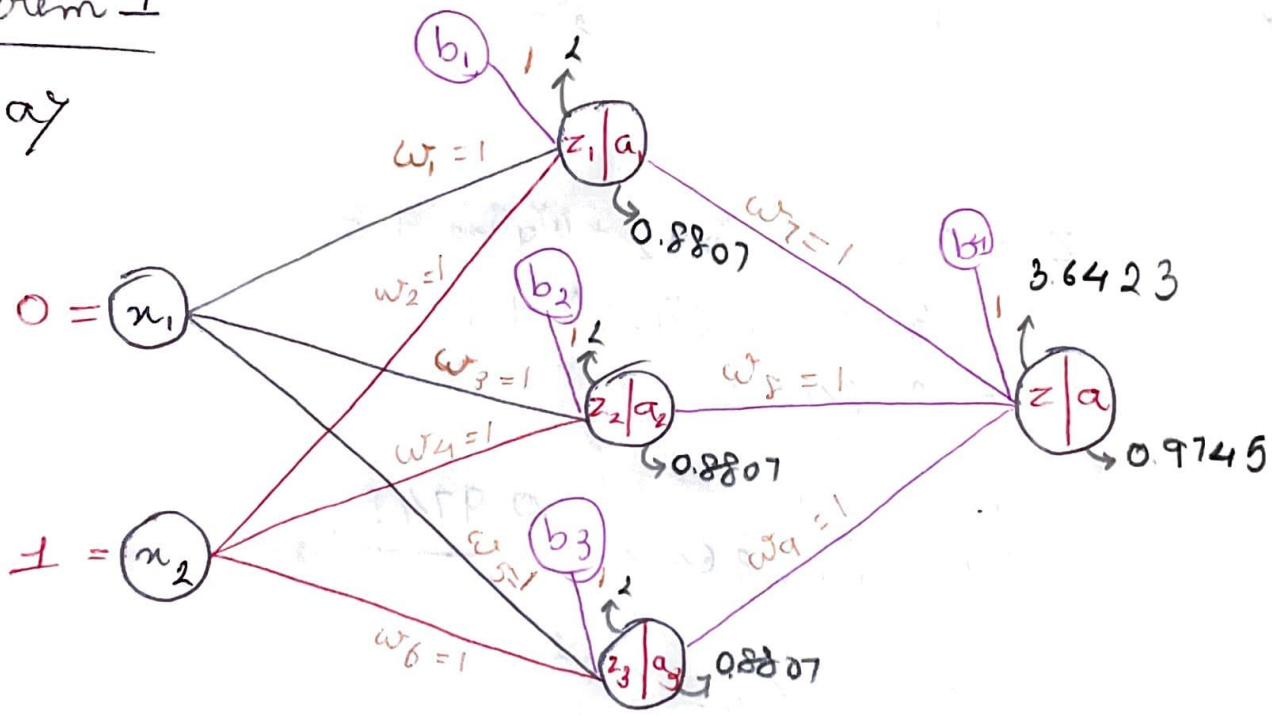
DATA - 255 Deep-Learning Technologies
HW 1 — Fall 24

Prayag Nikul Purani (01741737)

Problem 1

a)



Given

$x_1 = 0$ all weights & bias are = 1

$x_2 = 1$ $\alpha = 0.1$, $y = 1$

$$MSE = \frac{1}{2}(y - \hat{y})^2$$

activation = sigmoid.

No. of sample (n) = 1

$z_1 = w_1 x_1 + w_2 x_2 + b_1$

$= (1 \times 0) + (1 \times 1) + 1$

$= 2$

$a_1 = \frac{1}{1 + e^{-2}} = 0.8807$

$z_2 = w_3 x_1 + w_4 x_2 + b_2$

$= (1 \times 0) + (1 \times 1) + 1$

$= 2.$

$a_2 = \frac{1}{1 + e^{-2}} = 0.8807$

$z_3 = w_5 x_1 + w_6 x_2 + b_3$

$= (1 \times 0) + (1 \times 1) + 1$

$= 2$

$a_3 = \frac{1}{1 + e^{-2}} = 0.8807$

$$Z = w_7 a_1 + w_8 a_2 + w_9 a_3 + b$$

$$= 3 \times (1 \times 0.8807) + 1$$

$$= 3.6423$$

$$a/\hat{y} = \frac{1}{1 + e^{-3.6423}} = 0.9745.$$

$$L = \frac{1}{n} \times \frac{1}{2} (y - \hat{y})^2$$

$$= \frac{1}{2} (1 - 0.9745)^2 = 0.00032$$

Back propagation

$\rightarrow w_7 = 1$

$$w_7' = w_7 - \alpha \frac{\partial L}{\partial w_7}$$

$$\frac{\partial L}{\partial w_7} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_7}$$

$$\frac{\partial L}{\partial a} = \frac{1}{2}(y-a)^2$$
$$= \frac{2}{2}(y-a) - 1$$
$$= -(1 - 0.9745)$$
$$\frac{\partial L}{\partial a} = -0.0255$$

$$\frac{\partial z}{\partial w_7} = (w_7 a_1 + w_8 a_2 + w_9 a_3 + b_4)$$
$$= a_1 = 0.8807$$

$$\frac{\partial a}{\partial z} = a(1-a)$$
$$= 0.9745(1-0.9745)$$
$$= 0.0248.$$

$\Rightarrow w_7' = 1 - 0.1(0.8807 \times 0.0248 \times -0.025)$

$\boxed{w_7' = 1.000055}$

$\rightarrow w_8 = 1$

$$w_8' = w_8 - \alpha \frac{\partial L}{\partial w_8}$$

$$\frac{\partial L}{\partial w_8} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_8}$$

same

$$\frac{\partial z}{\partial w_8} = (w_7 a_1 + w_8 a_2 + w_9 a_3 + b_4)$$
$$= a_2 = 0.8807$$

$\Rightarrow w_8' = 1 - 0.1(5.57 \times 10^{-4})$

$\boxed{w_8' = 1.000055}$

$\rightarrow w_9 = 1$

$$w_9' = w_9 - \alpha \frac{\partial L}{\partial w_9}$$

$$\frac{\partial z}{\partial w_9} = a_3 = 0.8807$$

$$\frac{\partial L}{\partial w_9} = \frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_9}$$

same

$\boxed{w_9' = 1.000055}$

→ $w_1 = 1$

$$w_1' = w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}}_{\text{same.}} \times \frac{\partial z}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial z}{\partial a_1} = w_7 = 1 \; ; \; \frac{\partial a_1}{\partial z_1} = a_1(1-a_1) = 0.1051 \; ; \; \frac{\partial z_1}{\partial w_1} = x_1 = 0$$

$$w_1' = w_1 - 0.1(-0.0255 \times 0.0248 \times 1 \times 0.1051 \times 0)$$

$$\boxed{w_1' = w_1 = 1}$$

→ $w_2 = 1$

$$w_2' = w_2 - \alpha \frac{\partial L}{\partial w_2}$$

$$\frac{\partial L}{\partial w_2} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial a_1} \times \frac{\partial a_1}{\partial z_1}}_{\text{same.}} \times \frac{\partial z_1}{\partial w_2}$$

$$\frac{\partial z_1}{\partial w_2} = x_2 = 1$$

$$w_2' = 1 - 0.1(-0.0255 \times 0.0248 \times 1 \times 0.1051 \times 1)$$

$$\boxed{w_2' = 1.0000005}$$

→ $w_3 = 1$

$$w_3' = w_3 - \alpha \frac{\partial L}{\partial w_3}$$

$$\frac{\partial L}{\partial w_3} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}}_{\text{same}} \times \frac{\partial \varepsilon}{\partial a_2}^{1} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial w_3}^{0} = 0$$

$\frac{\partial a_2}{\partial z_2} \rightarrow 0.1051$

$$w_3' = w_3 - 0$$

$$\boxed{w_3' = w_3 = 1}$$

$\rightarrow w_4 = 1$

$$w_4' = w_4 - \alpha \frac{\partial L}{\partial w_4}$$

$$\frac{\partial L}{\partial w_4} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}}_{\text{same}} \times \frac{\partial z}{\partial a_2}\uparrow^{1} \times \frac{\partial a_2}{\partial z_2} \times \frac{\partial z_2}{\partial w_4}\uparrow^{1}$$

$\hookrightarrow 0.1051$

$$\boxed{w_4' = 1.0000005}$$

$\rightarrow w_5 = 1$

$$w_5' = w_5 - \alpha \frac{\partial L}{\partial w_5}$$

$$\frac{\partial L}{\partial w_5} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}}_{\text{same}} \times \frac{\partial z}{\partial a_3}\uparrow^{1} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial w_5}\uparrow^{0}$$

$\hookrightarrow 0.1051$

$$w_5' = w_5 - \alpha(0)$$

$$\boxed{w_5' = w_5 = 1}$$

$\rightarrow w_6 = 1$

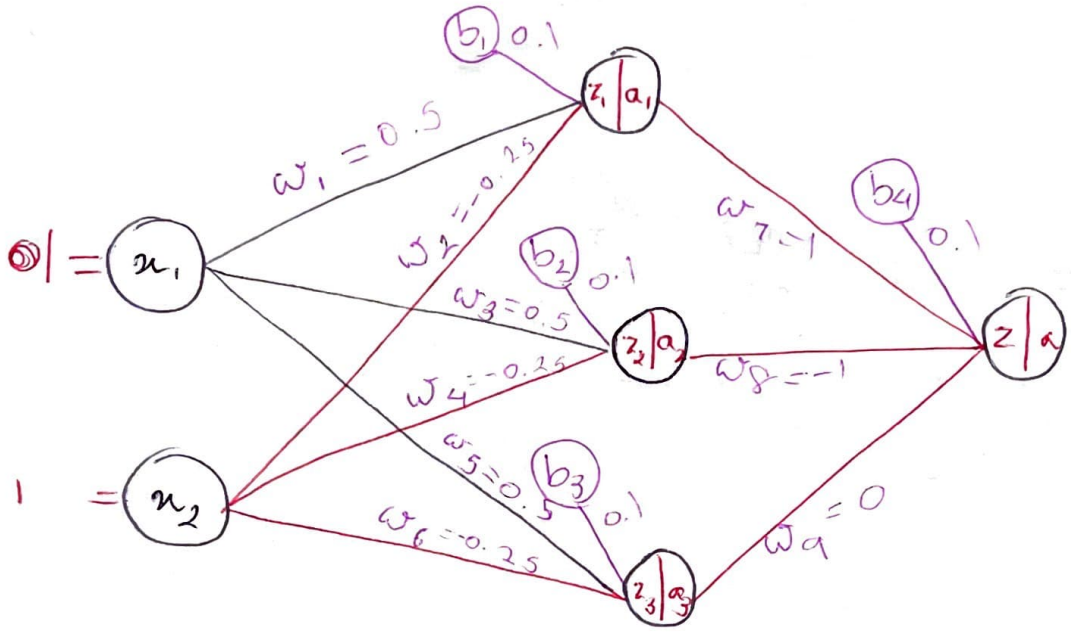
$$w_6' = w_6 - \alpha \frac{\partial L}{\partial w_6}$$

$$\frac{\partial L}{\partial w_6} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}}_{\text{same}} \times \frac{\partial z}{\partial a_3}\uparrow^{1} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial w_6}\uparrow^{1}$$

$\hookrightarrow 0.1051$

$$\boxed{w_6' = 1.0000005}$$

b)



$$z_1 = (w_1 \times x_1) + (w_2 \times x_2) + b_1$$
$$= (0.5 \times 1) + (-0.25 \times 1) + 0.1$$
$$= 0.35$$

$$a_1 = \frac{1}{1 + e^{-0.35}} = 0.5866$$

$$z_2 = (w_3 \times x_1) + (w_4 \times x_2) + b_2$$
$$= (0.5 \times 1) + (-0.25 \times 1) + 0.1$$
$$= 0.35$$

$$a_2 = \frac{1}{1 + e^{-0.35}} = 0.5866$$

$$z_3 = (w_5 \times x_1) + (w_6 \times x_2) + b_3$$
$$= (0.5 \times 1) + (-0.25 \times 1) + 0.1$$
$$= 0.35$$

$$a_3 = \frac{1}{1 + e^{-0.35}} = 0.5866$$

$$z = w_7 a_1 + w_8 a_2 + w_9 a_3 + b_4$$
$$= (1 \times 0.5866) + (0 \times 0.5866) + (-1 \times 0.5866) + 0.1$$
$$= 0.1$$

$$a = \frac{1}{1 + e^{-0.1}} = 0.5249$$

$$h = \frac{1}{1} \times \frac{1}{2} (y - a)^2$$
$$= \frac{1}{2} (1 - 0.5249)^2 = \underline{0.1128}$$

SJSU ID – 017416737

Old → w_1, w_5, w_7, w_9

1) $w_9 = 0$

$$w_9' = w_9 - \alpha \frac{\partial h}{\partial w_9}$$

$$\frac{\partial h}{\partial w_9} = \frac{\partial h}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_9}$$

$$\frac{\partial h}{\partial a} = \frac{\partial}{\partial a}\left[\frac{1}{2}(y-a)^2\right]$$
$$= -(y-a)$$
$$= -(1-0.5249)$$
$$= -0.4751$$

$$\frac{\partial z}{\partial w_9} = \frac{\partial}{\partial w_9}(w_7 a_1 + w_8 a_2 + w_9 a_3 + b_4)$$
$$= a_3 = 0.5866$$

$$\frac{\partial a}{\partial z} = a(1-a)$$
$$= 0.5249(1-0.5294)$$
$$= 0.249$$

$$w_9' = 0 - 0.1(-0.4751 \times 0.249 \times 0.5866)$$

$$\boxed{w_9' = 0.00693}$$

2) $w_7 = 1$

$$w_7' = w_7 - \alpha \frac{\partial h}{\partial w_7}$$

$$\frac{\partial z}{\partial w_7} = (w_7 a_1 + w_8 a_2 + w_9 a_3 + b_4)$$
$$= a_1 = 0.5866$$

$$\frac{\partial h}{\partial w_7} = \frac{\partial h}{\partial a} \times \frac{\partial a}{\partial z} \times \frac{\partial z}{\partial w_7}$$

same

$$w_7' = w_7 - \alpha(-0.4751 \times 0.249 \times 0.5866)$$

$$\boxed{w_7' = 1.00693}$$

3) $w_5 = 0.5$

$$w_5' = w_5 - \alpha \frac{\partial L}{\partial w_5}$$

$$\frac{\partial L}{\partial w_5} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}}_{\text{same}} \times \frac{\partial z}{\partial a_3} \times \frac{\partial a_3}{\partial z_3} \times \frac{\partial z_3}{\partial w_5}$$

$$\frac{\partial z}{\partial a_3} = w_4 = 0, \quad \frac{\partial a_3}{\partial z_3} = a_3(1-a_3) = 0.2425, \quad \frac{\partial z_3}{\partial w_5} = x_1$$

$$w_5' = 0.5 - 0.1(-0.4751 \times 0.249 \times 0 \times 0.2425 \times 1)$$

$$\boxed{w_5' = 0.5}$$

4) $w_1 = 0.5$

$$w_1' = w_1 - \alpha \frac{\partial L}{\partial w_1}$$

$$\frac{\partial L}{\partial w_1} = \underbrace{\frac{\partial L}{\partial a} \times \frac{\partial a}{\partial z}}_{\text{same}} \times \frac{\partial z}{\partial a_1} \times \frac{\partial a_1}{\partial z_1} \times \frac{\partial z_1}{\partial w_1}$$

$$\frac{\partial z}{\partial a_1} = w_7 = 1, \quad \frac{\partial a_1}{\partial z_1} = a_1(1-a_1) = 0.2425, \quad \frac{\partial z_1}{\partial w_1} = x_1$$

$$w_1' = 0.5 - 0.1(-0.4751 \times 0.249 \times 1 \times 0.2425 \times 1)$$

$$\boxed{w_1' = 0.50286}$$

Fall 2023 DATA 225 Deep Learning Technologies

Homework – 1

Name :- Prayag Nikul Purani

SJSU Id :- 017416737

Problem 2 - (Coding):-

➔ Loading data set.

Loading the data

```
In [2]: (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()
```

➔ Next step is to filter out the labels based on the last digit of SJSU ID which is 3 & 7

Filtering Based on last two digits 3 , 7

```
In [3]: y_train_filter = np.isin(y_train,[3,7])
        y_test_filter = np.isin(y_test,[3,7])
```

➔ Splitting the dataset into test train.

```
In [3]: y_train_filter = np.isin(y_train,[3,7])
        y_test_filter = np.isin(y_test,[3,7])

In [4]: y_train1 = y_train[y_train_filter]
        y_test1 = y_test[y_test_filter]

In [5]: x_train1 = x_train[y_train_filter]
        x_test1 = x_test[y_test_filter]

In [6]: print("Shape of X_train",x_train1.shape)
        print("Shape of X_test",x_test1.shape)
        print("Shape of y_train",y_train1.shape)
        print("Shape of y_test",y_test1.shape)

        Shape of X_train (12396, 28, 28)
        Shape of X_test (2038, 28, 28)
        Shape of y_train (12396,)
        Shape of y_test (2038,)
```

➔ Preprocessing scaling image between 0 and 1 and changing the shape as (28,28,1)

Preprocessing the Data

```
: # Scale images to the [0, 1] range
  x_train1 = x_train1.astype("float32") / 255
  x_test1 = x_test1.astype("float32") / 255
  # Make sure images have shape (28, 28, 1)
  x_train1 = np.expand_dims(x_train1, -1)
  x_test1 = np.expand_dims(x_test1, -1)

: y_train1 = np.where(y_train1==7,1,0)
  y_test1 = np.where(y_test1==7,1,0)
```

A. Build NN for binary classification with early stopping.

```
input_shape = (28,28,1)
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(1, activation="sigmoid"),
    ]
)
```

```
model_summary = model.summary()
print(model_summary)
```

```
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)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 5, 5, 64)	0
flatten (Flatten)	(None, 1600)	0
dropout (Dropout)	(None, 1600)	0
dense (Dense)	(None, 1)	1,601

```
Total params: 20,417 (79.75 KB)

Trainable params: 20,417 (79.75 KB)

Non-trainable params: 0 (0.00 B)

None
```

Table for Hyperparameters

Hyperparameter	Values
Activation Function	Relu, SoftMAx
Weight Initializer	he_normal, glorot_uniform, random_normal
Number of hidden layers	2
Number of neurons in hidden layers	32+64+64 = 160
Loss Function	categorical_crossentropy
Optimizer	adam
Number of Epochs	20
Learning Rate	0.001
Batch Size	32
Evaluation Metrics	Accuracy

➔ Early stopping criteria based

```
# Import early stopping from keras callbacks
from tensorflow.keras.callbacks import EarlyStopping
early_stopping = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
```

```
history2 = model.fit(x_train1, y_train1, batch_size= 512, epochs= 20, validation_split= 0.20,callbacks = [early_stopping])
```

```
Epoch 1/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 4s 173ms/step - accuracy: 0.9994 - loss: 0.0024 - val_accuracy: 1.0000 - val_loss: 5.1505e-04
Epoch 2/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 152ms/step - accuracy: 0.9990 - loss: 0.0029 - val_accuracy: 1.0000 - val_loss: 5.1404e-04
Epoch 3/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.9995 - loss: 0.0021 - val_accuracy: 1.0000 - val_loss: 5.3665e-04
Epoch 4/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 153ms/step - accuracy: 0.9997 - loss: 0.0018 - val_accuracy: 1.0000 - val_loss: 5.2478e-04
Epoch 5/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 154ms/step - accuracy: 0.9997 - loss: 0.0015 - val_accuracy: 1.0000 - val_loss: 5.2788e-04
Epoch 6/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 149ms/step - accuracy: 0.9995 - loss: 0.0036 - val_accuracy: 1.0000 - val_loss: 4.9431e-04
Epoch 7/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - accuracy: 0.9996 - loss: 0.0017 - val_accuracy: 1.0000 - val_loss: 5.1073e-04
Epoch 8/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 149ms/step - accuracy: 0.9991 - loss: 0.0024 - val_accuracy: 1.0000 - val_loss: 5.8417e-04
Epoch 9/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 155ms/step - accuracy: 0.9994 - loss: 0.0018 - val_accuracy: 1.0000 - val_loss: 5.5305e-04
Epoch 10/20
20/20 ━━━━━━━━━━━━━━━━━━━━ 3s 154ms/step - accuracy: 0.9995 - loss: 0.0016 - val_accuracy: 1.0000 - val_loss: 5.1822e-04
Epoch 10: early stopping
```

So, after applying early stopping condition with hyperparameters such as

Monitor = ‘val_loss’

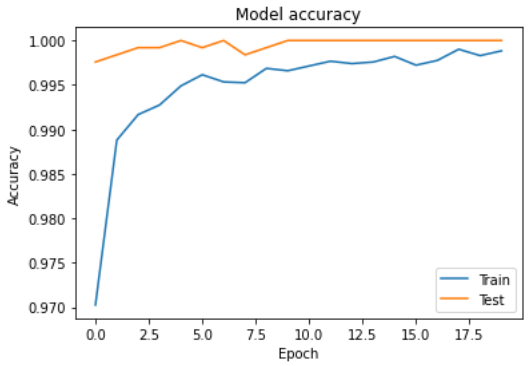
Mode = ‘min’

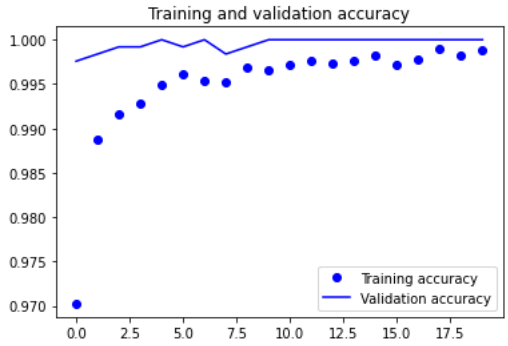
Verbose = 1

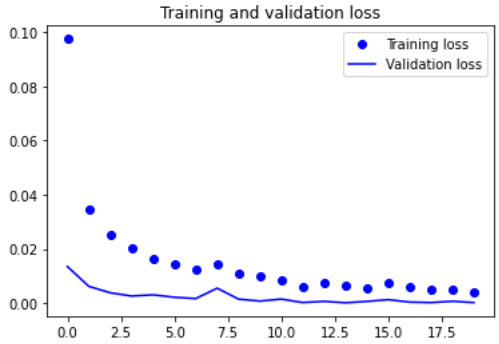
Patience = 4

And we can see that the total number of epoch is 20 but the model stops at 10 so we will be saving time and computational power to half the amount that was used in original condition.

➔ Learning curves







➔ Confusion Matrix

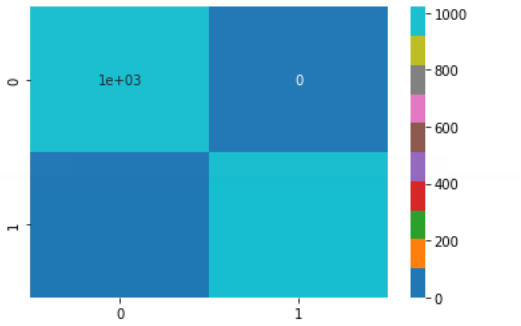
```
# importing the module
from sklearn.metrics import confusion_matrix

#printing the confusion matrix
prediction = model.predict(x_test1)
prediction = np.where(prediction>0.9,1,0)
cm = confusion_matrix(y_test1, prediction)
print(cm)
```

```
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
[[1010    0]
 [   7 1021]]
```

```
import seaborn as sns
```

```
# printing the confusion matrix in heatmap to visua
sns.heatmap(cm,annot=True, cmap= 'tab10')
```

```
<Axes: >
```



➔ Outputs predictions

```
actual value is 1
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 36ms/step
predicted value is 1
```

```
actual value is 0
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 40ms/step
predicted value is 0
```

For more outputs I would recommend to use ipynb file for better outputs.

B. Three different weight initializers

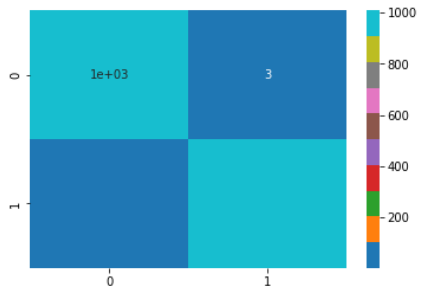
a. Three different confusion matrix

→Confusion Matrix for glorot uniform

```
: prediction = list_model[0].predict(x_test1)
  prediction = np.where(prediction>0.9,1,0)
  cm = confusion_matrix(y_test1, prediction)
  print(cm)
  # printing the confusion matrix in heatmap to visualise results for model with glorot_uniform kernel and glorot_uniform bias
  sns.heatmap(cm,annot=True, cmap= 'tab10')
```

```
64/64 ━━━━━━━━━━━━━━━━━━━━ 1s 7ms/step
[[1007    3]
 [  34  994]]
```

```
: <Axes: >
```



Glorot Uniform Kernel & Glorot Uniform Bias:

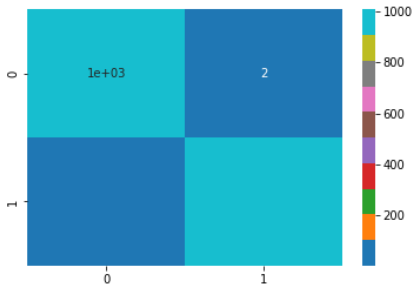
- True Negatives (TN): 1007
- False Positives (FP): 3
- False Negatives (FN): 34
- True Positives (TP): 994

→ Confusion Matrix for random uniform

```
prediction = list_model[1].predict(x_test1)
prediction = np.where(prediction>0.9,1,0)
cm = confusion_matrix(y_test1, prediction)
print(cm)
# printing the confusion matrix in heatmap to visualise results for model with random_uniform kernel and random_uniform bias
sns.heatmap(cm,annot=True, cmap= 'tab10')
```

```
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step
[[1008    2]
 [  28 1000]]
```

```
<Axes: >
```



Random Uniform Kernel & Random Uniform Bias:

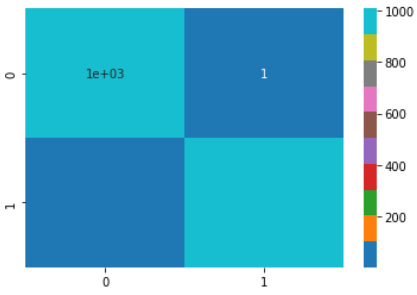
- True Negatives (TN): 1008
- False Positives (FP): 2
- False Negatives (FN): 28
- True Positives (TP): 1000

→ Confusion Matrix for random normal

```
prediction = list_model[2].predict(x_test1)
prediction = np.where(prediction>0.9,1,0)
cm = confusion_matrix(y_test1, prediction)
print(cm)
# printing the confusion matrix in heatmap to visualise results for model with random_normal kernel and random_normal bias
sns.heatmap(cm,annot=True, cmap= 'tab10')
```

```
64/64 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
[[1009    1]
 [  29  999]]
```

```
<Axes: >
```



Random Normal Kernel & Random Normal Bias:

- True Negatives (TN): 1009
- False Positives (FP): 1
- False Negatives (FN): 29
- True Positives (TP): 999

→ Comparison

Differences between the matrices:

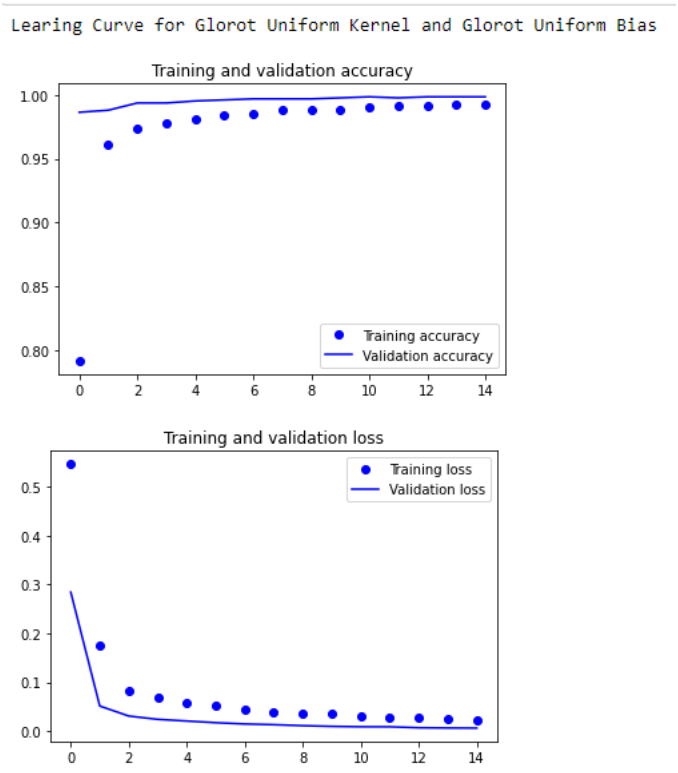
1. Glorot Uniform (first matrix) has slightly more false negatives (34) and a lower true positive rate (994).

2. Random Uniform (second matrix) and Random Normal (third matrix) perform better in terms of slightly fewer false negatives and a higher true positive count (1000 and 999, respectively).
3. The model with Random Normal Kernel and Random Normal Bias performs the best overall, with the fewest false positives (1) and a relatively low number of false negatives (29).

In conclusion, while all models perform well, the Random Normal Kernel and Bias model shows the best balance between minimizing false positives and false negatives.

b. Three different learning curve.

→ Confusion Matrix for glorot uniform

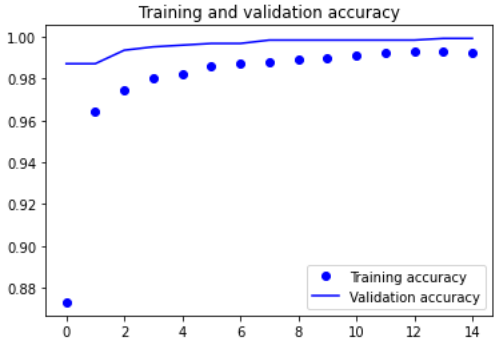


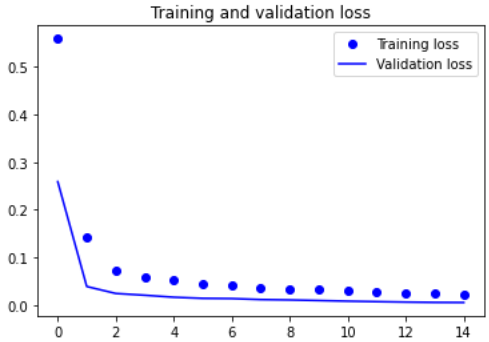
Accuracy: Training accuracy is almost at 100% from the beginning and remains stable. The validation accuracy, while improving, plateaus around 98%, indicating a slight gap between training and validation accuracy, but not too large.

Loss: Both training and validation loss decrease rapidly in the first few epochs, stabilizing at very low values, indicating good model performance without significant overfitting.

→ Confusion Matrix for random uniform

Learing Curve for Random Uniform Kernel and Random Uniform Bias



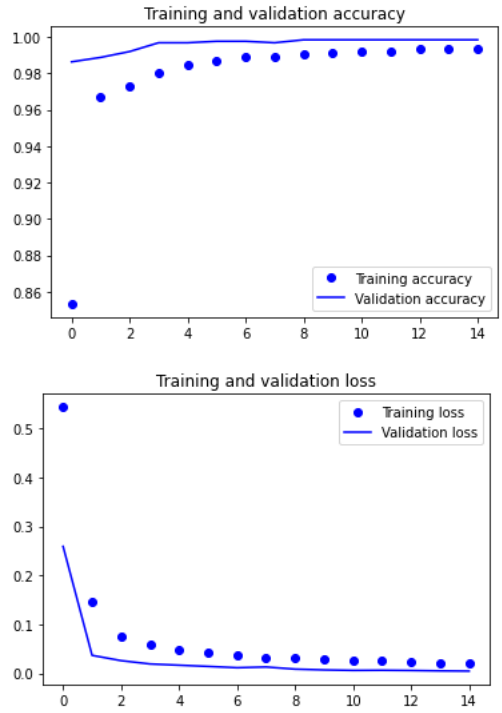


Accuracy: Training accuracy is high from the beginning, similar to the Glorot Uniform, but the validation accuracy is also very close to training accuracy. This suggests better generalization, as the gap between training and validation accuracy is smaller.

Loss: The loss decreases rapidly for both training and validation, indicating efficient learning. Validation loss is slightly lower compared to Glorot Uniform, suggesting this kernel and bias combination generalizes better.

→ Confusion Matrix for random normal

Learing Curve for Random Normal Kernel and Random Normal Bias



Accuracy: Training accuracy is near 100%, but the validation accuracy is slightly lower than the Random Uniform model. The gap between training and validation accuracy is minimal, suggesting this model has a good generalization ability.

Loss: The training and validation losses drop quickly and stabilize, similar to the other models, but the validation loss is slightly lower than Glorot Uniform and on par with Random Uniform.

→ Comparision

Yes, based on the visual analysis, the models are indeed performing quite similarly in terms of both accuracy and loss. The differences in performance between the models are minimal, and all models show:

- High training accuracy near 100%.
- Validation accuracy converging around the 98-99% range.
- Training and validation losses reducing rapidly and stabilizing at very low values.

These minor differences can be due to the randomness in kernel initialization (Glorot, Random Uniform, Random Normal), but they don't lead to significant variations in the model's overall performance. In practical terms, the models are nearly equivalent, and their generalization abilities are very close.

So assessment is — the models are performing nearly the same

Problem 3 - (Coding):-

➔ Importing Libraries and reshaping images

```
from tensorflow.keras.datasets import mnist
from tensorflow.keras.utils import to_categorical

(train_images, train_labels), (test_images, test_labels) = mnist.load_data()

train_images = train_images.reshape((60000, 28, 28, 1)).astype('float32') / 255
test_images = test_images.reshape((10000, 28, 28, 1)).astype('float32') / 255

train_labels = to_categorical(train_labels)
test_labels = to_categorical(test_labels)
```

➔ Models creation

```
from tensorflow.keras import layers, models

model = models.Sequential([
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(28, 28, 1)),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')  # 10 output units for 10 classes
])

model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

Table for hyperparameters

Hyperparameter	Values
Activation Function	Relu, SoftMAx
Weight Initializer	he_normal
Number of hidden layers	2
Number of neurons in hidden layers	32+64+64 = 160
Loss Function	categorical_crossentropy
Optimizer	adam
Number of Epochs	20
Learning Rate	0.001
Batch Size	32
Evaluation Metrics	Accuracy

➔ Early Stopping condition

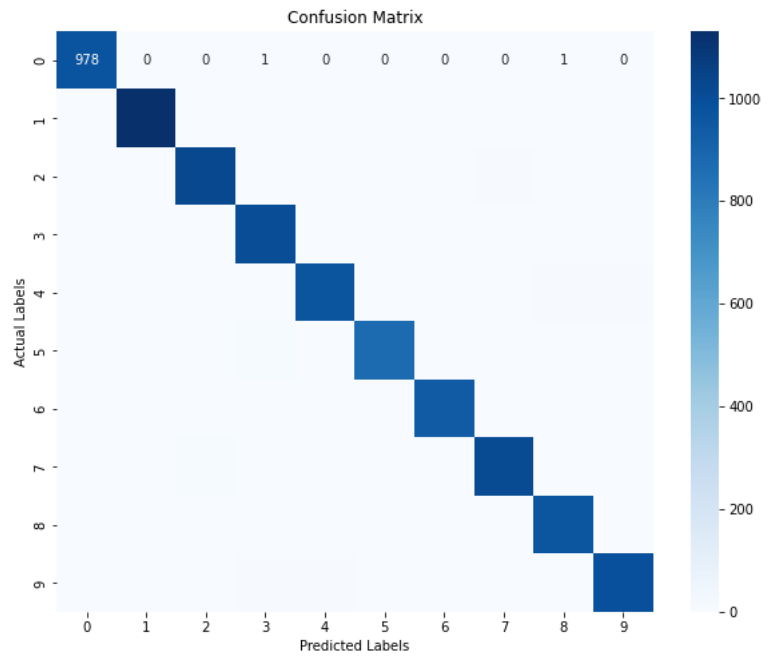
```
from tensorflow.keras.callbacks import EarlyStopping

early_stopping = EarlyStopping(monitor='val_loss', patience=3)
```

```
history = model.fit(train_images, train_labels, epochs=20, validation_split=0.1, callbacks=[early_stopping])
```

```
Epoch 1/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 38s 21ms/step - accuracy: 0.9130 - loss: 0.2834 - val_accuracy: 0.9877 - val_loss: 0.0476
Epoch 2/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 32s 19ms/step - accuracy: 0.9847 - loss: 0.0487 - val_accuracy: 0.9898 - val_loss: 0.0404
Epoch 3/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 33s 19ms/step - accuracy: 0.9911 - loss: 0.0268 - val_accuracy: 0.9880 - val_loss: 0.0432
Epoch 4/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 33s 19ms/step - accuracy: 0.9940 - loss: 0.0191 - val_accuracy: 0.9882 - val_loss: 0.0480
Epoch 5/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 35s 21ms/step - accuracy: 0.9961 - loss: 0.0123 - val_accuracy: 0.9897 - val_loss: 0.0367
Epoch 6/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 38s 19ms/step - accuracy: 0.9971 - loss: 0.0091 - val_accuracy: 0.9905 - val_loss: 0.0431
Epoch 7/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 33s 20ms/step - accuracy: 0.9975 - loss: 0.0077 - val_accuracy: 0.9902 - val_loss: 0.0395
Epoch 8/20
1688/1688 ━━━━━━━━━━━━━━━━━━━━ 40s 19ms/step - accuracy: 0.9978 - loss: 0.0060 - val_accuracy: 0.9903 - val_loss: 0.0433
```

➔ Confusion matrix .

```
313/313 ━━━━━━━━━━━━━━━━━━━━ 2s 5ms/step
```




```
conf_matrix
```

```
array([[ 973,    0,    2,    1,    1,    0,    3,    0,    0,    0],
       [   1, 1130,    1,    2,    0,    0,    1,    0,    0,    0],
       [   0,    3, 1023,    0,    0,    0,    0,    6,    0,    0],
       [   0,    0,    1, 1005,    0,    3,    0,    0,    1,    0],
       [   0,    0,    0,    0,  977,    0,    1,    0,    0,    4],
       [   1,    1,    0,    8,    0,  879,    2,    0,    1,    0],
       [   3,    2,    0,    0,    1,    3,  947,    0,    2,    0],
       [   0,    2,    7,    1,    0,    0,    0, 1010,    0,    8],
       [   2,    0,    2,    1,    0,    0,    0,    2,  965,    2],
       [   0,    0,    1,    1,    5,    3,    0,    0,    1,  998]])
```

This is a confusion matrix for a multi-class classification problem where the diagonal elements represent the number of correctly classified examples, and the off-diagonal elements represent the misclassified examples.
Diagonal elements: Each diagonal element (e.g., 978 for class 0) indicates how many instances of a particular class were correctly classified. For example, 978 instances of class 0 were predicted correctly.
Off-diagonal elements: The numbers that are not on the diagonal represent misclassified examples. For instance, if the number 1 appears at row 0 and column 3, it means 1 instance of class 0 was predicted as class 3.

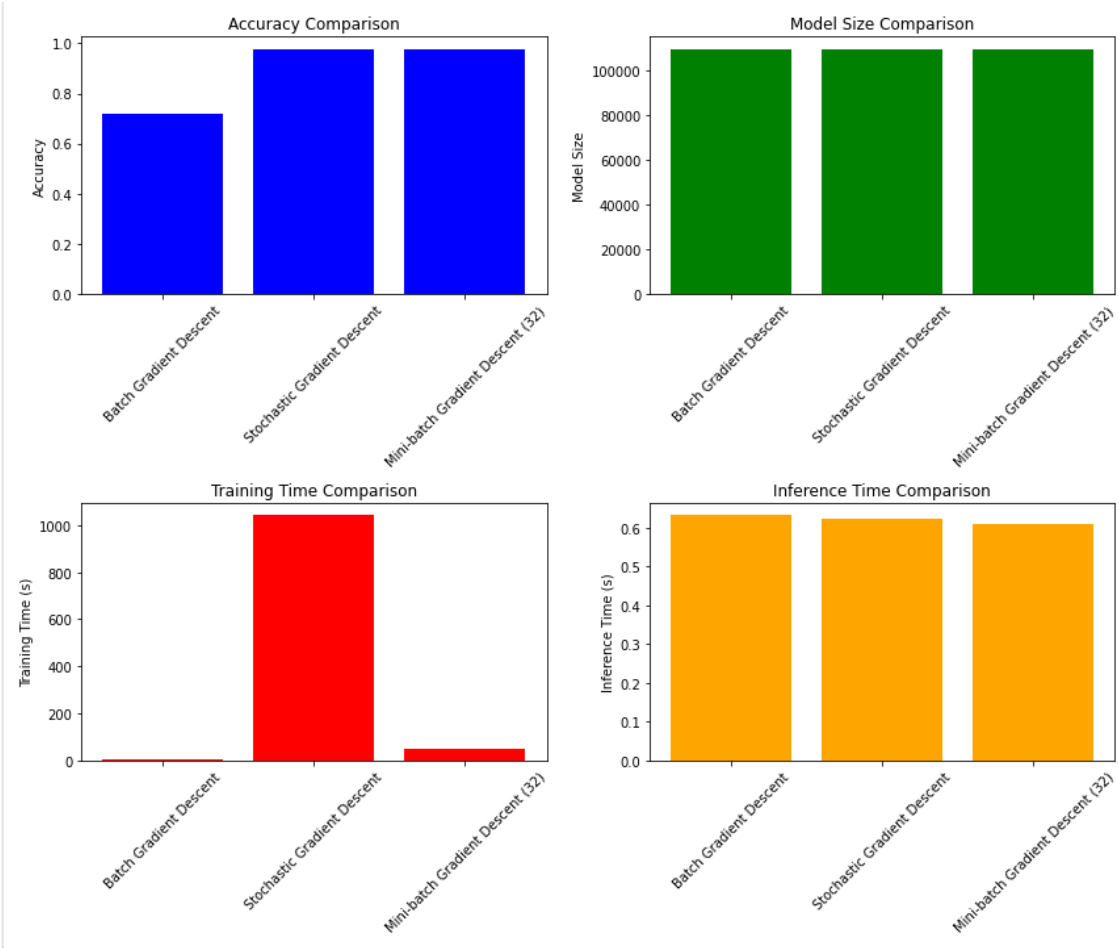
Class 0: 973 out of 1000 were correctly classified, with a few misclassified as other classes.
Class 1: 1130 instances were classified correctly, showing a very high precision for this class.
Class 8: Slightly more misclassifications here—965 correct, but 7 were misclassified as class 3 and 2 as class 5.

Both confusion matrices show that the model performs very well, as most values are on the diagonal, indicating a high number of correct classifications. The errors (off-diagonal values) are relatively sparse, meaning there are only a few misclassifications spread across different classes.

Problem 4 - (Coding):-

➔ Outputs

```
                          Method  Accuracy  Model Size  Training Time (s)  \
0           Batch Gradient Descent    0.7195      109386           6.090448
1      Stochastic Gradient Descent    0.9760      109386        1042.907096
2  Mini-batch Gradient Descent (32)    0.9772      109386          47.715127

   Inference Time (s)
0            0.633010
1            0.625026
2            0.609375
```



Accuracy Comparison:

Batch Gradient Descent: Has an accuracy of 0.7195 (around 72%), which is lower than the other two methods.

Stochastic Gradient Descent (SGD): Achieves an accuracy of 0.9760 (around 98%), which is significantly better.

Mini-batch Gradient Descent (Batch size = 32): Has an accuracy of 0.9772, very close to SGD but slightly better.

Training Time Comparison:

Batch Gradient Descent: Has the shortest training time at 6.09 seconds, but this comes with the tradeoff of lower accuracy.

Stochastic Gradient Descent: Takes the longest training time, 1042.91 seconds, likely because SGD updates weights after every data point (leading to more frequent but smaller updates).

Mini-batch Gradient Descent: Performs much better than SGD in terms of training time at 47.71 seconds. This is expected as mini-batch optimizes between the extremes of batch and stochastic methods by using smaller groups of data per update, speeding up the training process without sacrificing accuracy significantly.

Inference Time Comparison:

- Batch Gradient Descent: Inference time is 0.633 seconds, the highest among the three methods.
- Stochastic Gradient Descent: Inference time is 0.625 seconds, slightly better than Batch Gradient Descent.
- Mini-batch Gradient Descent: Performs the best in terms of inference time at 0.609 seconds, making it the fastest for predictions.

Model Size Comparison:

- All three methods have the same model size, 109386 parameters. This makes sense because gradient descent methods only affect how the model is trained, not the structure or size of the model itself.

Key Takeaways:

1. Mini-batch Gradient Descent offers the best balance between accuracy, training time, and inference time. It reaches a high accuracy of 97.72% with much faster training and inference times compared to SGD.
2. Stochastic Gradient Descent (SGD) achieves similar accuracy to Mini-batch but at a much higher training time, which makes it less efficient for this task.
3. Batch Gradient Descent, while having the fastest training time, significantly underperforms in terms of accuracy, making it less useful in scenarios where high accuracy is needed.

Thus, based on this analysis, Mini-batch Gradient Descent (with a batch size of 32) appears to be the most effective method for this particular problem, as it balances the trade-offs between speed and performance.