# Digit_recognization

June 29, 2023

# 1 Problem Statement: Digit Recognization

## 1.1 Description:

- **The MNIST dataset is a dataset of handwritten digits that is commonly used for training and testing machine learning models for digit recognition. The dataset contains 60,000 training images and 10,000 testing images. Each image is a 28x28 pixel grayscale image of a handwritten digit.**
- **To recognize digits using the MNIST dataset, we can use a variety of machine learning models. One common approach is to use a convolutional neural network (CNN). A CNN is a type of neural network that is particularly well-suited for image classification tasks. CNNs work by learning to extract features from images. These features can then be used to classify the images into different categories.**

Here are some of the advantages of using the MNIST dataset for digit recognition:

- **The dataset is large and well-balanced:** This means that the dataset contains a large number of images of each digit, and the distribution of images is evenly spread across the different digits.
- **The dataset is clean and well-curated:** This means that the images in the dataset are of high quality and they have been carefully selected to be representative of the different digits.
- **The dataset is open source:** This means that it is freely available to anyone who wants to use it.

# 2 Importing Libraries

[ ]:
```
# Importing Libraries
import tensorflow as tf
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os
import warnings
warnings.filterwarnings('ignore')
```

- Checking version

## 3 Setting Working Directory

[ ]:
```
print(f"Tensorflow Version{tf.__version__}")
print(f"Keras Version{tf.keras.__version__}")
```

```
Tensorflow Version2.12.0
Keras Version2.12.0
```

- GPU/TPU/CPU Check

[ ]:
```
tf.config.list_physical_devices("GPU")
```

[ ]:
```
[PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
```

[ ]:
```
tf.config.list_physical_devices("CPU")
```

[ ]:
```
[PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

## 4 Creating a simple classifier Using Keras

[ ]:
```
# loading datasets
mnist=tf.keras.datasets.mnist
(X_train_full,y_train_full),(X_test,y_test)=mnist.load_data()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-
datasets/mnist.npz
11490434/11490434 [==============================] - 1s 0us/step
```

[ ]:
```
print(f"data type of X_train_full: {X_train_full.dtype},\n shape of␣
↪X_train_full: {X_train_full.shape}")
```

```
data type of X_train_full: uint8,
 shape of X_train_full: (60000, 28, 28)
```

[ ]:
```
X_test.shape
```

[ ]:
```
(10000, 28, 28)
```

[ ]:
```
len(X_test[1][0])
```

[ ]:
```
28
```

[ ]:
```
len(X_test[1][1])
```

[ ]:
```
28
```

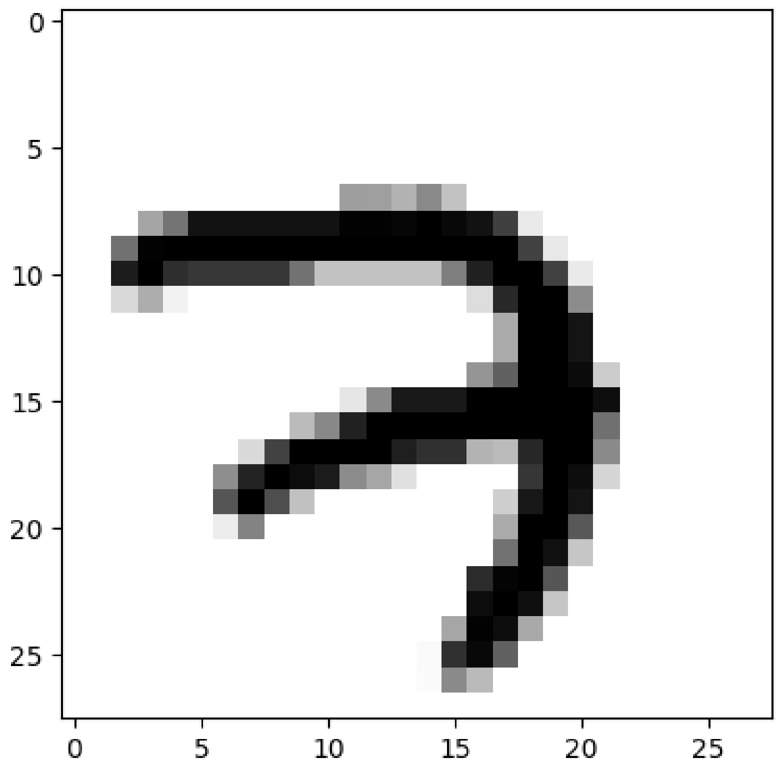
## 5 Training Data Preparation

[ ]:
```python
# create a validation data set from the full training data
# Scale the data between 0 to 1 by dividing it by 255. as its an unsigned data␣
↪between 0-255 range
X_valid, X_train = X_train_full[:5000] / 255., X_train_full[5000:] / 255.
y_valid, y_train = y_train_full[:5000], y_train_full[5000:]

# scale the test set as well
X_test = X_test / 255.
```

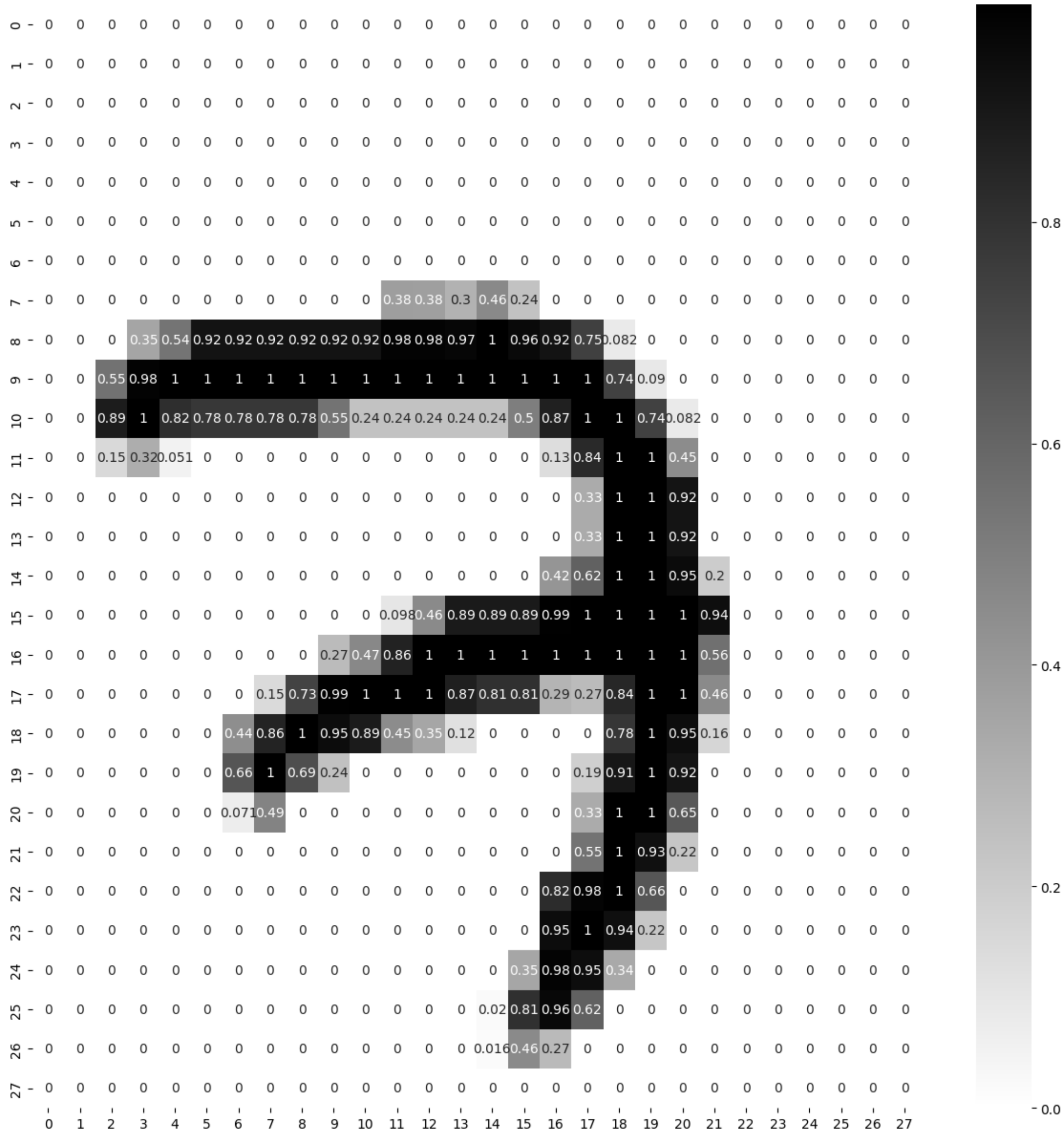
- **Let's View some data**

[ ]:
```python
plt.imshow(X_train[0],cmap='binary')
plt.show()
```




[ ]:
```python
plt.figure(figsize=(15,15))
sns.heatmap(X_train[0],annot=True,cmap='binary')
```

[ ]: <Axes: >



# 6 Architecture Used:

# 7 Creating Layers Of ANN

[ ]:
```
LAYERS=[tf.keras.layers.Flatten(input_shape=[28,28],name="inputLayer"),
        tf.keras.layers.Dense(300,activation='relu',name='hiddenLayer1'),
        tf.keras.layers.Dense(100,activation='relu',name='hiddenLayer2'),
        tf.keras.layers.Dense(10,activation='softmax',name='outputLayer')
        ]
```

```
model_clf=tf.keras.models.Sequential(LAYERS)
```

[ ]:
```
model_clf.layers
```

[ ]:
```
[<keras.layers.reshaping.flatten.Flatten at 0x7f8da01cefb0>,
 <keras.layers.core.dense.Dense at 0x7f8da0c8a1d0>,
 <keras.layers.core.dense.Dense at 0x7f8da01ce7a0>,
 <keras.layers.core.dense.Dense at 0x7f8da01ce380>]
```

- Let's check summary of model

[ ]:
```
model_clf.summary()
```

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

 hiddenLayer1 (Dense)        (None, 300)               235500

 hiddenLayer2 (Dense)        (None, 100)               30100

 outputLayer (Dense)         (None, 10)                1010

=================================================================
Total params: 266,610
Trainable params: 266,610
Non-trainable params: 0
_________________________________________________________________
```

- Total paramenter in ecah layer follows

[ ]:
```
# first Layer * second Layer + bias
784*300 + 300, 300*100+100, 100*10+10
```

[ ]:
```
(235500, 30100, 1010)
```

[ ]:
```
# Total parameters to be trained
sum((235500, 30100, 1010))
```

[ ]:
```
266610
```

[ ]:
```
hidden1 = model_clf.layers[1]
hidden1.name
```

[ ]:
```
'hiddenLayer1'
```

[ ]:
```
len(hidden1.get_weights()[1])
```

[ ]: 300

[ ]:
```
hidden1.get_weights()
```

[ ]:
```
[array([[-0.05778085, -0.01985833, -0.06345551, …, -0.06591441,
         -0.02038335,  0.02102906],
        [ 0.06355079,  0.06457365,  0.04312076, …,  0.05226387,
          0.00453257, -0.0458843 ],
        [-0.01369086,  0.02819347, -0.06354216, …,  0.02187817,
          0.01871001,  0.0307809 ],
        …,
        [-0.06798964, -0.07203194,  0.07317001, …, -0.00944325,
         -0.06340548, -0.00431037],
        [ 0.02980805, -0.01724534, -0.03318619, …,  0.07307966,
         -0.01279022, -0.06807948],
        [ 0.03112555,  0.04765788,  0.01050752, …, -0.06275913,
          0.06898342, -0.06909705]], dtype=float32),
 array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
        0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)]
```

[ ]:
```
weights, biases = hidden1.get_weights()
```

[ ]:
```
print("shape\n",weights.shape, "\n")
weights
```

```
shape
 (784, 300)
```

[ ]:
```
array([[-0.05778085, -0.01985833, -0.06345551, …, -0.06591441,
        -0.02038335,  0.02102906],
       [ 0.06355079,  0.06457365,  0.04312076, …,  0.05226387,
```

```
           0.00453257, -0.0458843 ],
         [-0.01369086,  0.02819347, -0.06354216, …,  0.02187817,
           0.01871001,  0.0307809 ],
         …,
         [-0.06798964, -0.07203194,  0.07317001, …, -0.00944325,
          -0.06340548, -0.00431037],
         [ 0.02980805, -0.01724534, -0.03318619, …,  0.07307966,
          -0.01279022, -0.06807948],
         [ 0.03112555,  0.04765788,  0.01050752, …, -0.06275913,
           0.06898342, -0.06909705]], dtype=float32)
```

[ ]:
```
print("shape\n", biases.shape)
```

```
shape
 (300,)
```

## 8 Model Training

[ ]:
```
LOSS_FUNCTION="sparse_categorical_crossentropy"
OPTIMIZER="SGD"
METRICS=['accuracy']

model_clf.compile(loss=LOSS_FUNCTION,
                  optimizer=OPTIMIZER,
                  metrics=METRICS)
```

## 9 Tensorboard callback function

[ ]:
```
# Logging

import time

def get_log_path(log_dir="logs/fit"):
  fileName = time.strftime("log_%Y_%m_%d_%H_%M_%S")
  logs_path = os.path.join(log_dir, fileName)
  print(f"Saving logs at {logs_path}")
  return logs_path

log_dir = get_log_path()
tb_cb = tf.keras.callbacks.TensorBoard(log_dir=log_dir)
```

```
Saving logs at logs/fit/log_2023_06_12_14_03_57
```

- Early Stopping callback

[ ]:
```
early_stopping_cb = tf.keras.callbacks.EarlyStopping(patience=5,␣
 ↪restore_best_weights=True)
```

- Model checkpoint callback

```
[ ]: CKPT_path = "Model_ckpt.h5"
     checkpointing_cb = tf.keras.callbacks.ModelCheckpoint(CKPT_path,␣
      ↪save_best_only=True)
```

```
[ ]: # Orginal train

     EPOCHS = 30
     VALIDATION_SET = (X_valid, y_valid)

     history = model_clf.fit(X_train, y_train, epochs=EPOCHS,
                         validation_data=VALIDATION_SET, batch_size=32,␣
      ↪callbacks=[tb_cb, early_stopping_cb,checkpointing_cb] )
```

```
Epoch 1/30
1719/1719 [==============================] - 15s 4ms/step - loss: 0.6119 -
accuracy: 0.8424 - val_loss: 0.3024 - val_accuracy: 0.9138
Epoch 2/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2914 -
accuracy: 0.9172 - val_loss: 0.2414 - val_accuracy: 0.9286
Epoch 3/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.2417 -
accuracy: 0.9301 - val_loss: 0.2090 - val_accuracy: 0.9430
Epoch 4/30
1719/1719 [==============================] - 11s 6ms/step - loss: 0.2075 -
accuracy: 0.9405 - val_loss: 0.1861 - val_accuracy: 0.9478
Epoch 5/30
1719/1719 [==============================] - 9s 5ms/step - loss: 0.1815 -
accuracy: 0.9478 - val_loss: 0.1638 - val_accuracy: 0.9544
Epoch 6/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.1602 -
accuracy: 0.9544 - val_loss: 0.1460 - val_accuracy: 0.9592
Epoch 7/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1433 -
accuracy: 0.9586 - val_loss: 0.1346 - val_accuracy: 0.9618
Epoch 8/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.1286 -
accuracy: 0.9635 - val_loss: 0.1251 - val_accuracy: 0.9654
Epoch 9/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1164 -
accuracy: 0.9670 - val_loss: 0.1191 - val_accuracy: 0.9678
Epoch 10/30
1719/1719 [==============================] - 8s 5ms/step - loss: 0.1060 -
accuracy: 0.9697 - val_loss: 0.1098 - val_accuracy: 0.9690
Epoch 11/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0977 -
accuracy: 0.9724 - val_loss: 0.1056 - val_accuracy: 0.9710
```

```
Epoch 12/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0900 -
accuracy: 0.9750 - val_loss: 0.1034 - val_accuracy: 0.9700
Epoch 13/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0826 -
accuracy: 0.9769 - val_loss: 0.0967 - val_accuracy: 0.9722
Epoch 14/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0767 -
accuracy: 0.9787 - val_loss: 0.0911 - val_accuracy: 0.9740
Epoch 15/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0716 -
accuracy: 0.9801 - val_loss: 0.0898 - val_accuracy: 0.9736
Epoch 16/30
1719/1719 [==============================] - 8s 5ms/step - loss: 0.0666 -
accuracy: 0.9816 - val_loss: 0.0857 - val_accuracy: 0.9754
Epoch 17/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0621 -
accuracy: 0.9834 - val_loss: 0.0880 - val_accuracy: 0.9740
Epoch 18/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0580 -
accuracy: 0.9844 - val_loss: 0.0821 - val_accuracy: 0.9762
Epoch 19/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0542 -
accuracy: 0.9854 - val_loss: 0.0793 - val_accuracy: 0.9762
Epoch 20/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0511 -
accuracy: 0.9867 - val_loss: 0.0788 - val_accuracy: 0.9758
Epoch 21/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0480 -
accuracy: 0.9873 - val_loss: 0.0752 - val_accuracy: 0.9764
Epoch 22/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0451 -
accuracy: 0.9883 - val_loss: 0.0756 - val_accuracy: 0.9766
Epoch 23/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0422 -
accuracy: 0.9894 - val_loss: 0.0771 - val_accuracy: 0.9770
Epoch 24/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0398 -
accuracy: 0.9900 - val_loss: 0.0735 - val_accuracy: 0.9772
Epoch 25/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0374 -
accuracy: 0.9907 - val_loss: 0.0720 - val_accuracy: 0.9778
Epoch 26/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0351 -
accuracy: 0.9912 - val_loss: 0.0709 - val_accuracy: 0.9776
Epoch 27/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0331 -
accuracy: 0.9920 - val_loss: 0.0700 - val_accuracy: 0.9784
```

```
Epoch 28/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0313 -
accuracy: 0.9926 - val_loss: 0.0703 - val_accuracy: 0.9780
Epoch 29/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0297 -
accuracy: 0.9929 - val_loss: 0.0729 - val_accuracy: 0.9784
Epoch 30/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0278 -
accuracy: 0.9939 - val_loss: 0.0707 - val_accuracy: 0.9790
```

[ ]:
```
# Checkpoint training

#loading Checkpoint model
ckpt_model = tf.keras.models.load_model(CKPT_path)

history = ckpt_model.fit(X_train, y_train, epochs=EPOCHS,
                    validation_data=VALIDATION_SET, batch_size=32,␣
 ↪callbacks=[tb_cb, early_stopping_cb,checkpointing_cb] )
```

```
Epoch 1/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0312 -
accuracy: 0.9924 - val_loss: 0.0715 - val_accuracy: 0.9778
Epoch 2/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0296 -
accuracy: 0.9928 - val_loss: 0.0707 - val_accuracy: 0.9786
Epoch 3/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0278 -
accuracy: 0.9939 - val_loss: 0.0751 - val_accuracy: 0.9770
Epoch 4/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0263 -
accuracy: 0.9941 - val_loss: 0.0702 - val_accuracy: 0.9792
Epoch 5/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0248 -
accuracy: 0.9948 - val_loss: 0.0704 - val_accuracy: 0.9774
Epoch 6/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0234 -
accuracy: 0.9950 - val_loss: 0.0692 - val_accuracy: 0.9802
Epoch 7/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0220 -
accuracy: 0.9955 - val_loss: 0.0688 - val_accuracy: 0.9790
Epoch 8/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0210 -
accuracy: 0.9957 - val_loss: 0.0713 - val_accuracy: 0.9782
Epoch 9/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0199 -
accuracy: 0.9963 - val_loss: 0.0693 - val_accuracy: 0.9798
Epoch 10/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0187 -
```

```
accuracy: 0.9967 - val_loss: 0.0697 - val_accuracy: 0.9804
Epoch 11/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0176 -
accuracy: 0.9969 - val_loss: 0.0710 - val_accuracy: 0.9796
Epoch 12/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0169 -
accuracy: 0.9972 - val_loss: 0.0673 - val_accuracy: 0.9802
Epoch 13/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0160 -
accuracy: 0.9977 - val_loss: 0.0696 - val_accuracy: 0.9796
Epoch 14/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0150 -
accuracy: 0.9978 - val_loss: 0.0674 - val_accuracy: 0.9804
Epoch 15/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0143 -
accuracy: 0.9981 - val_loss: 0.0682 - val_accuracy: 0.9806
Epoch 16/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0136 -
accuracy: 0.9982 - val_loss: 0.0734 - val_accuracy: 0.9792
Epoch 17/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0128 -
accuracy: 0.9984 - val_loss: 0.0688 - val_accuracy: 0.9790
```

## 10 Saving Model

[ ]:
```
import time
import os

def save_model_path(MODEL_dir = "TRAINED_MODEL"):
  os.makedirs(MODEL_dir, exist_ok= True)
  fileName = time.strftime("Model_%Y_%m_%d_%H_%M_%S_.h5")
  model_path = os.path.join(MODEL_dir, fileName)
  print(f"Model {fileName} will be saved at {model_path}")
  return model_path
```

[ ]:
```
UNIQUE_PATH = save_model_path()
UNIQUE_PATH
```

```
Model Model_2023_06_12_14_12_30_.h5 will be saved at
TRAINED_MODEL/Model_2023_06_12_14_12_30_.h5
```

[ ]: 'TRAINED_MODEL/Model_2023_06_12_14_12_30_.h5'

[ ]:
```
tf.keras.models.save_model(model_clf, UNIQUE_PATH)
```

[ ]:
```
history.params
```

[ ]: {'verbose': 1, 'epochs': 30, 'steps': 1719}

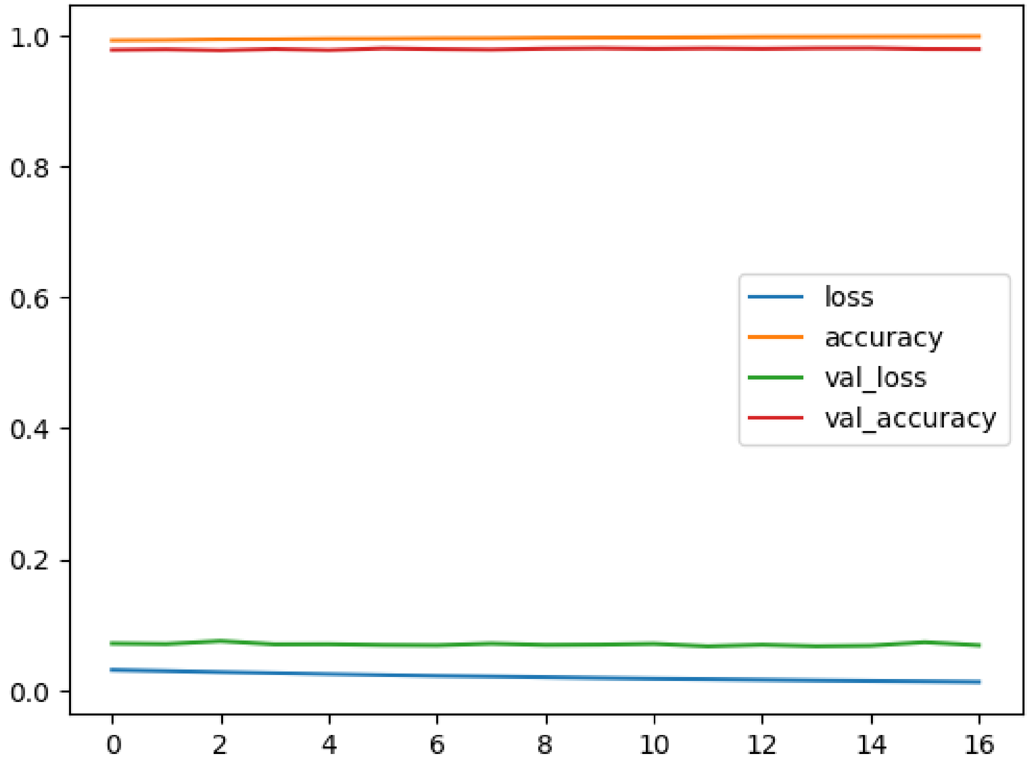
[ ]:
```
pd.DataFrame(history.history)
```

[ ]:

| | loss | accuracy | val_loss | val_accuracy |
|---|---|---|---|---|
| 0 | 0.031211 | 0.992364 | 0.071503 | 0.9778 |
| 1 | 0.029584 | 0.992818 | 0.070688 | 0.9786 |
| 2 | 0.027826 | 0.993873 | 0.075070 | 0.9770 |
| 3 | 0.026270 | 0.994055 | 0.070230 | 0.9792 |
| 4 | 0.024788 | 0.994782 | 0.070407 | 0.9774 |
| 5 | 0.023379 | 0.994964 | 0.069229 | 0.9802 |
| 6 | 0.022022 | 0.995491 | 0.068793 | 0.9790 |
| 7 | 0.021021 | 0.995673 | 0.071334 | 0.9782 |
| 8 | 0.019868 | 0.996345 | 0.069291 | 0.9798 |
| 9 | 0.018659 | 0.996655 | 0.069728 | 0.9804 |
| 10 | 0.017610 | 0.996855 | 0.071045 | 0.9796 |
| 11 | 0.016891 | 0.997164 | 0.067300 | 0.9802 |
| 12 | 0.015974 | 0.997673 | 0.069595 | 0.9796 |
| 13 | 0.015046 | 0.997836 | 0.067428 | 0.9804 |
| 14 | 0.014283 | 0.998055 | 0.068243 | 0.9806 |
| 15 | 0.013627 | 0.998218 | 0.073351 | 0.9792 |
| 16 | 0.012850 | 0.998436 | 0.068802 | 0.9790 |

[ ]:
```
pd.DataFrame(history.history).plot()
```

[ ]: <Axes: >




## 11 Testing Model

[ ]:
```
x_new = X_test[:3]
# x_new
```

[ ]:
```
actual = y_test[:3]
actual
```

[ ]:
```
array([7, 2, 1], dtype=uint8)
```

[ ]:
```
y_prob = model_clf.predict(x_new)
y_prob.round(3)
```

```
1/1 [==============================] - 0s 74ms/step
```

[ ]:
```
array([[0.   , 0.   , 0.001, 0.001, 0.   , 0.   , 0.   , 0.999, 0.   ,
        0.   ],
       [0.   , 0.   , 0.998, 0.001, 0.   , 0.   , 0.   , 0.   , 0.   ,
        0.   ],
       [0.   , 0.996, 0.   , 0.   , 0.   , 0.   , 0.   , 0.001, 0.001,
        0.   ]], dtype=float32)
```

[ ]:
```
y_prob
```

[ ]:
```
array([[3.8651456e-06, 3.6640265e-07, 5.0879118e-04, 6.0440681e-04,
        2.1154120e-08, 9.4563384e-06, 2.0078365e-10, 9.9877948e-01,
        1.9681072e-05, 7.3911047e-05],
       [2.4963595e-06, 6.7389403e-05, 9.9848819e-01, 1.4372601e-03,
        3.5959191e-12, 6.0718486e-07, 2.2031618e-06, 3.9537163e-12,
        1.8024812e-06, 2.2890223e-11],
       [1.2404228e-05, 9.9614775e-01, 2.5005249e-04, 3.7335056e-05,
        4.3321427e-04, 1.2218449e-04, 1.5830527e-04, 1.4332801e-03,
        1.3978776e-03, 7.5923363e-06]], dtype=float32)
```

[ ]:
```
y_pred = np.argmax(y_prob, axis = -1)
```

[ ]:
```
y_pred
```

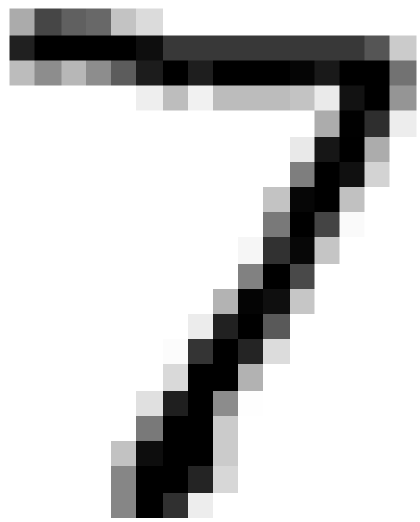
[ ]:
```
array([7, 2, 1])
```

[ ]:
```
actual
```

[ ]:
```
array([7, 2, 1], dtype=uint8)
```

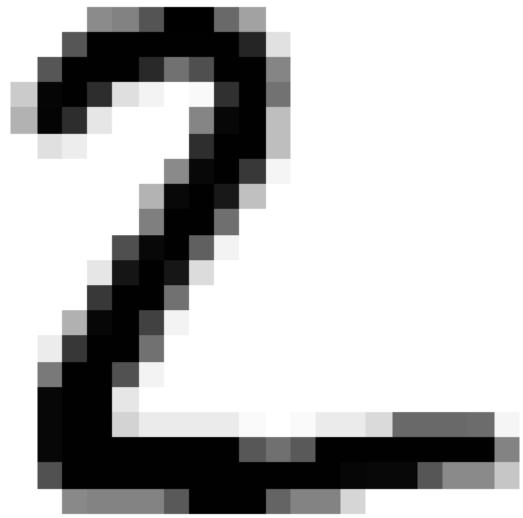
[ ]:
```
# plot
for data, pred, actual_data in zip(x_new, y_pred, actual):
  plt.imshow(data, cmap="binary")
  plt.title(f"Predicted {pred} and Actual {actual_data}")
  plt.axis("off")
  plt.show()
  print("######################")
```

Predicted 7 and Actual 7



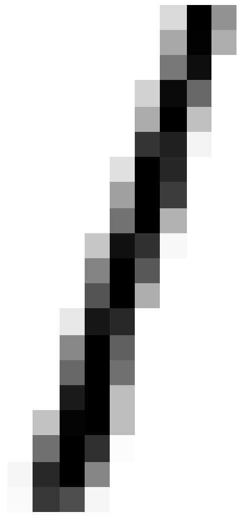
######################

Predicted 2 and Actual 2



#####################

Predicted 1 and Actual 1



######################

## 12 Thank You!