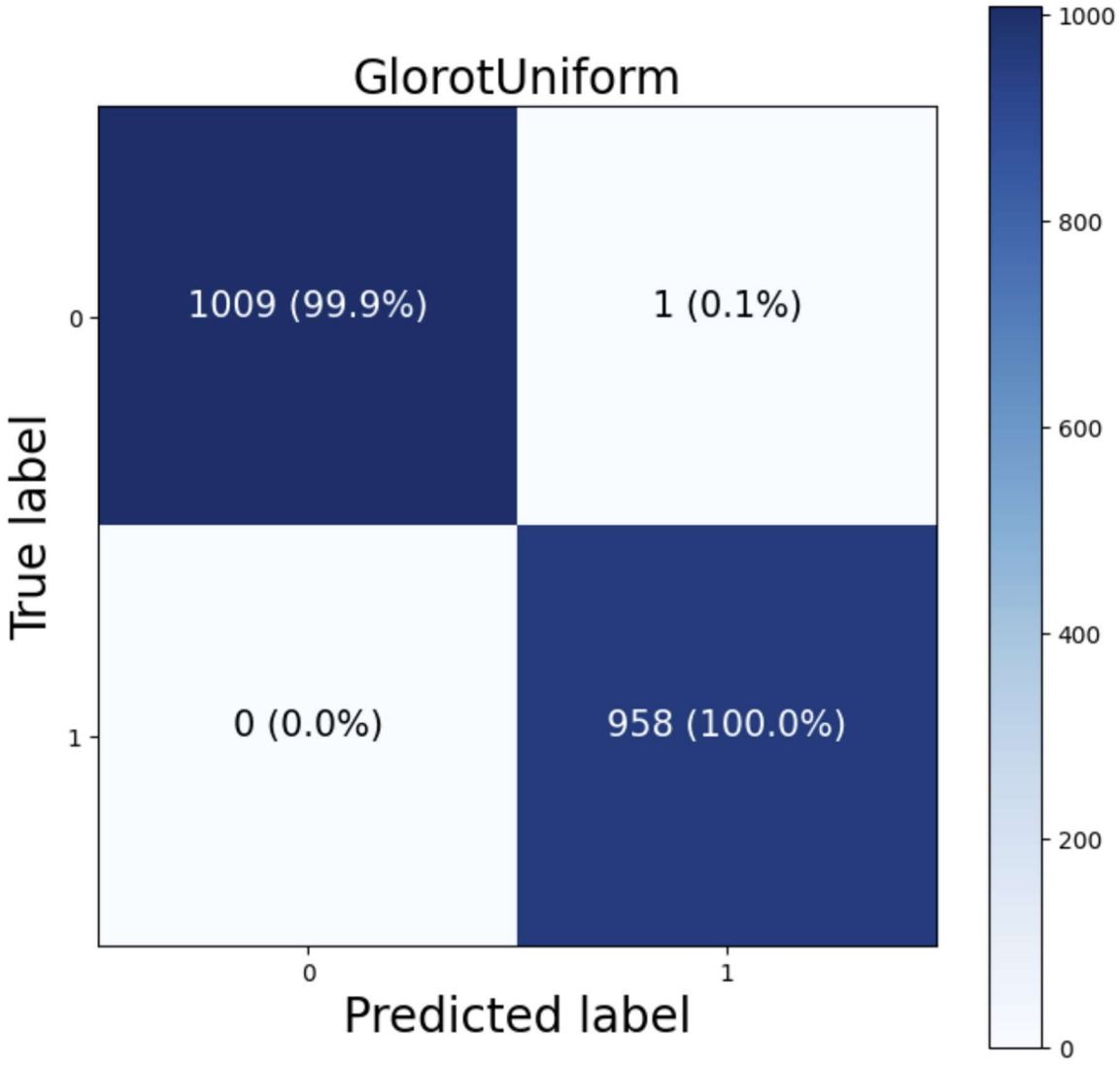
```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_1 (Flatten)         (None, 784)               0

 dense_2 (Dense)             (None, 128)               100480

 dense_3 (Dense)             (None, 1)                 129

=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
_________________________________________________________________
62/62 [==============================] - 0s 738us/step
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_2 (Flatten)         (None, 784)               0

 dense_4 (Dense)             (None, 128)               100480

 dense_5 (Dense)             (None, 1)                 129

=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
_________________________________________________________________
62/62 [==============================] - 0s 721us/step
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_3 (Flatten)         (None, 784)               0

 dense_6 (Dense)             (None, 128)               100480

 dense_7 (Dense)             (None, 1)                 129

=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
```

```
62/62 [==============================] - 0s 754us/step
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_4 (Flatten)         (None, 784)               0

 dense_8 (Dense)             (None, 128)               100480

 dense_9 (Dense)             (None, 1)                 129

=================================================================
Total params: 100,609
Trainable params: 100,609
Non-trainable params: 0
_________________________________________________________________
62/62 [==============================] - 0s 869us/step
```


GlorotNormal
1009 (99.9%)
1 (0.1%)
1 (0.1%)
957 (99.9%)
True label
Predicted label
0
1
0
1
1000
800
600
400
200

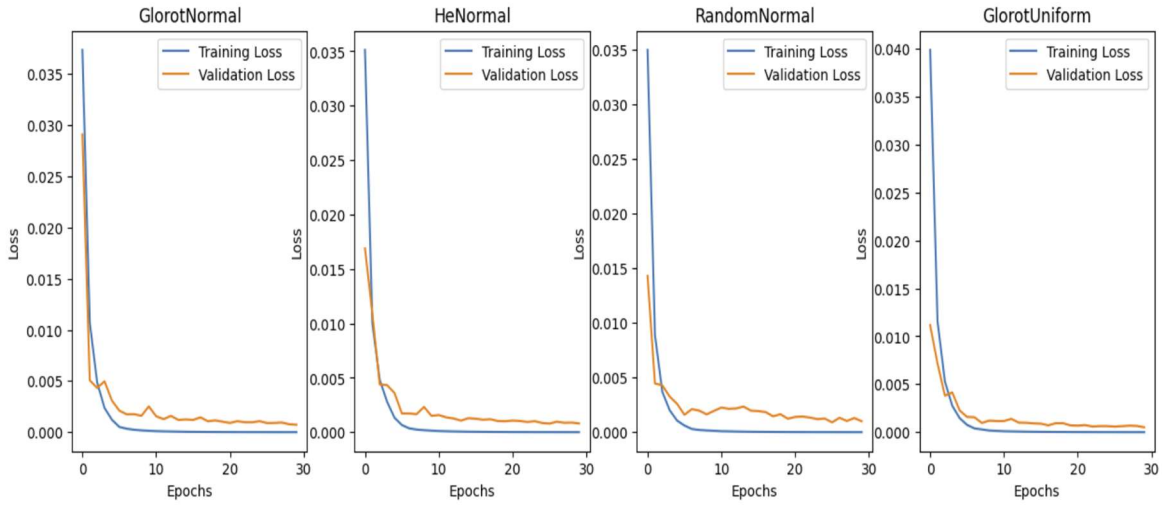

HeNormal
1010 (100.0%)
0 (0.0%)
0 (0.0%)
958 (100.0%)
True label
Predicted label
0
1
0
1
1000
800
600
400
200
0


RandomNormal
1010 (100.0%)
0 (0.0%)
1 (0.1%)
957 (99.9%)
True label
Predicted label
0
1
0
1
1000
800
600
400
200
0



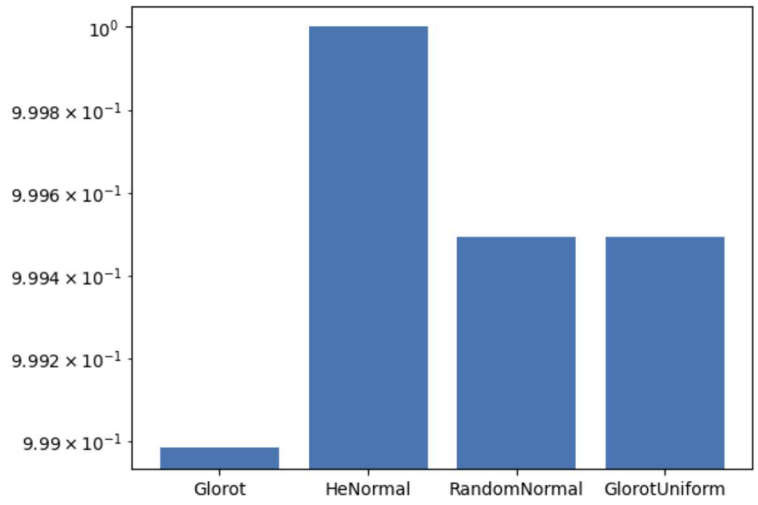
All of these initializers initializes the weights with a normal distribution centered around 0 but with difference in standard deviation. Glorot calculates standard deviation based on number of input units and output units, HE Normal calculates standard deviation with respect to number of input units and RandomNormal calculates std. as specified by the user. Here, He Normal performs the best with 100% accuracy. It performs better with ReLU activation function that has been used in the hidden layers.

3



HeNormal shows a less erractic behavior as compared to Glorot and RandomNormal.

HeNormal is also smoother after 10 epochs as compared to other initializers. RandomNormal is the most erratic throughout the epoch cycle.



In the above plot, HE Normal performs the best with the highest accuracy, followed by RandomNormal and GlorotUniform.

He Normal performs best with ReLU and Glorot performs best with sigmoid. Here, in the hidden layer ReLU is used, therefore, HeNormal performs better than the other.

TABLE for 2-Class Hyperparameters – With Four Different Weight Initializers

Activation function – hidden layer	ReLU
Activation function - output layer	Sigmoid
Weight Initializer	GlorotNormal(seed=1), HeNormal(seed=2), RandomNormal(mean=0.0, stddev=0.1, seed=3), GlorotUniform(seed=4
Number of hidden layers	1 (2 including the output layer)
Neurons in hidden layers	128
Neurons in the output layer	1
Loss function	Binary_crossentropy
Optimizer	Adam
Number of epochs	20
Batch size	32
Learning rate	0.001
Evaluation Metric	Accuracy

Q 3. 10 - class classification of MNIST

```
1 model.summary()
```

```
Model: "sequential_6"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 flatten_6 (Flatten)         (None, 784)               0

 dense_12 (Dense)            (None, 128)               100480

 dense_13 (Dense)            (None, 10)                1290

=================================================================
Total params: 101,770
Trainable params: 101,770
Non-trainable params: 0
```

```
]:  1 iling the model .We use sparse categorical that will hot encode the classes internally.
    2 compile(optimizer=tf.keras.optimizers.Adam(learning_rate=learning_rate), loss='sparse_categorical_crossentropy',metrics=['acc
```

```
]:  1 # Early stopping criteria based on validation loss
    2 # patience = Number of epochs with no improvement after which training will be stopped.
    3 # restore_best_weights = Whether to restore model weights from the epoch with the best value of the monitored quantity.
    4 early_stopping = EarlyStopping(monitor='val_loss', patience=10, restore_best_weights=True)
```

```
]:  1 # Training the model
    2 history = model.fit(x_train, y_train, batch_size=batch_size, epochs=20, validation_split=0.2, callbacks=[early_stopping])
```

```
Epoch 1/20
1500/1500 [==============================] - 2s 1ms/step - loss: 0.2875 - accuracy: 0.9175 - val_loss: 0.1539 - val_accuracy:
0.9578
Epoch 2/20
1500/1500 [==============================] - 2s 1ms/step - loss: 0.1271 - accuracy: 0.9634 - val_loss: 0.1077 - val_accuracy:
0.9690
Epoch 3/20
1500/1500 [==============================] - 2s 1ms/step - loss: 0.0879 - accuracy: 0.9744 - val_loss: 0.0967 - val_accuracy:
```

```
55]:  1 # Evaluate the model on the test data
      2 test_loss, test_accuracy = model.evaluate(x_test, y_test)
      3 print("Test Accuracy:", test_accuracy)
      4
```

```
313/313 [==============================] - 0s 772us/step - loss: 0.0789 - accuracy: 0.9781
Test Accuracy: 0.9781000018119812
```

CONFUSION MATRIX

```
[67]:  1 #draw_confusion_matrix(y_test, y_pred_classes,"classes")
       2 # Construct a confusion matrix
       3 conf_matrix = confusion_matrix(y_test, y_pred_classes)
       4 print("Confusion Matrix:")
       5 print(conf_matrix)
       6
```

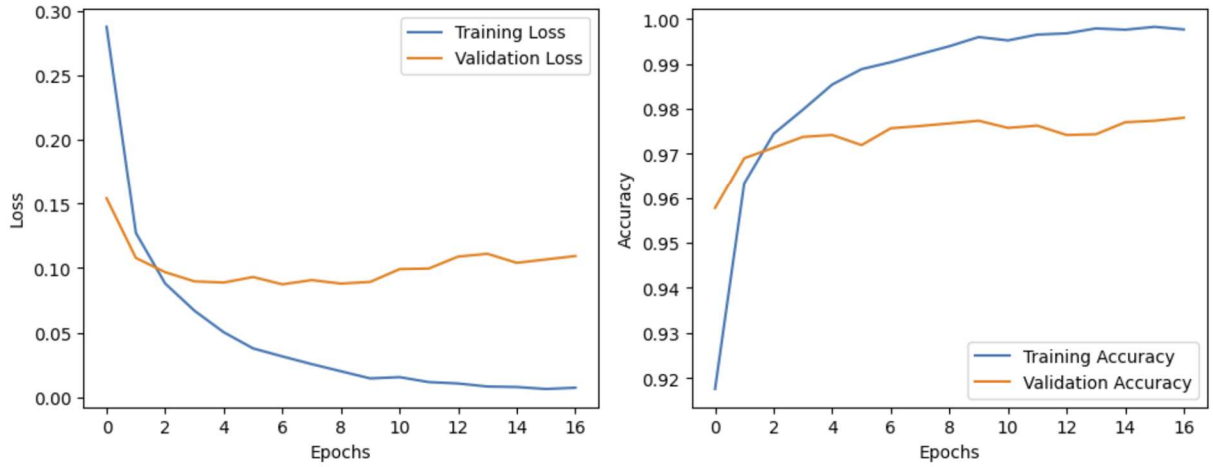
```
Confusion Matrix:
[[ 969    0    1    1    1    3    2    2    0]
 [   0 1120    4    2    0    1    3    1    4    0]
 [   2    0 1014    5    1    0    2    5    3    0]
 [   0    0    5  996    0    1    0    3    3    2]
 [   1    0    3    0  961    0    5    2    0   10]
 [   2    0    0   17    1  859    6    2    4    1]
 [   6    2    3    1    3    1  939    0    3    0]
 [   1    3   12    4    0    0    0 1002    3    3]
 [   3    0    5   11    4    1    1    4  942    3]
 [   1    3    0    6    7    3    0    9    1  979]]
```

As per the above confusion matrix, mostly all the classes from 0-9 are classified correctly. Class 0 and 1 are the least misclassified. Class 5 is misclassified as class 3 for 17 times; Class 7 has been misclaasified as 2 for 12 times; Class 8 is misclassified as 3 for 11 times.

```python
# Display a classification report
class_report = classification_report(y_test, y_pred_classes)
print("Classification Report:")
print(class_report)
```

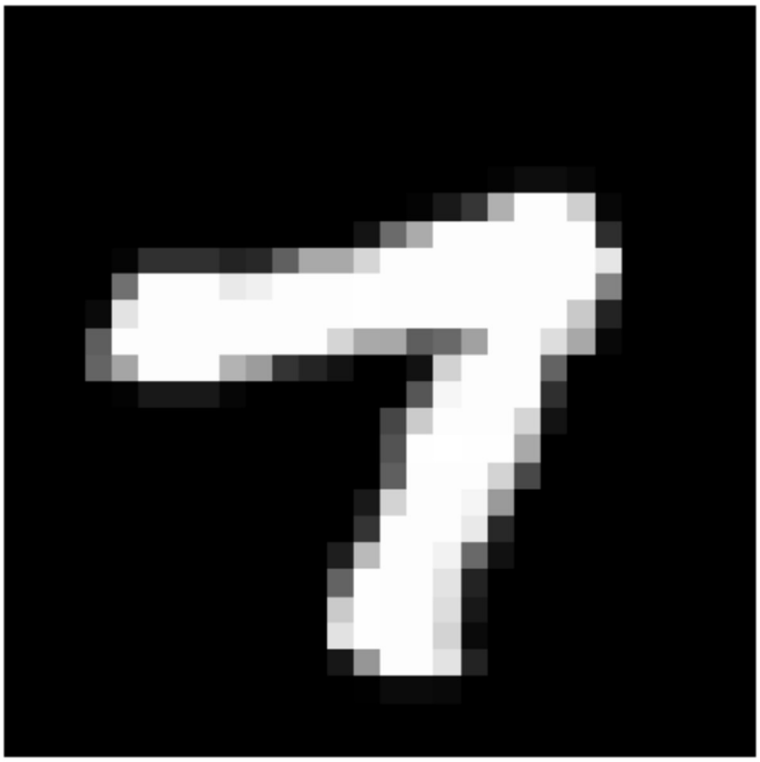
```
Classification Report:
              precision    recall  f1-score   support

           0       0.98      0.99      0.99       980
           1       0.99      0.99      0.99      1135
           2       0.97      0.98      0.98      1032
           3       0.95      0.99      0.97      1010
           4       0.98      0.98      0.98       982
           5       0.99      0.96      0.98       892
           6       0.98      0.98      0.98       958
           7       0.97      0.97      0.97      1028
           8       0.98      0.97      0.97       974
           9       0.98      0.97      0.98      1009

    accuracy                           0.98     10000
   macro avg       0.98      0.98      0.98     10000
weighted avg       0.98      0.98      0.98     10000
```

```python
# Plot the learning curve
plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['loss'], label='Training Loss')
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(1, 2, 2)
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()

plt.tight_layout()
plt.show()
```




```python
# Show some examples of predictions
eg_indices = np.random.choice(len(x_test), 10) # 10 randomly selected samples
eg_images = x_test[eg_indices]
eg_labels = y_test[eg_indices]
eg_predictions = model.predict(eg_images)
eg_predictions_classes = np.argmax(eg_predictions, axis=1)
for i in range(len(eg_images)):
    plt.figure()
    plt.imshow(eg_images[i], cmap='gray')
    plt.title(f"True Label: {eg_labels[i]}, Predicted: {eg_predictions_classes[i]}")
    plt.axis('off')

```

True Label: 7, Predicted: 7



True Label: 2, Predicted: 2

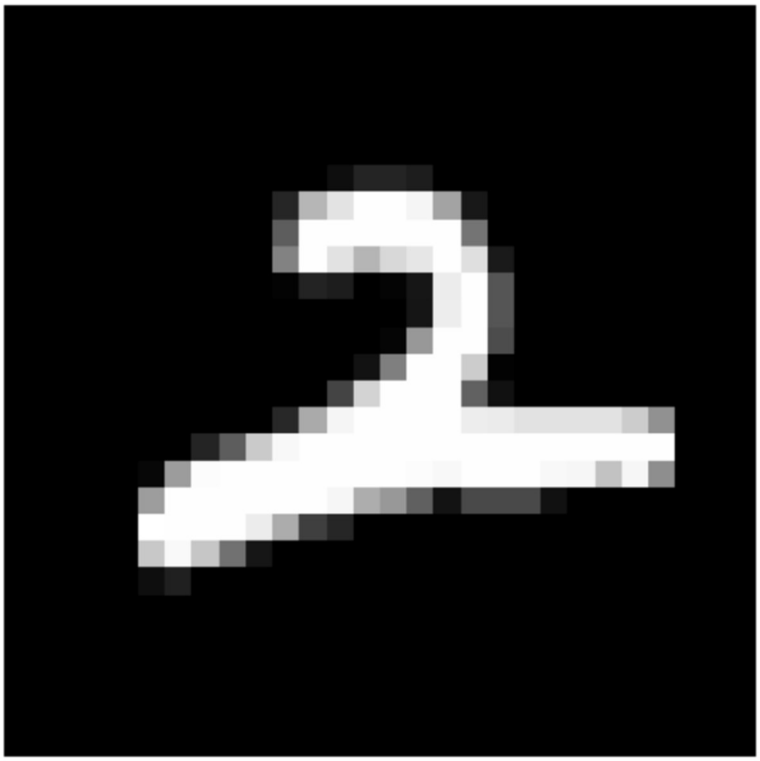


TABLE for 10 Class hyperparameters

Activation function – hidden layer	ReLU
Activation function - output layer	Softmax
Weight Initializer	Default - GlorotUniform
Number of hidden layers	1 (2 including the output layer)
Neurons in hidden layers	128
Neurons in the output layer	10
Loss function	sparse_categorical_crossentropy
Optimizer	Adam
Number of epochs	20
Batch size	32
Learning rate	0.001
Evaluation Metric	Accuracy