Bilenkin550Week11_Exercise_11.2

May 23, 2025

0.0.1 11.2 Exercise: Building a CNN Image Classifier

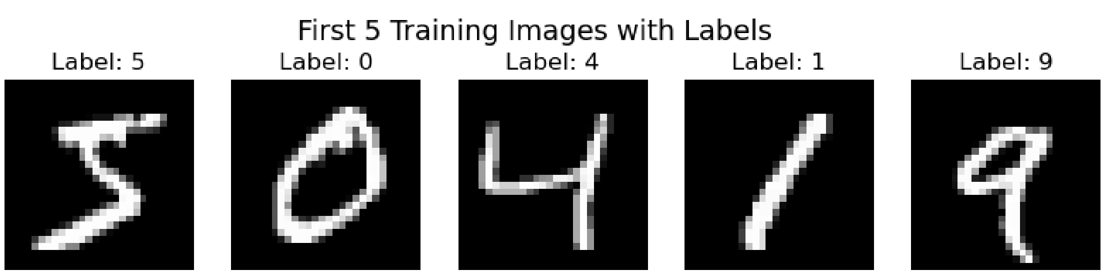
1. Load the MNIST data set.

[260]:
```
from tensorflow.keras.datasets import mnist

# Loading the dataset
(x_train, y_train), (x_test, y_test) = mnist.load_data()

# Showing the shapes
print("Training data shape:", x_train.shape)
print("Test data shape:", x_test.shape)
```

```
Training data shape: (60000, 28, 28)
Test data shape: (10000, 28, 28)
```

2. Display the first five images in the training data set and compare them to the first five training labels.

[261]:
```
import matplotlib.pyplot as plt

# Displaying the first 5 images
plt.figure(figsize=(10, 2))
for i in range(5):
    plt.subplot(1, 5, i + 1)
    plt.imshow(x_train[i], cmap='gray')
    plt.title(f"Label: {y_train[i]}")
    plt.axis('off')
plt.suptitle("First 5 Training Images with Labels", fontsize=14, y=1.05)
plt.show()
```



3. Build and train a Keras CNN classifier on the MNIST training set.

[262]:
```python
import tensorflow as tf
from tensorflow.keras import layers, models
from IPython.display import Markdown, display

# Normalizing images
x_train_norm = x_train / 255.0
x_test_norm = x_test / 255.0

# Reshaping for grayscale channel
x_train_norm = x_train_norm.reshape(-1, 28, 28, 1)
x_test_norm = x_test_norm.reshape(-1, 28, 28, 1)

# Building CNN model
model = models.Sequential([
    layers.Input(shape=(28, 28, 1)),
    layers.Conv2D(32, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),
    layers.Flatten(),
    layers.Dense(64, activation='relu'),
    layers.Dense(10, activation='softmax')
])

# Compiling model
model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

# Creating formatted Markdown table
header = "| **Name**          | **Type**      | **Output Shape**    | **Params**  ␣
↪|\n"
header␣
↪+="|------------------|---------------|--------------------|-------------|\n"
rows = ""

for layer in model.layers:
    name = layer.name[:16].ljust(16)
    layer_type = layer.__class__.__name__.ljust(15)
    output_shape = str(layer.output.shape)[:21].ljust(21)
    params = f"{layer.count_params():,}".rjust(10)
    rows += f"| {name} | {layer_type} | {output_shape} | {params} |\n"
```

```python
# Displaying table as Markdown so it will fit well the PDF page width
display(Markdown(header + rows))
```

Name	Type	Output Shape	Params
conv2d_88	Conv2D	(None, 26, 26, 32)	320
max_pooling2d_88	MaxPooling2D	(None, 13, 13, 32)	0
conv2d_89	Conv2D	(None, 11, 11, 64)	18,496
max_pooling2d_89	MaxPooling2D	(None, 5, 5, 64)	0
flatten_44	Flatten	(None, 1600)	0
dense_88	Dense	(None, 64)	102,464
dense_89	Dense	(None, 10)	650

Train the above CNN Model

[263]:
```python
history = model.fit(x_train_norm, y_train, epochs=5, batch_size=64,␣
↪validation_split=0.1)
```

```
Epoch 1/5
844/844                7s 7ms/step -
accuracy: 0.8615 - loss: 0.4415 - val_accuracy: 0.9822 - val_loss: 0.0589
Epoch 2/5
844/844                6s 7ms/step -
accuracy: 0.9798 - loss: 0.0619 - val_accuracy: 0.9887 - val_loss: 0.0411
Epoch 3/5
844/844                6s 7ms/step -
accuracy: 0.9876 - loss: 0.0425 - val_accuracy: 0.9867 - val_loss: 0.0508
Epoch 4/5
844/844                6s 7ms/step -
accuracy: 0.9903 - loss: 0.0320 - val_accuracy: 0.9873 - val_loss: 0.0397
Epoch 5/5
844/844                7s 8ms/step -
accuracy: 0.9923 - loss: 0.0251 - val_accuracy: 0.9885 - val_loss: 0.0372
```

4. Report the test accuracy of your model.

[264]:
```python
# Evaluating the model on the test set
test_loss, test_accuracy = model.evaluate(x_test_norm, y_test, verbose=2)

print(f"Test accuracy: {test_accuracy:.4f}")
```

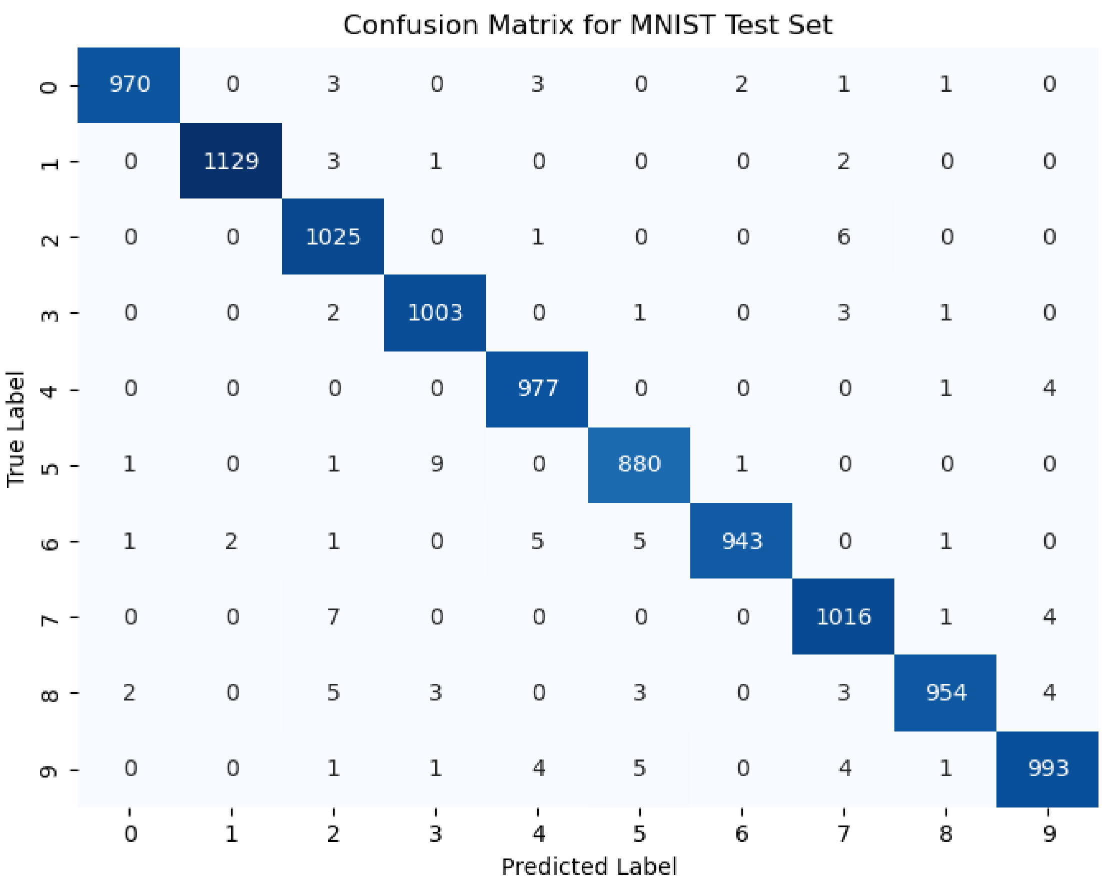
```
313/313 - 1s - 2ms/step - accuracy: 0.9890 - loss: 0.0346
Test accuracy: 0.9890
```

5. Display a confusion matrix on the test set classifications.

[265]:
```python
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

```
# Predicting the classes for the test set
y_pred_probs = model.predict(x_test_norm)  # probabilities for each class
y_pred = np.argmax(y_pred_probs, axis=1)   # convert to class labels

# Computing confusion matrix
cm = confusion_matrix(y_test, y_pred)

# Plotting confusion matrix
plt.figure(figsize=(8,6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix for MNIST Test Set')
plt.show()
```

```
313/313              1s 2ms/step
```



6. Summary of Results.

In this exercise, I built a convolutional neural network (CNN) to recognize handwritten digits from the MNIST dataset. I started by normalizing and reshaping the image data to prepare it for the model. The CNN model had two convolutional layers followed by pooling layers, then a flatten layer and two dense layers. After training the model for 5 epochs, it reached a high accuracy of 98.96% on the test set.

I also looked at the confusion matrix, which showed that most digits were predicted correctly, with only a few mistakes. This shows that the model did a good job learning how to tell the digits apart. Overall, this CNN is a good starting point for image classification tasks like digit recognition.