ann-using-keras

June 14, 2023

[ ]:
```python
import tensorflow as tf
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import os
```

[ ]:
```python
## cheking version of tensorflow ans keras

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

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

[ ]:
```python
## get current working directory

os.getcwd()
```

[ ]: '/content'

GPU / CPU Check

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

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

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

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

[ ]:
```python
check_list = ['GPU' , 'CPU']

for device in check_list:
  out = tf.config.list_physical_devices(device)
  if len(out) > 0:
    print(f"(device) is available ")
    print(f"Details >> {out}")
```

```
else:
  print(f"(device) isn't available ")
```

```
(device) is available
Details >> [PhysicalDevice(name='/physical_device:GPU:0', device_type='GPU')]
(device) is available
Details >> [PhysicalDevice(name='/physical_device:CPU:0', device_type='CPU')]
```

Creating a Simple classifier using keras on MNIST data

[ ]:
```
mnist = tf.keras.datasets.mnist
```

[ ]:
```
mnist
```

[ ]:
```
<module 'keras.api._v2.keras.datasets.mnist' from
'/usr/local/lib/python3.10/dist-
packages/keras/api/_v2/keras/datasets/mnist/__init__.py'>
```

[ ]:
```
(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
```

[ ]:
```
X_train_full.shape
```

[ ]:
```
(60000, 28, 28)
```

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

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

[ ]:
```
X_train_full[0]
```

[ ]:
```
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,   3,
  18,  18,  18, 126, 136, 175,  26, 166, 255, 247, 127,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,  30,  36,  94, 154, 170,
 253, 253, 253, 253, 253, 225, 172, 253, 242, 195,  64,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,  49, 238, 253, 253, 253, 253,
 253, 253, 253, 253, 251,  93,  82,  82,  56,  39,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,  18, 219, 253, 253, 253, 253,
 253, 198, 182, 247, 241,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,  80, 156, 107, 253, 253,
 205,  11,   0,  43, 154,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,  14,   1, 154, 253,
  90,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0, 139, 253,
 190,   2,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  11, 190,
 253,  70,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  35,
 241, 225, 160, 108,   1,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
  81, 240, 253, 253, 119,  25,   0,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  45, 186, 253, 253, 150,  27,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,  16,  93, 252, 253, 187,   0,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,   0,   0,   0, 249, 253, 249,  64,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
   0,  46, 130, 183, 253, 253, 207,   2,   0,   0,   0,   0,   0,
   0,   0],
 [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  39,
 148, 229, 253, 253, 253, 250, 182,   0,   0,   0,   0,   0,   0,
   0,   0],
```

```
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,  24, 114, 221,
        253, 253, 253, 253, 201,  78,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,  23,  66, 213, 253, 253,
        253, 253, 198,  81,   2,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,  18, 171, 219, 253, 253, 253, 253,
        195,  80,   9,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,  55, 172, 226, 253, 253, 253, 253, 244, 133,
         11,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0, 136, 253, 253, 253, 212, 135, 132,  16,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
          0,   0],
       [  0,   0,   0,   0,   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=uint8)
```

[ ]:
```
X_train_full[0].shape
```

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

[ ]:
```
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: (X_train_full.dtype),
 shape of X_train_full(X_train_full.shape)
```

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

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

[ ]:
```
## 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 unsighned␣
↪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
```

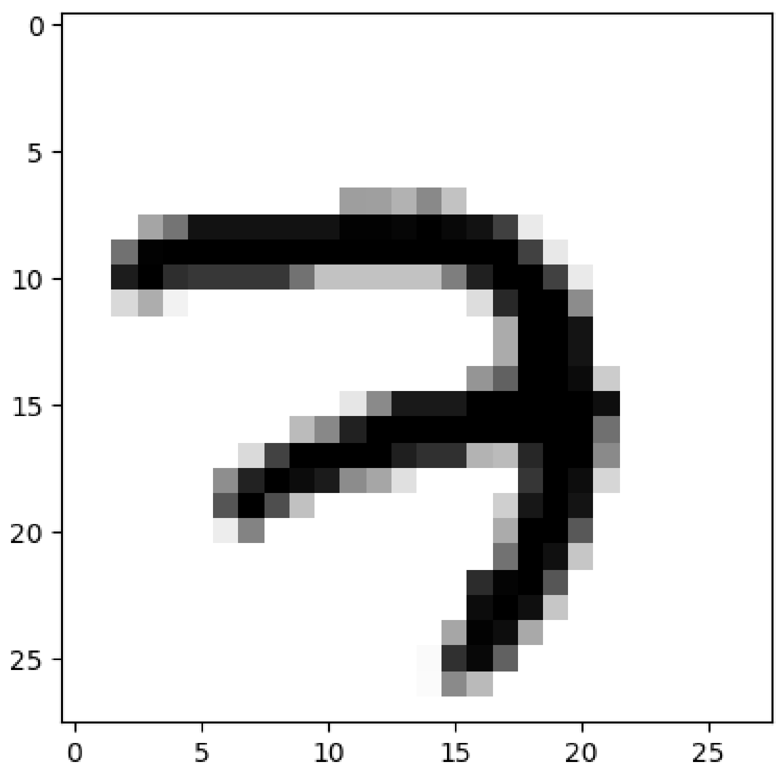
```
X_test = X_test / 255.
```

[ ]:
```
## TRAIN - 5000
## TEST - 10000
## VAL - 5000
```

[ ]:
```
len(X_train_full[5000:])
```

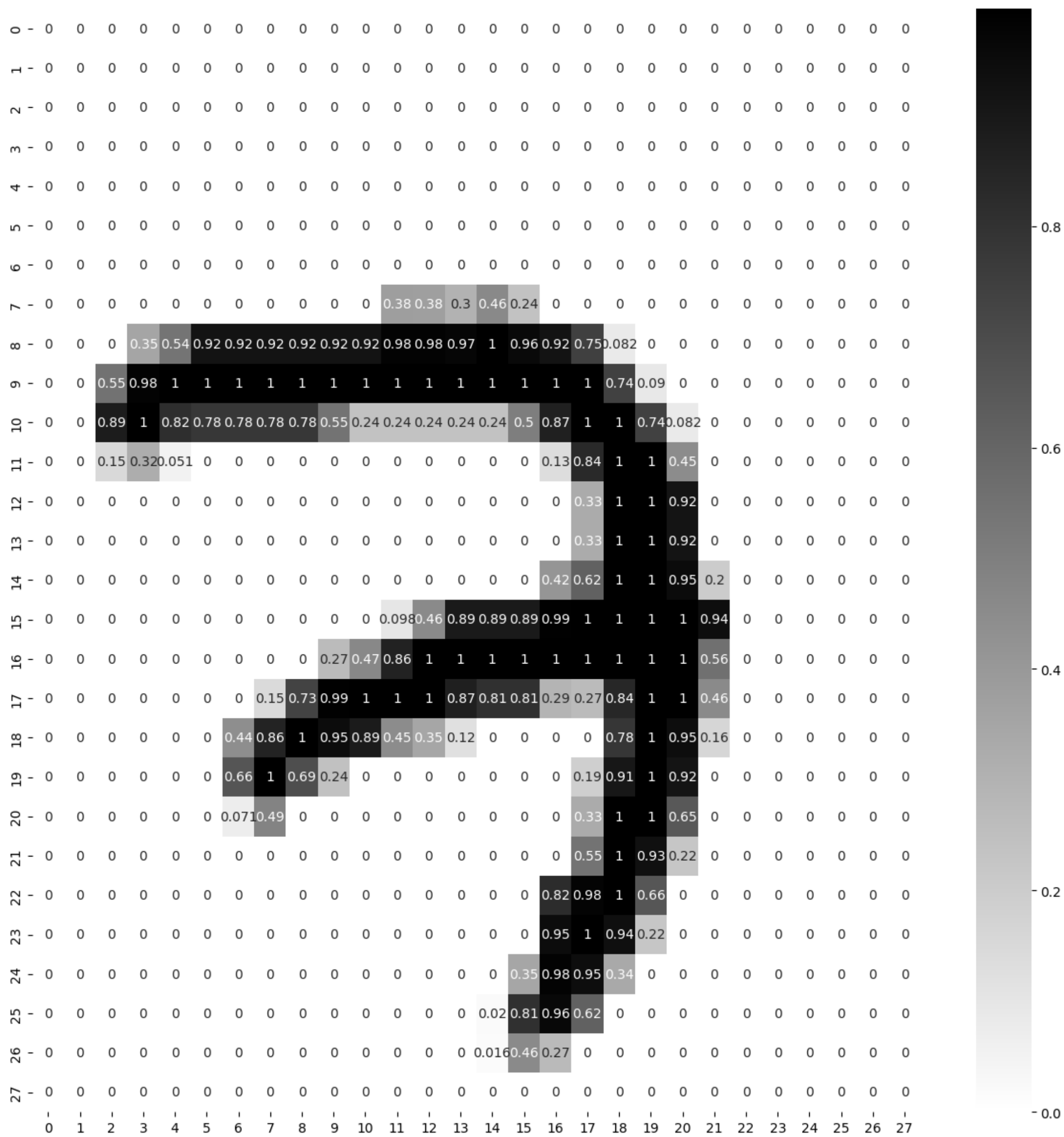
[ ]: 55000

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



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

[ ]: <Axes: >



[ ]:
```
28*28
```

[ ]: 784

[ ]:
```
## 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")]
```

[ ]:
```
LAYERS
```

[ ]:
```
[<keras.layers.reshaping.flatten.Flatten at 0x7fd420ef9ae0>,
 <keras.layers.core.dense.Dense at 0x7fd420ef8c70>,
 <keras.layers.core.dense.Dense at 0x7fd420ef9420>,
 <keras.layers.core.dense.Dense at 0x7fd420efa7d0>]
```

[ ]:
```
model_clf = tf.keras.models.Sequential(LAYERS)
```

[ ]:
```
model_clf
```

[ ]:
```
<keras.engine.sequential.Sequential at 0x7fd420efa620>
```

[ ]:
```
## METHOD 2
```

[ ]:
```
#from keras.models import Sequential
#from keras.layers import Dense , Flatten
# # Define the model
#model = Sequential()
#model.add(Dense(units=64,activation='relu',input_dim=100))
#model.add(Dense(units=10,activation='softmax'))
```

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

[ ]:
```
[<keras.layers.reshaping.flatten.Flatten at 0x7fd420ef9ae0>,
 <keras.layers.core.dense.Dense at 0x7fd420ef8c70>,
 <keras.layers.core.dense.Dense at 0x7fd420ef9420>,
 <keras.layers.core.dense.Dense at 0x7fd420efa7d0>]
```

[ ]:
```
300*100+100
```

[ ]:
```
30100
```

[ ]:
```
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
_________________________________________________________________
```

[ ]:
```python
# firstLayer * secondLayer + bias

784 * 300 + 300,300*100+100 , 100*10+10
```

[ ]: (235500, 30100, 1010)

[ ]:
```python
# Total parameters to be trained

sum((235500 , 30100 , 1010))
```

[ ]: 266610

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

[ ]: 'hiddenLayer1'

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

[ ]: 2

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

[ ]:
```
[array([[-0.07052253, -0.00980838, -0.00663119, …, -0.02820122,
          0.04396266,  0.07257029],
        [-0.04128718, -0.02377704, -0.04849973, …,  0.04107536,
         -0.06322491, -0.04616933],
        [-0.01920016, -0.00866343,  0.017957  , …,  0.06990054,
          0.0445122 ,  0.03071921],
        …,
        [ 0.05985168,  0.04764889, -0.05305887, …, -0.05786416,
         -0.04138853, -0.02648744],
        [-0.02726017,  0.05402628,  0.05745846, …,  0.07112448,
          0.01653311, -0.04043264],
        [-0.02783885, -0.05906404, -0.03861446, …,  0.05584833,
          0.02226647, -0.06523681]], 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()
```

[ ]:
```
weights.shape
```

[ ]:
```
(784, 300)
```

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

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

[ ]:
```
array([[-0.07052253, -0.00980838, -0.00663119, …, -0.02820122,
         0.04396266,  0.07257029],
       [-0.04128718, -0.02377704, -0.04849973, …,  0.04107536,
        -0.06322491, -0.04616933],
       [-0.01920016, -0.00866343,  0.017957  , …,  0.06990054,
         0.0445122 ,  0.03071921],
       …,
       [ 0.05985168,  0.04764889, -0.05305887, …, -0.05786416,
        -0.04138853, -0.02648744],
       [-0.02726017,  0.05402628,  0.05745846, …,  0.07112448,
         0.01653311, -0.04043264],
       [-0.02783885, -0.05906404, -0.03861446, …,  0.05584833,
         0.02226647, -0.06523681]], dtype=float32)
```

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

```
shape
 (300,)
```

[ ]:
```
LOSS_FUNCTION = "sparse_categorical_crossentropy" ## use => tf.losses.
↪sparde_categorical_crossentropy
OPTIMIZER = "SGD" ## or use with custom learning rata =>tf.keras.optimizers.SGD␣
↪(0.02)
METRICS =  ["accuracy"]

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

[ ]:
```
##    taining

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)
```

```
Epoch 1/30
1719/1719 [==============================] - 18s 6ms/step - loss: 0.6079 -
accuracy: 0.8442 - val_loss: 0.3077 - val_accuracy: 0.9160
Epoch 2/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.2912 -
accuracy: 0.9174 - val_loss: 0.2407 - val_accuracy: 0.9326
Epoch 3/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.2401 -
accuracy: 0.9324 - val_loss: 0.2065 - val_accuracy: 0.9442
Epoch 4/30
1719/1719 [==============================] - 8s 5ms/step - loss: 0.2057 -
accuracy: 0.9414 - val_loss: 0.1844 - val_accuracy: 0.9486
Epoch 5/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.1790 -
accuracy: 0.9487 - val_loss: 0.1636 - val_accuracy: 0.9546
Epoch 6/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.1586 -
accuracy: 0.9545 - val_loss: 0.1477 - val_accuracy: 0.9586
Epoch 7/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.1421 -
accuracy: 0.9594 - val_loss: 0.1345 - val_accuracy: 0.9626
Epoch 8/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.1277 -
accuracy: 0.9629 - val_loss: 0.1251 - val_accuracy: 0.9660
Epoch 9/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.1161 -
accuracy: 0.9671 - val_loss: 0.1168 - val_accuracy: 0.9662
Epoch 10/30
```

```
1719/1719 [==============================] - 6s 3ms/step - loss: 0.1062 -
accuracy: 0.9697 - val_loss: 0.1106 - val_accuracy: 0.9684
Epoch 11/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0976 -
accuracy: 0.9725 - val_loss: 0.1036 - val_accuracy: 0.9712
Epoch 12/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0900 -
accuracy: 0.9747 - val_loss: 0.1006 - val_accuracy: 0.9736
Epoch 13/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0834 -
accuracy: 0.9768 - val_loss: 0.0949 - val_accuracy: 0.9728
Epoch 14/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0773 -
accuracy: 0.9786 - val_loss: 0.0901 - val_accuracy: 0.9738
Epoch 15/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0719 -
accuracy: 0.9804 - val_loss: 0.0887 - val_accuracy: 0.9738
Epoch 16/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0672 -
accuracy: 0.9818 - val_loss: 0.0858 - val_accuracy: 0.9756
Epoch 17/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0629 -
accuracy: 0.9826 - val_loss: 0.0848 - val_accuracy: 0.9758
Epoch 18/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0584 -
accuracy: 0.9845 - val_loss: 0.0794 - val_accuracy: 0.9768
Epoch 19/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0550 -
accuracy: 0.9853 - val_loss: 0.0779 - val_accuracy: 0.9770
Epoch 20/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0515 -
accuracy: 0.9864 - val_loss: 0.0767 - val_accuracy: 0.9770
Epoch 21/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0483 -
accuracy: 0.9874 - val_loss: 0.0748 - val_accuracy: 0.9778
Epoch 22/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0453 -
accuracy: 0.9883 - val_loss: 0.0757 - val_accuracy: 0.9782
Epoch 23/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0428 -
accuracy: 0.9894 - val_loss: 0.0728 - val_accuracy: 0.9778
Epoch 24/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0404 -
accuracy: 0.9898 - val_loss: 0.0709 - val_accuracy: 0.9784
Epoch 25/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0380 -
accuracy: 0.9906 - val_loss: 0.0716 - val_accuracy: 0.9786
Epoch 26/30
```

```
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0356 -
accuracy: 0.9914 - val_loss: 0.0712 - val_accuracy: 0.9786
Epoch 27/30
1719/1719 [==============================] - 6s 3ms/step - loss: 0.0337 -
accuracy: 0.9920 - val_loss: 0.0718 - val_accuracy: 0.9786
Epoch 28/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0317 -
accuracy: 0.9930 - val_loss: 0.0689 - val_accuracy: 0.9784
Epoch 29/30
1719/1719 [==============================] - 6s 4ms/step - loss: 0.0298 -
accuracy: 0.9934 - val_loss: 0.0683 - val_accuracy: 0.9788
Epoch 30/30
1719/1719 [==============================] - 7s 4ms/step - loss: 0.0282 -
accuracy: 0.9940 - val_loss: 0.0696 - val_accuracy: 0.9772
```

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

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

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

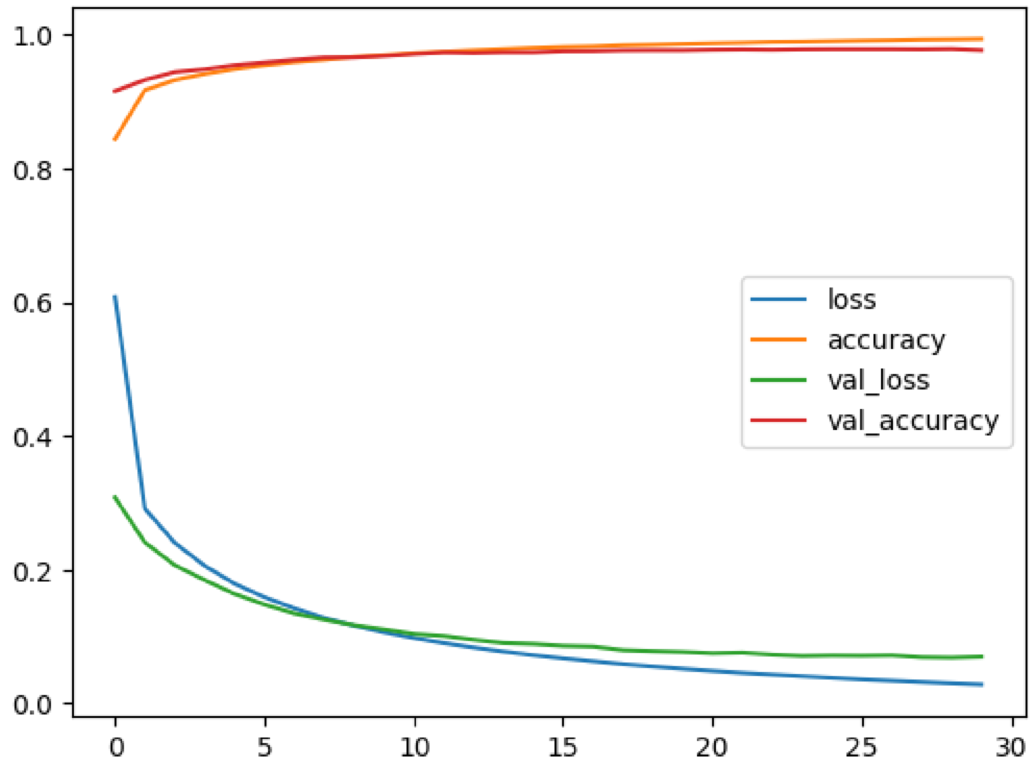
[ ]:

	loss	accuracy	val_loss	val_accuracy
0	0.607868	0.844218	0.307728	0.9160
1	0.291235	0.917382	0.240729	0.9326
2	0.240074	0.932382	0.206504	0.9442
3	0.205666	0.941400	0.184353	0.9486
4	0.179019	0.948691	0.163580	0.9546
5	0.158637	0.954491	0.147691	0.9586
6	0.142117	0.959382	0.134518	0.9626
7	0.127722	0.962909	0.125068	0.9660
8	0.116085	0.967055	0.116831	0.9662
9	0.106178	0.969655	0.110586	0.9684
10	0.097589	0.972545	0.103613	0.9712
11	0.090037	0.974691	0.100573	0.9736
12	0.083350	0.976782	0.094939	0.9728
13	0.077338	0.978564	0.090148	0.9738
14	0.071936	0.980382	0.088699	0.9738
15	0.067166	0.981782	0.085830	0.9756
16	0.062855	0.982618	0.084756	0.9758
17	0.058448	0.984473	0.079437	0.9768
18	0.054994	0.985273	0.077934	0.9770
19	0.051508	0.986418	0.076740	0.9770
20	0.048302	0.987400	0.074771	0.9778
21	0.045329	0.988255	0.075667	0.9782
22	0.042787	0.989400	0.072750	0.9778
23	0.040374	0.989764	0.070873	0.9784
24	0.038045	0.990564	0.071561	0.9786

25	0.035601	0.991400	0.071233	0.9786
26	0.033651	0.992036	0.071750	0.9786
27	0.031717	0.993000	0.068914	0.9784
28	0.029819	0.993400	0.068341	0.9788
29	0.028168	0.994018	0.069624	0.9772

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

[ ]: <Axes: >



[ ]:
```
model_clf.evaluate(X_test , y_test)
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0690 -
accuracy: 0.9777
```

[ ]: [0.06899382919073105, 0.9776999950408936]

SAMPLE CHECK - TEST DATA

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

[ ]:
```
x_new
```

[ ]: 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.]],

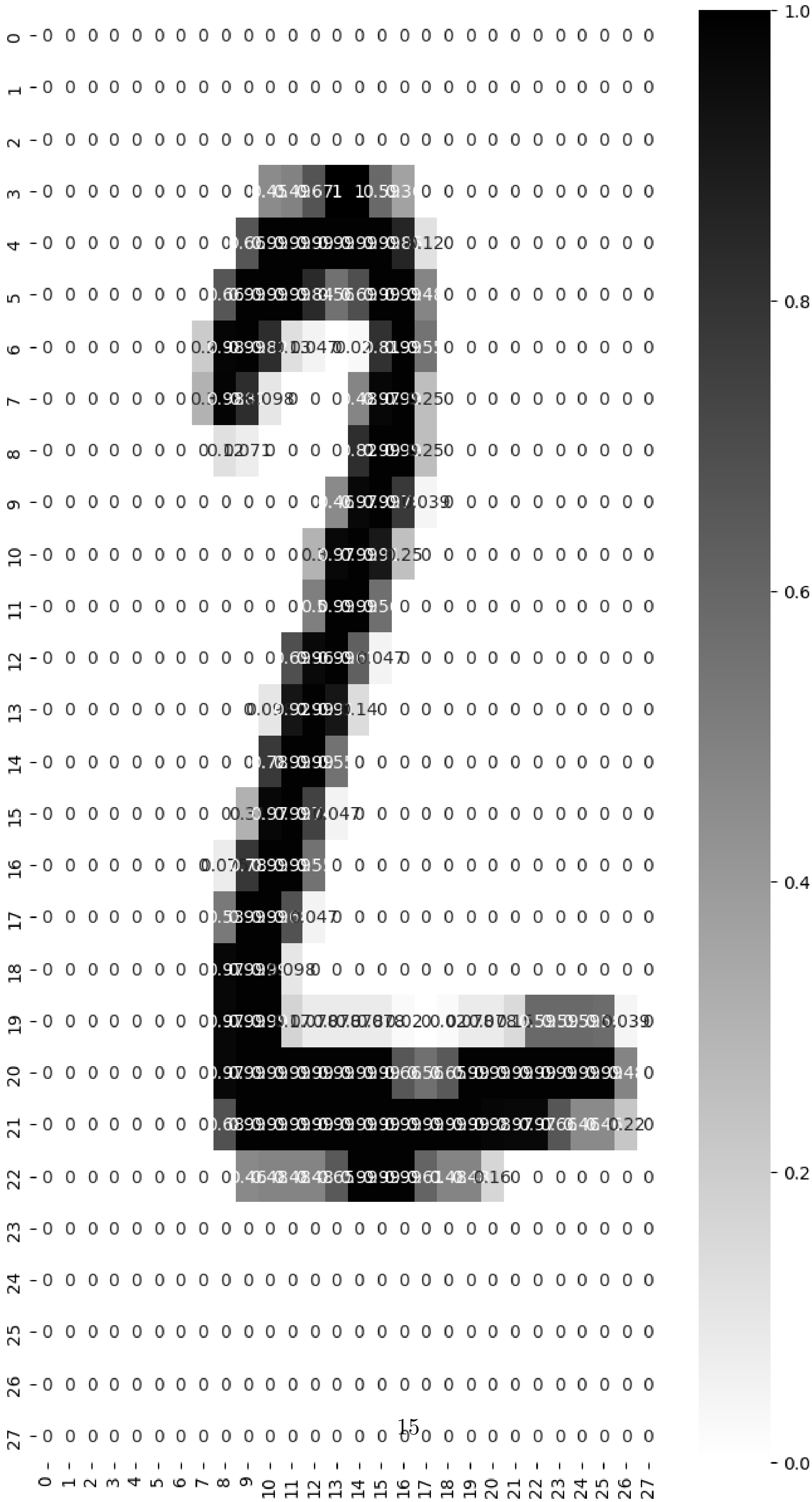
        [[0., 0., 0., …, 0., 0., 0.],
         [0., 0., 0., …, 0., 0., 0.],
         [0., 0., 0., …, 0., 0., 0.],
         …,
         [0., 0., 0., …, 0., 0., 0.],
         [0., 0., 0., …, 0., 0., 0.],
         [0., 0., 0., …, 0., 0., 0.]]])

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

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

[ ]:
```
plt.figure(figsize=(8,15))
sns.heatmap(X_test[1] , annot=True,cmap = "binary")
```

[ ]: <Axes: >



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

```
1/1 [==============================] - 0s 79ms/step
```

[ ]:
```
array([[0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   , 1.   , 0.   ,
        0.   ],
       [0.   , 0.   , 1.   , 0.   , 0.   , 0.   , 0.   , 0.   , 0.   ,
        0.   ],
       [0.   , 0.997, 0.   , 0.   , 0.   , 0.   , 0.   , 0.001, 0.001,
        0.   ]], dtype=float32)
```

[ ]:
```
y_prob
```

[ ]:
```
array([[6.6345700e-07, 3.4693590e-09, 3.1537584e-05, 2.4579174e-04,
        3.1278964e-09, 4.1930657e-06, 2.9685993e-12, 9.9970031e-01,
        3.4255977e-06, 1.4072972e-05],
       [1.1369716e-06, 8.0029145e-05, 9.9970919e-01, 1.9793920e-04,
        1.6554232e-10, 1.3252111e-07, 1.6247924e-06, 3.9134309e-09,
        9.9291683e-06, 2.8132382e-13],
       [3.4066672e-06, 9.9740076e-01, 2.1284110e-04, 6.8445959e-05,
        3.5438739e-04, 3.0558334e-05, 4.1828876e-05, 1.0401690e-03,
        8.2951097e-04, 1.7992008e-05]], dtype=float32)
```

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

[ ]:
```
y_pred
```

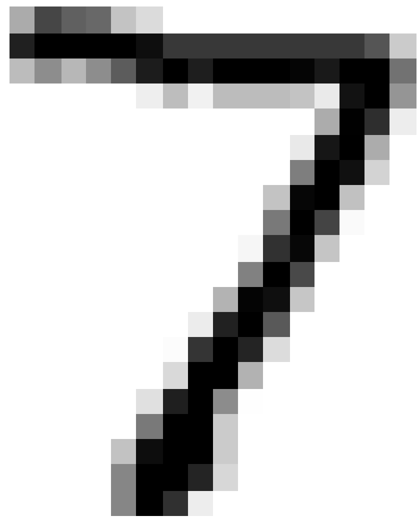
[ ]:
```
array([7, 2, 1])
```

[ ]:
```
actual
```

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

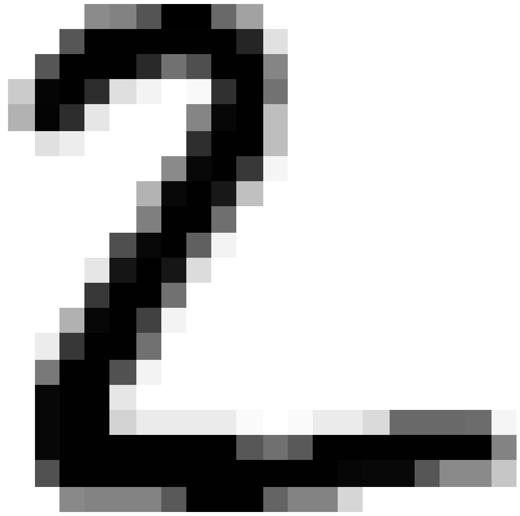
[ ]:
```
## plot

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(pred)and Actual 7



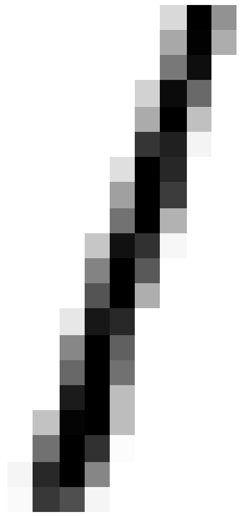
#######

Predicted(pred)and Actual 2



#######

Predicted(pred)and Actual 1



#######

[ ]: