2021

Cairo University

Faculty of computers and artificial intelligence

AI322

Supervised Learning

Assignment_3

CNN

Name: Gehad Mustafa Mansour Refae

SUPERVISED LEARNING

STUD 20180080MNIST DATASET

Default code:

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from subprocess import check_output
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.image as mpimg
import numpy as np
from numpy import random
import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, M
axPooling2D
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau, Callback
from keras import regularizers
from keras.optimizers import Adam
import tensorflow as tf
from keras.optimizers import SGD
from sklearn.utils import shuffle
data_train = pd.read_csv('/content/mnist_train.csv')
data_test = pd.read_csv('/content/mnist_test.csv')
print("data_train" , data_train.shape)
print("data_test" , data_test.shape)
img_rows, img_cols = 28, 28
input_shape = (img_rows, img_cols, 1)
X = np.array(data_train.iloc[:, 1:])
y = to_categorical(np.array(data_train.iloc[:, 0]))
#Here we split validation data to optimiza classifier during training
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2,
random_state=13)
#Test data
X_test = np.array(data_test.iloc[:, 1:])
y_test = to_categorical(np.array(data_test.iloc[:, 0]))
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
X_train, y_train = shuffle(X_train, y_train)
X_test, y_test = shuffle(X_test, y_test)
X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1)
```

```
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_val = X_val.astype('float32')
X_train /= 255
X_test /= 255
X_val /= 255
print(X_train)
print("X_train:{}\ny_train:{}\n\nX_val:{}\ny_val:{}\n\nX_test:{}\ny_tes
t:{}".format
(X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.shape,
y_test.shape))
batch_size = 32
num_classes = 10
epochs = 5
#input image dimensions
img_rows, img_cols = 28, 28

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
activation='relu',
kernel_initializer='he_normal',
input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
# opt = SGD(lr=0.001, momentum=0.9)
# opt = keras.optimizers.SGD(learning_rate=0.001)
opt = keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=[
'accuracy'])

# model.compile(loss = 'categorical_crossentropy', optimizer = 'adam',
metrics = ['accuracy'])
model.summary()
history = model.fit(X_train, y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1,
validation_data=(X_val, y_val))
score = model.evaluate(X_test, y_test, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
#get the predictions for the test data
predicted_classes = model.predict_classes(X_test)
```

```
#get the indices to be plotted
y_true = data_test.iloc[:, 0]
from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i in range(num_classes)]
print(classification_report(y_true, predicted_classes, target_names=tar
get_names))
```

Epochs:

Epoch	Accuracy	Loss
5	0.9828000068664551	0.06191356107592583
10	0.991100013256073	0.03980569168925285
15	0.9883000254631042	0.0770006999373436
30	0.9897000193595886	0.09153448790311813

- We will choose number of epochs = 10 because it has high accuracy, take little time and good loss.

Learning Rates:

Learning Rate	Accuracy	Loss
0.001	0. 991100013256073	0. 03980569168925285
0.005	0.980400025844574	0.10530269891023636
0.05	0.10279999673366547	2.3154354095458984

- We will choose learning rate = 0.001 because it has high accuracy, take little time and good loss.

Models:

- ## Model 1 :

✓
```
batch_size = 32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28


model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',

input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(128, activation='relu'))


model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.Adam(learning_rate=0.001)

model.compile(optimizer=opt, loss='categorical_crossentropy', m
etrics=['accuracy'])
```

- ✓ Test loss: 0. 03980569168925285
- ✓ Test accuracy: 0. 991100013256073
- ✓ Total params: 241,546
- ✓ Trainable params: 241,546
- ✓ Non-trainable params: 0

```
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)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
dense (Dense)                (None, 128)               147584
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
Total params: 241,546
Trainable params: 241,546
```

- **Model 2 :**

✓
```
batch_size =32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',

input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Flatten())

model.add(Dense(64, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.Adam(learning_rate=0.001)
```

```
model.compile(optimizer=opt, loss='categorical_crossentropy', m
etrics=['accuracy'])
```

- ✓ Test accuracy: 0.9886999726295471
- ✓ Test loss: 0.048302967101335526
- ✓ Total params: 121,930
- ✓ Trainable params: 121,930
- ✓ Non-trainable params: 0

```
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)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 64)                102464
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 121,930
Trainable params: 121,930
```

- **Model 3:**

✓
```
batch_size = 32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',

input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
```

```
model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.Adam(learning_rate=0.001)

model.compile(optimizer=opt, loss='categorical_crossentropy', m
etrics=['accuracy'])
```

- ✓ Loss: 0.07243524491786957
- ✓ Accuracy : 0.989300012588501
- ✓ Total params: 2,021,194
- ✓ Trainable params: 2,021,194
- ✓ Non-trainable params: 0

```
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, 128)       36992
_________________________________________________________________
flatten (Flatten)            (None, 15488)             0
_________________________________________________________________
dense (Dense)                (None, 128)               1982592
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1290
=================================================================
```

• Model 4:

✓
```
batch_size = 32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28

model = Sequential()

model.add(Conv2D(30, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',

input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
```

```
model.add(Flatten())

model.add(Dense(30, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.Adam(learning_rate=0.001)

model.compile(optimizer=opt, loss='categorical_crossentropy', me
trics=['accuracy'])
```

- ✓ Loss: 0.051949601620435715
- ✓ Accuracy: 0.984499990940094
- ✓ Total params: 152,740
- ✓ Trainable params: 152,740
- ✓ Non-trainable params: 0

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 30)        300
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 30)        0
_________________________________________________________________
flatten (Flatten)            (None, 5070)              0
_________________________________________________________________
dense (Dense)                (None, 30)                152130
_________________________________________________________________
dense_1 (Dense)              (None, 10)                310
=================================================================
```

- **Model 5:**

✓
```
batch_size = 32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28

model = Sequential()

model.add(Conv2D(64, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',
```

```
input_shape=input_shape))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
model.add(Dense(num_classes, activation='softmax'))
opt = keras.optimizers.Adam(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy',
 metrics=['accuracy'])
```

- ✓ Loss: 0.0526624396443367
- ✓ Accuracy: 0.9891999959945679
- ✓ Total params: 485,514
- ✓ Trainable params: 485,514
- ✓ Non-trainable params: 0

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 64)	640
max_pooling2d (MaxPooling2D)	(None, 13, 13, 64)	0
conv2d_1 (Conv2D)	(None, 11, 11, 128)	73856
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 128)	0
flatten (Flatten)	(None, 3200)	0
dense (Dense)	(None, 128)	409728
dense_1 (Dense)	(None, 10)	1290

- We will choose Model 1 because it has high accuracy = %99.11 , take little time , good loss = %3.9 and minimum number of parameters .

Batch Sizes:

Batch Size	Accuracy	Loss
32	0.991100013256073	0. 03980569168925285
64	0.9897000193595886	0.04299784079194069
128	0.9908000230789185	0.03621158003807068

- We will choose batch size = 32 because it has high accuracy, take little time and good loss.

Activation Functions:

Activation Function	Accuracy	Loss
ReLU	0. 991100013256073	0. 03980569168925285
tanh	0.9901000261306763	0.03669922053813934
sigmoid	0.9904999732971191	0.03228173032402992

- We will choose activation function: ReLU because it has high accuracy, take little time and good loss.

Optimizers:

Optimizer	Accuracy	Loss
Adam	0. 991100013256073	0. 03980569168925285
SGD	0.9729999899864197	0.09458785504102707
Adamax	0.9914000034332275	0.030355028808116913

- We will choose optimizer: Adamax because it has high accuracy, take little time and good loss.

Dropout:

- **First:**

```
batch_size = 32
num_classes = 10
epochs = 10
#input image dimensions
img_rows, img_cols = 28, 28

model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
activation='relu',
kernel_initializer='he_normal',
input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
```

- `model.add(Dropout(0.25))`

```
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
```

- `model.add(Dropout(0.4))`

```
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
```

- `model.add(Dropout(0.3))`

```
model.add(Dense(num_classes, activation='softmax'))
opt = keras.optimizers.Adamax(learning_rate=0.001)
```

```
model.compile(optimizer=opt, loss='categorical_crossentropy', metric
s=['accuracy'])
```

- ✓ Loss: 0.021553359925746918
- ✓ Test accuracy: 0.9929999709129333
- ✓ When we add `model.add(Dropout(0.25))` eliminate about a quarter of the neurans from the first conv2D layer.
- ✓ When we add `model.add(Dropout(0.4))` eliminate about a `%40` of the neurans from the second conv2D layer.
- ✓ When we add `model.add(Dropout(0.3))` eliminate about a `%30` of the neurans from the third conv2D layer.

- **Second:**

```
batch_size = 32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',

input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
```

- `model.add(Dropout(0.35))`

```
model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.Adamax(learning_rate=0.001)

model.compile(optimizer=opt, loss='categorical_crossentropy', metric
s=['accuracy'])
```

- ✓ Loss: 0.026227835565805435
- ✓ Test accuracy: 0.9915000200271606
- ✓ When we add `model.add(Dropout(0.35))` eliminate about a %35 of the neurans from the first conv2D layer.

• Third:

```
batch_size = 32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28


model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',

input_shape=input_shape))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
```

- ➢ `model.add(Dropout(0.15))`

```
model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(128, activation='relu'))


model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.Adamax(learning_rate=0.001)

model.compile(optimizer=opt, loss='categorical_crossentropy', metric
s=['accuracy'])
```

- ✓ Loss: 0.028334137052297592
- ✓ Test accuracy: 0.991599977016449

✓ When we add `model.add(Dropout(0.15))` eliminate about a %15 of the neurans from the second conv2D layer.

- **Fours:**

```
#m_1
batch_size = 32
num_classes = 10
epochs = 10
#input image dimensions
img_rows, img_cols = 28, 28


model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
activation='relu',
kernel_initializer='he_normal',
input_shape=input_shape))


model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
    ➢ model.add(Dropout(0.35))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2),strides=2))
model.add(Conv2D(128, (3, 3), activation='relu'))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
    ➢ model.add(Dropout(0.70))
model.add(Dense(num_classes, activation='softmax'))
opt = keras.optimizers.Adamax(learning_rate=0.001)
model.compile(optimizer=opt, loss='categorical_crossentropy', metri
cs=['accuracy'])
```

✓ Loss: 0.030612064525485 04
✓ Test accuracy: 0.9904000163078308
✓ When we add `model.add(Dropout(0.35))` eliminate about a %35 of the neurans from the first conv2D layer.

- ✓ When we add `model.add(Dropout(0.70))` eliminate about a %70 of the neurans from the third conv2D layer.

We will choose first dropout because it has high accuracy, take little time and good loss

- **Best model with test batch size:**

```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_c
sv)
from subprocess import check_output
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
import matplotlib
import matplotlib.image as mpimg
import numpy as np
from numpy import random
import keras
from keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, Dropout, Activation, Flatten, Conv2
D, MaxPooling2D
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator
from keras.callbacks import ReduceLROnPlateau, Callback
from keras import regularizers
from keras.optimizers import Adam
import tensorflow as tf
from keras.optimizers import SGD
from sklearn.utils import shuffle

data_train = pd.read_csv('/content/mnist_train.csv')
data_test = pd.read_csv('/content/mnist_test.csv')
print("data_train" , data_train.shape)
print("data_test" , data_test.shape)
img_rows, img_cols = 28, 28
input_shape = (img_rows, img_cols, 1)
X = np.array(data_train.iloc[:, 1:])
y = to_categorical(np.array(data_train.iloc[:, 0]))
#Here we split validation data to optimiza classifier during traini
ng
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.
2, random_state=13)
#Test data
X_test = np.array(data_test.iloc[:, 1:])
```

```
y_test = to_categorical(np.array(data_test.iloc[:, 0]))
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)
X_train, y_train = shuffle(X_train, y_train)
X_test, y_test = shuffle(X_test, y_test)
X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
X_val = X_val.reshape(X_val.shape[0], img_rows, img_cols, 1)
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
X_val = X_val.astype('float32')
X_train /= 255
X_test /= 255
X_val /= 255

print(X_train)
print("X_train:{}\ny_train:{}\n\nX_val:{}\ny_val:{}\n\nX_test:{}\ny
_test:{}".format
(X_train.shape, y_train.shape, X_val.shape, y_val.shape, X_test.sha
pe, y_test.shape))
#m_1

batch_size = 32

num_classes = 10

epochs = 10

#input image dimensions

img_rows, img_cols = 28, 28

model = Sequential()

model.add(Conv2D(32, kernel_size=(3, 3),

activation='relu',

kernel_initializer='he_normal',

input_shape=input_shape))


model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Dropout(0.25))

model.add(Conv2D(64, (3, 3), activation='relu'))

model.add(MaxPooling2D(pool_size=(2, 2),strides=2))

model.add(Dropout(0.4))
```

```
model.add(Conv2D(128, (3, 3), activation='relu'))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.3))

model.add(Dense(num_classes, activation='softmax'))

opt = keras.optimizers.Adamax(learning_rate=0.001)

model.compile(optimizer=opt, loss='categorical_crossentropy', metri
cs=['accuracy'])


model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                 Output Shape              Param #
=================================================================
 conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
 max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
 dropout (Dropout)            (None, 13, 13, 32)        0
_________________________________________________________________
 conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
 max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
 dropout_1 (Dropout)          (None, 5, 5, 64)          0
_________________________________________________________________
 conv2d_2 (Conv2D)            (None, 3, 3, 128)         73856
_________________________________________________________________
 flatten (Flatten)            (None, 1152)              0
_________________________________________________________________
 dense (Dense)                (None, 128)               147584
_________________________________________________________________
 dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
 dense_1 (Dense)              (None, 10)                1290
=================================================================
```

```
Total params: 241,546
Trainable params: 241,546
Non-trainable params: 0
```

```
_________________________________
history = model.fit(X_train, y_train,
batch_size=batch_size,
epochs=epochs,
verbose=1,
validation_data=(X_val, y_val))
score = model.evaluate(X_test, y_test, verbose=0 , batch_size=128)
```

```
Epoch 1/10
1500/1500 [==============================] - 54s 36ms/step - loss: 0.6774 - accuracy: 0.7749 -
val_loss: 0.0955 - val_accuracy: 0.9693
Epoch 2/10
1500/1500 [==============================] - 53s 36ms/step - loss: 0.1390 - accuracy: 0.9576 -
val_loss: 0.0616 - val_accuracy: 0.9806
Epoch 3/10
1500/1500 [==============================] - 53s 35ms/step - loss: 0.0861 - accuracy: 0.9742 -
val_loss: 0.0488 - val_accuracy: 0.9852
Epoch 4/10
1500/1500 [==============================] - 53s 36ms/step - loss: 0.0739 - accuracy: 0.9763 -
val_loss: 0.0453 - val_accuracy: 0.9850
Epoch 5/10
1500/1500 [==============================] - 53s 36ms/step - loss: 0.0665 - accuracy: 0.9792 -
val_loss: 0.0374 - val_accuracy: 0.9881
Epoch 6/10
1500/1500 [==============================] - 53s 35ms/step - loss: 0.0545 - accuracy: 0.9829 -
val_loss: 0.0357 - val_accuracy: 0.9887
Epoch 7/10
1500/1500 [==============================] - 53s 35ms/step - loss: 0.0525 - accuracy: 0.9839 -
val_loss: 0.0373 - val_accuracy: 0.9883
Epoch 8/10
1500/1500 [==============================] - 53s 35ms/step - loss: 0.0457 - accuracy: 0.9858 -
val_loss: 0.0339 - val_accuracy: 0.9891
Epoch 9/10
1500/1500 [==============================] - 53s 35ms/step - loss: 0.0433 - accuracy: 0.9867 -
val_loss: 0.0314 - val_accuracy: 0.9902
Epoch 10/10
1500/1500 [==============================] - 52s 34ms/step - loss: 0.0384 - accuracy: 0.9877 -
val_loss: 0.0314 - val_accuracy: 0.9908
```

```
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

```
Test loss: 0.023057807236909866
```

```
Test accuracy: 0.9926000237464905
```

```
#get the predictions for the test data
predicted_classes = model.predict_classes(X_test)
#get the indices to be plotted
y_true = data_test.iloc[:, 0]
from sklearn.metrics import classification_report
target_names = ["Class {}".format(i) for i in range(num_classes)]
print(classification_report(y_true, predicted_classes, target_names
=target_names))
```

```
              precision    recall  f1-score   support

     Class 0       0.09      0.09      0.09       980
     Class 1       0.13      0.13      0.13      1135
     Class 2       0.12      0.12      0.12      1032
     Class 3       0.10      0.10      0.10      1010
     Class 4       0.09      0.09      0.09       982
     Class 5       0.09      0.09      0.09       892
     Class 6       0.10      0.10      0.10       958
     Class 7       0.10      0.10      0.10      1028
     Class 8       0.10      0.10      0.10       974
     Class 9       0.11      0.11      0.11      1009

    accuracy                           0.10     10000
   macro avg       0.10      0.10      0.10     10000
weighted avg       0.10      0.10      0.10     10000
```

Summary:

Epochs	Learning Rate	Batch size	Activation function	optimizers	Accuracy	Loss
5	**0.001**	**32**	**relu**	**Adam**	**0.982**	**0.061**
10	**0.001**	**32**	**relu**	**Adam**	**0.991**	**0.039**
15	**0.001**	**32**	**relu**	**Adam**	**0.988**	**0.077**

30	**0.001**	**32**	**relu**	**Adam**	**0.989**	**0.091**
10	**0.005**	**32**	**relu**	**Adam**	**0.980**	**0.105**
10	**0.05**	**32**	**relu**	**Adam**	**0.102**	**2.315**
10	**0.001**	**64**	**relu**	**Adam**	**0.989**	**0.042**
10	**0.001**	**128**	**relu**	**Adam**	**0.990**	**0.036**
10	**0.001**	**32**	**tanh**	**Adam**	**0.990**	**0.036**
10	**0.001**	**32**	**sigmoid**	**Adam**	**0.990**	**0.032**
10	**0.001**	**32**	**relu**	**SGD**	**0.972**	**0.0945**
10	**0.001**	**32**	**relu**	**Adamax**	**0.9914**	**0.0303**