The goal is to implement a CNN to classify MNIST handwritten digit images using Python

Fetching the MNIST handwritten digit classification dataset-

```
# example of loading the mnist dataset
from tensorflow.keras.datasets import mnist
from matplotlib import pyplot as plt
# load dataset
(trainX, trainy), (testX, testy) = mnist.load_data()
# summarize loaded dataset
print('Train: X=%s, y=%s' % (trainX.shape, trainy.shape))
print('Test: X=%s, y=%s' % (testX.shape, testy.shape))
# plot first few images
for i in range(9):
# define subplot
plt.subplot(330 + 1 + i)
# plot raw pixel data
plt.imshow(trainX[i], cmap=plt.get_cmap('gray'))
# show the figure
plt.show()
```

Running the example loads the MNIST train and test dataset and prints their shape.

Although the MNIST dataset is effectively solved, it can be a useful starting point for developing and practicing a methodology for solving image classification tasks using convolutional neural networks.

We can load the images and reshape the data arrays to have a single-color channel-

```
# load dataset

(trainX, trainY), (testX, testY) = mnist.load_data()

# reshape dataset to have a single channel

trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))

testX = testX.reshape((testX.shape[0], 28, 28, 1))
```

The load_dataset() function implements these behaviors and can be used to load the dataset-

```
# load train and test dataset

def load_dataset():

# load dataset

(trainX, trainY), (testX, testY) = mnist.load_data()

# reshape dataset to have a single channel

trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))

testX = testX.reshape((testX.shape[0], 28, 28, 1))

# one hot encode target values

trainY = to_categorical(trainY)

testY = to_categorical(testY)

return trainX, trainY, testX, testY
```

We know that the pixel values for each image in the dataset are unsigned integers in the range between black and white, or 0 and 255. We do not know the best way to scale the pixel values for modeling, but we know that some scaling will be required. Normalizing Pixel Values-

```
# convert from integers to floats

train_norm = train.astype('float32')

test_norm = test.astype('float32')

# normalize to range 0-1

train_norm = train_norm / 255.0

test_norm = test_norm / 255.0
```

Function must be called to prepare the pixel values prior to any modeling-

```
# scale pixels

def prep_pixels(train, test):

# convert from integers to floats

train_norm = train.astype('float32')

test_norm = test.astype('float32')

# normalize to range 0-1

train_norm = train_norm / 255.0
```

```
test_norm = test_norm / 255.0

# return normalized images

return train_norm, test_norm
```

Next, we need to define a baseline convolutional neural network model for the problem-

```
# evaluate a model using k-fold cross-validation

def evaluate_model(dataX, dataY, n_folds=5):

scores, histories = list(), list()

# prepare cross validation

kfold = KFold(n_folds, shuffle=True, random_state=1)

# enumerate splits

for train_ix, test_ix in kfold.split(dataX):

# define model

model = define_model()

# select rows for train and test

trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]

# fit model

history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)

# evaluate model

_, acc = model.evaluate(testX, testY, verbose=0)

print('> %.3f' % (acc * 100.0))

# stores scores

scores.append(acc)

histories.append(history)

return scores, histories
```

Once the model has been evaluated, we can present the results-

```
# plot diagnostic learning curves

def summarize_diagnostics(histories):

for i in range(len(histories)):

# plot loss

plt.subplot(2, 1, 1)

plt.title('Cross Entropy Loss')

plt.plot(histories[i].history['loss'], color='blue', label='train')

plt.plot(histories[i].history['val_loss'], color='orange', label='test')

# plot accuracy

plt.subplot(2, 1, 2)
```

```
plt.title('Classification Accuracy')

plt.plot(histories[i].history['accuracy'], color='blue', label='train')

plt.plot(histories[i].history['val_accuracy'], color='orange', label='test')

plt.show()
```

<u>The *summarize_performance()* function below implements this for a given list of scores collected during model evaluation-</u>

```
# summarize model performance

def summarize_performance(scores):

# print summary

print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))

# box and whisker plots of results

plt.boxplot(scores)

plt.show()
```

We need a function that will drive the test harness. This involves calling all the defined functions-

```
# run the test harness for evaluating a model

def run_test_harness():

# load dataset

trainX, trainY, testX, testY = load_dataset()

# prepare pixel data

trainX, testX = prep_pixels(trainX, testX)

# evaluate model

scores, histories = evaluate_model(trainX, trainY)

# learning curves

summarize_diagnostics(histories)

# summarize estimated performance

summarize_performance(scores)
```

<u>We now have everything we need; the complete code example for a baseline convolutional neural network model on the MNIST dataset-</u>

```
def load_dataset():

# load dataset

(trainX, trainY), (testX, testY) = mnist.load_data()

# reshape dataset to have a single channel

trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))

testX = testX.reshape((testX.shape[0], 28, 28, 1))
```

```
# one hot encode target values
trainY = to_categorical(trainY)
testY = to_categorical(testY)
return trainX, trainY, testX, testY
# scale pixels
def prep_pixels(train, test):
# convert from integers to floats
train_norm = train.astype('float32')
test_norm = test.astype('float32')
# normalize to range 0-1
train_norm = train_norm / 255.0
test_norm = test_norm / 255.0
# return normalized images
return train_norm, test_norm
# define cnn model
def define_model():
model = Sequential()
model.add(Conv2D(32, (3, 3), activation='relu', kernel_initializer='he_uniform', input_shape=(28, 28, 1)))
model.add(MaxPooling2D((2, 2)))
model.add(Flatten())
model.add(Dense(100, activation='relu', kernel_initializer='he_uniform'))
model.add(Dense(10, activation='softmax'))
# compile model
opt = SGD(learning_rate=0.01, momentum=0.9)
model.compile(optimizer=opt, loss='categorical_crossentropy', metrics=['accuracy'])
return model
# evaluate a model using k-fold cross-validation
def evaluate_model(dataX, dataY, n_folds=5):
scores, histories = list(), list()
# prepare cross validation
kfold = KFold(n_folds, shuffle=True, random_state=1)
# enumerate splits
for train_ix, test_ix in kfold.split(dataX):
# define model
model = define_model()
# select rows for train and test
```

```
trainX, trainY, testX, testY = dataX[train_ix], dataY[train_ix], dataX[test_ix], dataY[test_ix]
# fit model
history = model.fit(trainX, trainY, epochs=10, batch_size=32, validation_data=(testX, testY), verbose=0)
# evaluate model
_, acc = model.evaluate(testX, testY, verbose=0)
print('> %.3f' % (acc * 100.0))
# stores scores
scores.append(acc)
histories.append(history)
return scores, histories
# plot diagnostic learning curves
def summarize_diagnostics(histories):
for i in range(len(histories)):
# plot loss
plt.subplot(2, 1, 1)
plt.title('Cross Entropy Loss')
plt.plot(histories[i].history['loss'], color='blue', label='train')
plt.plot(histories[i].history['val_loss'], color='orange', label='test')
# plot accuracy
plt.subplot(2, 1, 2)
plt.title('Classification Accuracy')
plt.plot(histories[i].history['accuracy'], color='blue', label='train')
plt.plot(histories[i].history['val_accuracy'], color='orange', label='test')
plt.show()
# summarize model performance
def summarize_performance(scores):
# print summary
print('Accuracy: mean=%.3f std=%.3f, n=%d' % (mean(scores)*100, std(scores)*100, len(scores)))
# box and whisker plots of results
plt.boxplot(scores)
plt.show()
# run the test harness for evaluating a model
def run_test_harness():
# load dataset
trainX, trainY, testX, testY = load_dataset()
# prepare pixel data
```

```
trainX, testX = prep_pixels(trainX, testX)
# evaluate model
scores, histories = evaluate_model(trainX, trainY)
# learning curves
summarize_diagnostics(histories)
# summarize estimated performance
summarize_performance(scores)
# entry point, run the test harness
run_test_harness()
```

DONE BY-

Sriman Satwik Reddy Chinnam

Yeruva Suprith Reddy

P. Rithvik Rao