Assignment-3: Neural Network

Sajan Kumer Sarker
Id: 2111131642
Email: sajan.sarker@northsouth.edu

Rafsan Jani Chowdhury
Id: 2011424642
Email: rafsan.chowdhury@northsouth.edu

Rosely Mohammad
Id: 2014219642
Email: rosely.mohammad@northsouth.edu

October 21, 2024

I. Introduction

In this Assignment report on Neural Network we will discuss about training and evaluation of the Neural Network model on the provided dataset of hand sign digits. Here, we will construct a Neural Network model using a preferred architecture determined through trail and error. We'll train the model on the provided dataset and for testing the model we will build a new dataset. For the model we'll report overall model accuracy, per class accuracy and MSE. We will build the test dataset by following a resource which will be given on the reference section. [1].

II. Methods

A. Building the Neural Network

Necessary Imports:

```
import numpy as np
import pandas as pd
import tensorflow as tf
from tensorflow.keras import regularizers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.activations import linear, relu, sigmoid
from tensorflow.keras.losses import BinaryCrossentropy, SparseCategoricalCrossentropy
from tensorflow.keras.callbacks import EarlyStopping
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, mean_squared_error
```

Imported all the necessary libraries and modules that will be used in the project.

Checking for Available GPUs in TensorFlow:

```
physical_devices = tf.config.experimental.list_physical_devices('GPU')
print("Num GPUs Available: ", len(physical_devices))

# Optionally print device name for verification
if len(physical_devices) > 0:
    print("GPU device name: ", physical_devices[0])
```

We've manually selected Python 3 Google Compute Engine backend (GPU) and we make sure we use the GPU to Train the model.

Loading The Training Dataset:

```
X = np.load('/content/handsignX.npy')
y = np.load('/content/handsigny.npy')

print('The first element of X is:', X[0])
print('The first element of y is:', y[0,0])
print ('The last element of y is:', y[-1,0])
print ('The shape of X is: ' + str(X.shape))
print ('The shape of y is: ' + str(y.shape))
```

Here, we load the Training Dataset, where X contain the instances and y contain the target. Then it outputs the first element of X, first and last elements of y and displays the shapes of both arrays to provide an overview of the dataset.

Visualizing Training Dataset:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

m, n = X.shape

fig, axes = plt.subplots(8,8, figsize=(5,5))
fig.tight_layout(pad=0.13,rect=[0, 0.03, 1, 0.91]) #[left, bottom, right, top]

#fig.tight_layout(pad=0.5)

for i,ax in enumerate(axes.flat):
    # Select random indices
    random_index = np.random.randint(m)

    # Select rows corresponding to the random indices and
    # reshape the image
    X_random_reshaped = X[random_index].reshape((20,20)).T

    # Display the image
    ax.imshow(X_random_reshaped, cmap='gray')

    # Display the label above the image
    ax.set_title(y[random_index,0])
    ax.set_axis_off()
    fig.suptitle("Label, image", fontsize=14)
```

This code randomly selects some images from the dataset X, then reshaping the image into (500x20x20) pixels and displaying it in grayscale. Each image is displayed alongside its corresponding label from y, with the title "Label, image" shown at the top of the grid/image.

Model Design and Training:

```
#Neural Network
tf.random.set_seed(1234) # for consistent results

model = Sequential([
    tf.keras.Input(shape=(400,)),
    Dense(units=25, activation='relu', kernel_regularizer=regularizers.l2(0.01),
    name='hidden_layer1'),
    Dense(units=15, activation='relu', kernel_regularizer=regularizers.l2(0.001),
```

```
    name='hidden_layer2'),
    Dense(units=10, activation='relu', kernel_regularizer=regularizers.l2(0.0001),
    name='hidden_layer3'),
    Dense(units=10, activation='linear')
])
model.summary()

# model training
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-3),
    loss=SparseCategoricalCrossentropy(from_logits=True)
)

model.fit(
    X, y,
    batch_size=500,
    epochs=500,
    callbacks=[
        tf.keras.callbacks.EarlyStopping(
            monitor='loss',
            patience=500,
            restore_best_weights=True
        )
    ]
)
```

We build a Neural Network Mode and Train the model. The model contain 400 units input layer, because the dataset contain 400 (20x20) features. Then the first Hidden layer contain 25 units, second hidden layer contain 15 units, and the third hidden layer contain 10 units, and each hidden layer used ReLU as activation function and L2 regularization. The last layer contain 10 units and Linear activation function which is the output layer. In order to address over-fitting issue we regularized the 3 hidden layer with decreasing regularizing hyper parameter 0.01 to 0.0001. The model give best performance on the training dataset. Then we compile the model with Adam optimizer with 1e-3 learning rate and sparse categorical cross-entropy loss. Then we train the model using the on the dataset X and y for up to 500 epochs with 500 batch size. It includes an early stopping mechanism that monitors the training loss and halts the training if no improvement is seen for 500 epochs, while restoring the best model weights.

Prediction Inference for single instance:

```
image_of_two = X[1015]
fig, ax = plt.subplots(1,1, figsize=(0.5,0.5))
X_reshaped = image_of_two.reshape((20,20)).T

# Display the image
ax.imshow(X_reshaped, cmap='gray')

prediction = model.predict(image_of_two.reshape(1,400))  # prediction
#print(f" predicting a Two: \n{prediction}")
#print(f" Largest Prediction index: {np.argmax(prediction)}")

prediction_p = tf.nn.softmax(prediction)
# print(f" predicting a Two. Probability vector: \n{prediction_p}")
# print(f"Total of predictions: {np.sum(prediction_p):0.3f}")

yhat = np.argmax(prediction_p)
# print(f"np.argmax(prediction_p): {yhat}")

ax.set_title(f"Actual: {y[1015]}, Prediction: {yhat}")
plt.show()
```

We pick a image that is written '2' from the dataset and predict the digit using the model. It shows the actual digit and predicted digit side by side.

Multiple Random Prediction from Training Dataset:

```
fig, axes = plt.subplots(8,8, figsize=(12,12))
#fig.tight_layout(pad=0.13,rect=[0, 0.03, 1, 0.91]) #[left, bottom, right, top]

for i, ax in enumerate(axes.flat):
  index = np.random.randint(m)
  X_reshaped_random = X[index].reshape((20,20)).T

  # Display the image
  ax.imshow(X_reshaped_random, cmap='gray')

  prediction = model.predict(X[index].reshape(1,400))  # prediction
  #print(f" predicting a Two: \n{prediction}")
  #print(f" Largest Prediction index: {np.argmax(prediction)}")

  prediction_p = tf.nn.softmax(prediction)
  # print(f" predicting a Two. Probability vector: \n{prediction_p}")
  # print(f"Total of predictions: {np.sum(prediction_p):0.3f}")

  yhat = np.argmax(prediction_p)
  # print(f"np.argmax(prediction_p): {yhat}")

  ax.set_title(f"Actual: {y[index]}, Prediction: {yhat}", fontsize=8)
  ax.set_axis_off()

fig.subplots_adjust(wspace=0.4, hspace=0.2)
plt.show()
```

This code pick multiple random image from the dataset and predict the digit using the model. It also shows the actual digit and predicted digit side by side.

Prediction Accuracy for Each Class:

```
# finding training dataset prediction accuracy for each class
predictions = model.predict(X)
predicted_labels = np.argmax(predictions, axis=1)

per_class_accuracy_list = []

for i in range(10):
    class_indices = np.where(y == i)[0]
    class_accuracy = accuracy_score(y[class_indices], predicted_labels[class_indices])
    per_class_accuracy_list.append([i, class_accuracy])

for class_accuracy in per_class_accuracy_list:
    print(f"Class {class_accuracy[0]}: {class_accuracy[1] * 100:.2f}%")
```

This part of the code find the accuracy for each class 0 to 9 on the training dataset. The accuracy are store in list of list format. Then it shows the per-class accuracy in percentage format. Overall, the model achieves good accuracy on the training dataset.

Accuracy and MSE of the Model:

```
# Calculate Accuracy and MSE of the model
print(f'Accuracy: {accuracy_score(y, predicted_labels)}')
print(f'MSE: {mean_squared_error(y, predicted_labels)}')
```

This code shows the accuracy and MSE on the training dataset. The model receive Accuracy: 0.9996 and MSE: 0.0234.

Loading Test Dataset and Normalizing:

```
test_data = np.load('Test_X.npy')

scalar = StandardScaler()
scalar.fit(test_data)
test_data = scalar.transform(test_data)

predictions = model.predict(test_data)
predicted_labels = np.argmax(predictions, axis=1)
```

This part of the code load the dataset to Test the model and normalizes the dataset using StandardScalar() function to achieve a standardized data format across your entire system.

Prediction Inference for single instance on Test Dataset:

```
X = np.load('Test_X.npy')
y = np.load('Test_y.npy')

# Check the shape of X
print(f"Shape of X: {X.shape}")

# Assuming X has shape (num_samples, height, width), unpack only the first dimension
m = X.shape[0]  # Number of samples

# Select a random sample
random = np.random.randint(m)
image = X[random]
pre = X[random]  # Assuming you meant X instead of test_data here

# Plot the image
fig, ax = plt.subplots(1, 1, figsize=(0.5, 0.5))
X_reshaped = image.reshape((20, 20)).T  # Adjust reshaping based on your image size

# Display the image
ax.imshow(X_reshaped, cmap='gray')

# Make a prediction (assuming your model expects flattened input)
prediction = model.predict(pre.reshape(1, -1))  # Flatten to 1D if needed for prediction

# Convert to probabilities using softmax
prediction_p = tf.nn.softmax(prediction)

# Get the predicted label
yhat = np.argmax(prediction_p)

# Display the actual and predicted labels
ax.set_title(f"Actual: {y[random]}, Prediction: {yhat}")
plt.show()
```

We pick a random image from the test dataset and predict the digit using the model. It shows the actual digit and predicted digit side by side.

Multiple Random Prediction from Test Dataset::

```
fig, axes = plt.subplots(8,8, figsize=(14,14))
#fig.tight_layout(pad=0.13,rect=[0, 0.03, 1, 0.91]) #[left, bottom, right, top]

```

```
for i, ax in enumerate(axes.flat):
  index = np.random.randint(m)
  X_reshaped_random = X[index].reshape((20,20)).T

  # Display the image
  ax.imshow(X_reshaped_random, cmap='gray')

  prediction = model.predict(X[index].reshape(1,400))  # prediction
  #print(f" predicting a Two: \n{prediction}")
  #print(f" Largest Prediction index: {np.argmax(prediction)}")

  prediction_p = tf.nn.softmax(prediction)
  # print(f" predicting a Two. Probability vector: \n{prediction_p}")
  # print(f"Total of predictions: {np.sum(prediction_p):0.3f}")

  yhat = np.argmax(prediction_p)
  # print(f"np.argmax(prediction_p): {yhat}")

  ax.set_title(f"Actual: {y[index]}, Prediction: {yhat}", fontsize=8)
  ax.set_axis_off()

fig.subplots_adjust(wspace=0.4, hspace=0.2)
plt.show()
```

Here, we pick multiple random image from the test dataset and predict the digit using the model. It also shows the actual digit and predicted digit side by side.

Prediction Accuracy for Each Class on Test Dataset:

```
X_reshaped_random = np.load("Test_X.npy")
y = np.load("Test_y.npy")

scaler = StandardScaler()
scaler.fit(X)

X = scaler.transform(X)

predictions = model.predict(X)
predicted_labels = np.argmax(predictions, axis=1)

per_class_accuracy_list = []

for i in range(10):
    class_indices = np.where(y == i)[0]
    class_accuracy = accuracy_score(y[class_indices], predicted_labels[class_indices])
    per_class_accuracy_list.append([i, class_accuracy])
print("Accuracy")
for class_name, accuracy in per_class_accuracy_list:
    print(f"Class {class_name}: {accuracy * 100:.2f}%")
```

This part of the code find the accuracy for each class 0 to 9 on the Test dataset. The accuracy are store in list of list format. Then it shows the per-class accuracy in percentage format. But the model didn't achieve good accuracy on the training dataset.

Accuracy and MSE of the Model on Test Dataset:

```
# Calculate Accuracy and MSE of the model
print(f'Accuracy: {accuracy_score(y, predicted_labels)}')
print(f'MSE: {mean_squared_error(y, predicted_labels)}')
```

The model receive Accuracy: 0.26 and MSE: 10.09.

B. Test Dataset Creation and Processing:

Necessary Imports:

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.patches as patches
from PIL import Image, ImageEnhance, ImageOps
%matplotlib inline
```

Imported all the necessary libraries and modules that will used to create the Test Dataset.

Load image:

```
# load image
image = Image.open('/content/digits5.png')

plt.figure(figsize=(8,4))
plt.imshow(image)
plt.show()
```

Load the image to create the test dataset

Change Color Mode:

```
image = image.convert(mode="L")  # L is 8-bit black-and-white image mode
plt.figure(figsize=(12, 12))
plt.imshow(image, cmap='gray')
image = ImageEnhance.Contrast(image).enhance(1.5)
plt.figure(figsize=(12, 12))
plt.imshow(image, cmap='gray')
plt.show()
```

We change the color mode from RGBA to Black and White.

Cut the Images:

```
images = []
size = 30
for digit, y in enumerate(range(0, image.width, size)):
  cuts=[]
  for x in range(0, image.width, size):
    cut = image.crop(box=(x,y, x+size, y+size))
    cuts.append(cut)
  images.append(cuts)
print(f'Cut {len(images)*len(images[0])}')
```

Cut the image into multiple pieces.

Ploting Test Dataset Image:

```
fig = plt.figure(figsize=(18,2))
ax=fig.subplots(1,6)

for i, n in enumerate(np.random.randint(0, len(images), size=6)):
  m = (np.random.randint(0, len(images[n])))
  ax[i].imshow(images[n][m])
  ax[i].set_title(f'Digit: [{n}]')
plt.show()
```

Plot the cutting image from the Test Data.

Center Image:

```
sample = images[np.random.randint(7)][np.random.randint(7)]

# Inver sample, get bbox and display all that stuff.
inv_sample = ImageOps.invert(sample)
bbox = inv_sample.getbbox()

fig = plt.figure(figsize=(2, 2))
ax = fig.add_axes([0,0,1,1])
ax.imshow(inv_sample)
rect = patches.Rectangle((bbox[0], bbox[3]), bbox[2]-bbox[0], -bbox[3]+bbox[1]-1,fill=False, a
ax.add_patch(rect)
plt.show()
```

Center the cutting image to create the test dataset.

Resize the Images:

```
#resize image
crop = inv_sample.crop(bbox)

#resize back
new_size = 20
delta_w = new_size - crop.size[0]
delta_h = new_size - crop.size[1]

padding = (delta_w//2, delta_h//2, delta_w-(delta_w//2), delta_h-(delta_h//2))
new_im = ImageOps.expand(crop, padding)

def resize_and_center(sample, new_size=20):
    inv_sample = ImageOps.invert(sample)
    bbox = inv_sample.getbbox()
    crop = inv_sample.crop(bbox)

    delta_w = new_size - crop.size[0]
    delta_h = new_size - crop.size[1]

    padding = (delta_w//2, delta_h//2, delta_w-(delta_w//2), delta_h-(delta_h//2))
    return ImageOps.expand(crop, padding)

resized_images = []
for row in images:
    resized_images.append([resize_and_center(sample) for sample in row])
```

We resize and center the cutting image to properly formatting the test dataset.

Ploating the Processed Dataset:

```
preview = Image.new('L', (len(images[0]) * 30, len(images) * 30))
x = 0
y = 0
for row in images:
    for sample in row:
        preview.paste(sample, (x, y))
        x += 30
    y += 30
    x = 0

plt.imshow(preview, cmap='gray')
plt.axis('off')
```

```
plt.figure(figsize=(18, 18))
plt.show()
```

We plot the processed new test dataset.

Save the Result in NumPy Binary Format:

```
binary_samples = np.array([[sample.getdata() for sample in row] for row in resized_images])
binary_samples = binary_samples.reshape(len(resized_images)*len(resized_images[0]), 20, 20)

random_index = np.random.randint(77)
sample_image = binary_samples[random_index]

plt.figure(figsize=(1, 1))
plt.imshow(sample_image, cmap='gray')
plt.show()

classes = np.array([[i]*10 for i in range(10)]).reshape(100,1)

print(f'X shape: {binary_samples.shape}')
print(f'y shape: {classes.shape}')
```

Then we processes a set of resized images by extracting pixel data, reshaping it into 20x20 arrays, and storing the binary pixel values in binary samples. It then randomly selects and displays one image from the dataset, and prints the shapes of the input data (X) and labels (y).

Plotting the Binary Formatted Dataset:

```
for i in np.random.randint(x_test.shape[0], size=5):
    plt.figure(figsize=(1, 1))
    plt.imshow(x_test[i], cmap='gray')
    plt.title(f'Digit {y_test[i]}')
    plt.axis('off')
    plt.show()
```

Plot some image with label from the newly created test dataset.

Flatten the Array:

```
arr = np.load("digits_x_test.npy")
arr_transposed = np.transpose(arr, axes=(0, 2, 1))
arr_flattened = arr_transposed.reshape((100, 400))
np.save('digits_x_test2.npy', arr_flattened)
```

Flatten the Array from dimensions (100,20,20) to (100,400).

Visualizing Test Dataset:

```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)
# You do not need to modify anything in this cell

m, n = X.shape

fig, axes = plt.subplots(8,8, figsize=(5,5))
fig.tight_layout(pad=0.13,rect=[0, 0.03, 1, 0.91]) #[left, bottom, right, top]

#fig.tight_layout(pad=0.5)

for i,ax in enumerate(axes.flat):
    # Select random indices
```

```
    random_index = np.random.randint(m)

    # Select rows corresponding to the random indices and
    # reshape the image
    X_random_reshaped = X[random_index].reshape((20,20)).T

    # Display the image
    ax.imshow(X_random_reshaped, cmap='gray')

    # Display the label above the image
    ax.set_title(y[random_index,0])
    ax.set_axis_off()
    fig.suptitle("Label, image", fontsize=14)
```

This code randomly selects some images from the Test dataset, then reshaping the image into (500x20x20) pixels and displaying it in grayscale. Each image is displayed alongside its corresponding label from y, with the title "Label, image" shown at the top of the grid/image.

Concatenate Test Dataset:

```
data1 = np.load('digits_x_test2.npy')
data2 = np.load('digits_x_test2.npy')

concatenated_X_data = np.concatenate((data1, data2), axis=0)
np.save('Test_X.npy', concatenated_X_data)

data3 = np.load('Test_X.npy')
print('Shape of Test_X:',data3.shape)


data4 = np.load('digits_y_test.npy')
data5 = np.load('digits_y_test.npy')

concatenated_y_data = np.concatenate((data4, data5), axis=0)
np.save('Test_y.npy', concatenated_y_data)

data6 = np.load('Test_y.npy')
print('Shape of Test_y:',data6.shape)
```

Then we loads two datasets (`digits_x_test2.npy` for input data and `digits_y_test.npy` for labels), concatenates them along the first axis to combine the test data and labels, and saves the resulting arrays as `Test_X.npy` and `Test_y.npy`. It then reloads the concatenated data and prints the shapes of both the combined input data (`Test_X`) and label data (`Test_y`) for verification.

III. Experiment results

A. Constant Hyper-Parameters vs Decreasing Hyper-Parameters:

	Constant Hyper-Parameter	Decreasing Hyper-Parameter
Hidden layer 1	0.001	0.01
Hidden layer 2	0.001	0.001
Hidden layer 3	0.001	0.0001
MSE	12.82	10.09
Accuracy	0.30	0.26

From the Table We can say that, The Hyper-parameter plays a significant role one the model accuracy and MSE. Using decreasing values for the regularization hyper-parameters for each layer minimize the MSE on Test dataset and also minimize the accuracy for the model.

B. 2 vs 3 Hidden Layers:

No of Hidden Layers	MSE	Accuracy
2 Hidden Layers	11.39	0.35
3 Hidden Layers	10.09	0.26

From the Table We can say that, number of layers also plays a significant role one the model accuracy and MSE. Here 2 hidden layers models achieved 35% accuracy but 3 hidden layers model achieved 26% accuracy. But 3 layers model achieved lower MSE which is 10.09.

C. Final Model Architecture:

Our Final Model Architecture contain total 3 hidden layer and one output layer and one input layer. The model contain 400 units input layer, because the dataset contain 400 (20x20) features. Then the first Hidden layer contain 25 units, second hidden layer contain 15 units, and the third hidden layer contain 10 units, and each hidden layer used ReLU as activation function and L2 regularization. The last layer contain 10 units and Linear activation function which is the output layer. In order to address over-fitting issue we regularized the 3 hidden layer with decreasing regularizing hyper parameter 0.01 to 0.0001.

Accuracy for Each Classes:

Per Class Accuracy	
Train Dataset	**Test Dataset**
Class 0: 100.00%	Class 0: 60.00%
Class 1: 100.00%	Class 1: 30.00%
Class 2: 100.00%	Class 2: 30.00%
Class 3: 99.80%	Class 3: 0.00%
Class 4: 100.00%	Class 4: 40.00%
Class 5: 100.00%	Class 5: 0.00%
Class 6: 100.00%	Class 6: 40.00%
Class 7: 100.00%	Class 7: 60.00%
Class 8: 100.00%	Class 8: 0.00%
Class 9: 99.80%	Class 9: 0.00%

MSE		**Accuracy**	
Train Dataset	**Test Dataset**	**Train Dataset**	**Test Dataset**
0.0234	10.09	99.96%	26.00%

The model that we build is performing good on the training dataset, but it is performing very bad on the test dataset, that we've created.

Prediction on Train Dataset vs Test Dataset:

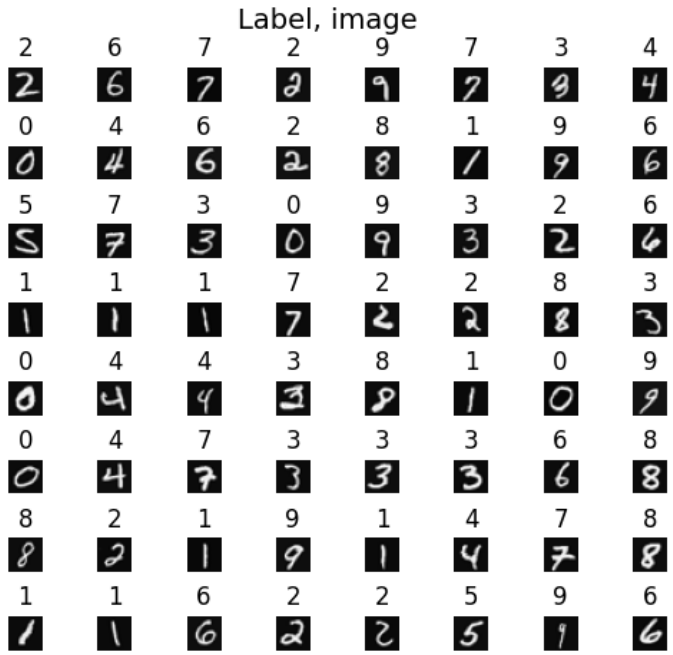

Fig. 1. Prediction on Train Dataset

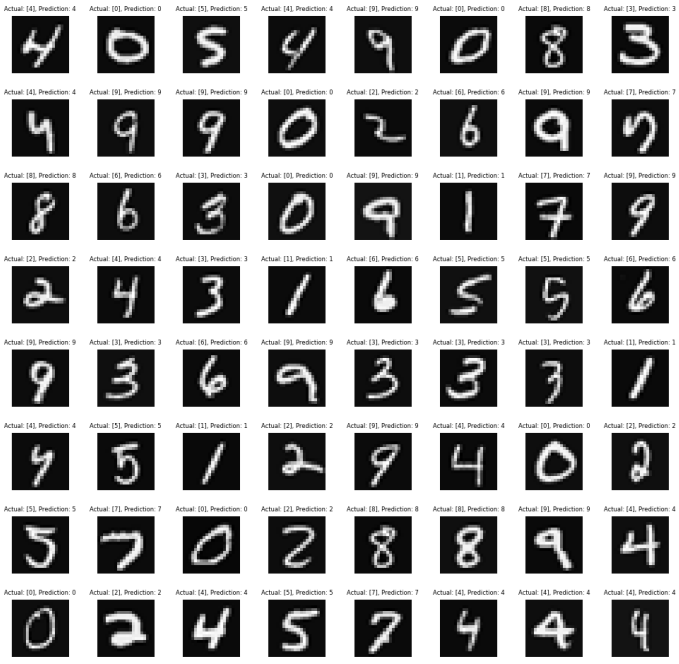

Fig. 2. Prediction on Train Dataset

IV. Discussion

During this assignment, we learned about Neural Network, How can we create a Neural Network model and train a Neural Network Model to recognize hand written digits. We train the model on a build in dataset and we test the model on a new dataset that we've created. We build 3 different model with different architecture some contain different number of hidden layer and some contain different Hyper Parameter. Out final model still didn't achieved good accuracy on the test dataset. To build the dataset we follow a different resource the link is given below on the References sections.

References

[1] Mikalaichaly. Diy mnist dataset. https://www.kaggle.com/code/mikalaichaly/diy-mnist-dataset, 2023.