```
# Import libraries
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import loadmat
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
import seaborn as sns
```

```
# Download SVHN dataset (train, test)
import urllib.request

train_url = "http://ufldl.stanford.edu/housenumbers/train_32x32.mat"
test_url = "http://ufldl.stanford.edu/housenumbers/test_32x32.mat"

# Download the data
urllib.request.urlretrieve(train_url, "train_32x32.mat")
urllib.request.urlretrieve(test_url, "test_32x32.mat")
```

```
('test_32x32.mat', <http.client.HTTPMessage at 0x7e987d3f3040>)
```

```
# Load the dataset
train_data = loadmat('train_32x32.mat')
test_data = loadmat('test_32x32.mat')

# Extract images and labels
X_train = train_data['X']
y_train = train_data['y']
X_test = test_data['X']
y_test = test_data['y']
```

```
# Transpose image dimensions to (num_samples, height, width, channels)
X_train = np.transpose(X_train, (3, 0, 1, 2))
X_test = np.transpose(X_test, (3, 0, 1, 2))

# Convert labels to one-hot encoded vectors
y_train[y_train == 10] = 0
y_test[y_test == 10] = 0
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)

# Normalize the pixel values to [0, 1]
X_train = X_train / 255.0
X_test = X_test / 255.0
```

```
# Apply data augmentation
from tensorflow.keras.preprocessing.image import ImageDataGenerator

datagen = ImageDataGenerator(
    rotation_range=10,        # Rotate images up to 10 degrees
    zoom_range=0.1,           # Apply zoom augmentation
    horizontal_flip=True,     # Flip the images horizontally
    width_shift_range=0.1,    # Shift the width slightly
    height_shift_range=0.1     # Shift the height slightly
)

# Fit the data generator to the training data
datagen.fit(X_train)
```

```
# Define the CNN model
model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
```

```
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),

    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(10, activation='softmax')  # 10 classes for digit classification
])
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not pass an `input_shape`/`inpu
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])

# Model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 30, 30, 32)	896
max_pooling2d (MaxPooling2D)	(None, 15, 15, 32)	0
conv2d_1 (Conv2D)	(None, 13, 13, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 6, 6, 64)	0
conv2d_2 (Conv2D)	(None, 4, 4, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 2, 2, 128)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 128)	65,664
dense_1 (Dense)	(None, 10)	1,290

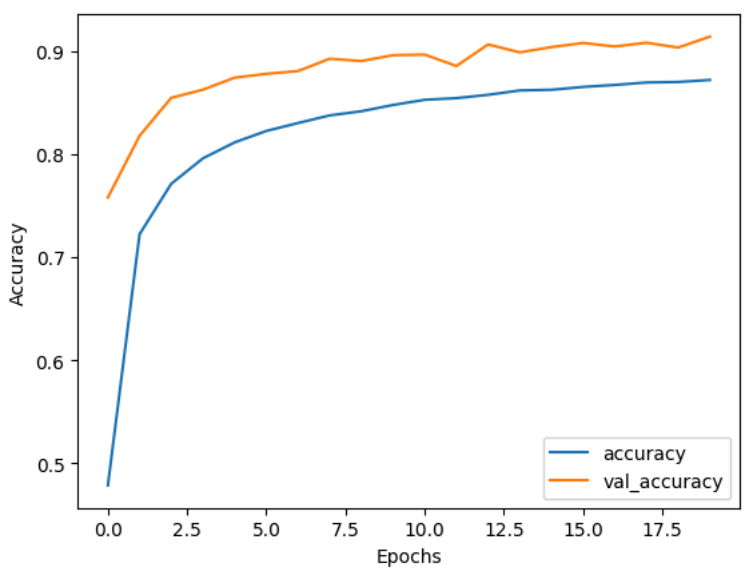
Total params: 160,202 (625.79 KB)
Trainable params: 160,202 (625.79 KB)
Non-trainable params: 0 (0.00 B)

```
# Train the model with augmented data
history = model.fit(datagen.flow(X_train, y_train, batch_size=64),
                    epochs=20,
                    validation_data=(X_test, y_test))
```

```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset` class
  self._warn_if_super_not_called()
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 164s 141ms/step - accuracy: 0.3230 - loss: 1.9071 - val_accuracy: 0.7579 - val_loss: 0.8093
Epoch 2/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 164s 143ms/step - accuracy: 0.7015 - loss: 0.9367 - val_accuracy: 0.8177 - val_loss: 0.6007
Epoch 3/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 177s 154ms/step - accuracy: 0.7626 - loss: 0.7476 - val_accuracy: 0.8546 - val_loss: 0.4985
Epoch 4/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 188s 142ms/step - accuracy: 0.7936 - loss: 0.6574 - val_accuracy: 0.8624 - val_loss: 0.4564
Epoch 5/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 163s 142ms/step - accuracy: 0.8083 - loss: 0.6106 - val_accuracy: 0.8741 - val_loss: 0.4221
Epoch 6/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 162s 142ms/step - accuracy: 0.8197 - loss: 0.5723 - val_accuracy: 0.8778 - val_loss: 0.4087
Epoch 7/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 162s 141ms/step - accuracy: 0.8284 - loss: 0.5509 - val_accuracy: 0.8805 - val_loss: 0.4115
Epoch 8/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 165s 144ms/step - accuracy: 0.8379 - loss: 0.5177 - val_accuracy: 0.8924 - val_loss: 0.3733
Epoch 9/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 199s 141ms/step - accuracy: 0.8405 - loss: 0.5096 - val_accuracy: 0.8903 - val_loss: 0.3807
Epoch 10/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 162s 141ms/step - accuracy: 0.8489 - loss: 0.4850 - val_accuracy: 0.8959 - val_loss: 0.3629
Epoch 11/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 162s 141ms/step - accuracy: 0.8533 - loss: 0.4785 - val_accuracy: 0.8965 - val_loss: 0.3626
Epoch 12/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 201s 140ms/step - accuracy: 0.8562 - loss: 0.4632 - val_accuracy: 0.8854 - val_loss: 0.3885
Epoch 13/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 163s 142ms/step - accuracy: 0.8574 - loss: 0.4558 - val_accuracy: 0.9063 - val_loss: 0.3386
```

```
Epoch 14/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 162s 141ms/step - accuracy: 0.8622 - loss: 0.4497 - val_accuracy: 0.8986 - val_loss: 0.3530
Epoch 15/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 201s 140ms/step - accuracy: 0.8615 - loss: 0.4469 - val_accuracy: 0.9038 - val_loss: 0.3331
Epoch 16/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 202s 140ms/step - accuracy: 0.8659 - loss: 0.4328 - val_accuracy: 0.9078 - val_loss: 0.3187
Epoch 17/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 162s 142ms/step - accuracy: 0.8654 - loss: 0.4332 - val_accuracy: 0.9043 - val_loss: 0.3242
Epoch 18/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 202s 142ms/step - accuracy: 0.8686 - loss: 0.4249 - val_accuracy: 0.9080 - val_loss: 0.3220
Epoch 19/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 161s 140ms/step - accuracy: 0.8692 - loss: 0.4206 - val_accuracy: 0.9033 - val_loss: 0.3327
Epoch 20/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 202s 140ms/step - accuracy: 0.8723 - loss: 0.4167 - val_accuracy: 0.9139 - val_loss: 0.3120
```

```
# Plot training and validation accuracy
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```




```
# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print("Test accuracy:", test_acc)
```

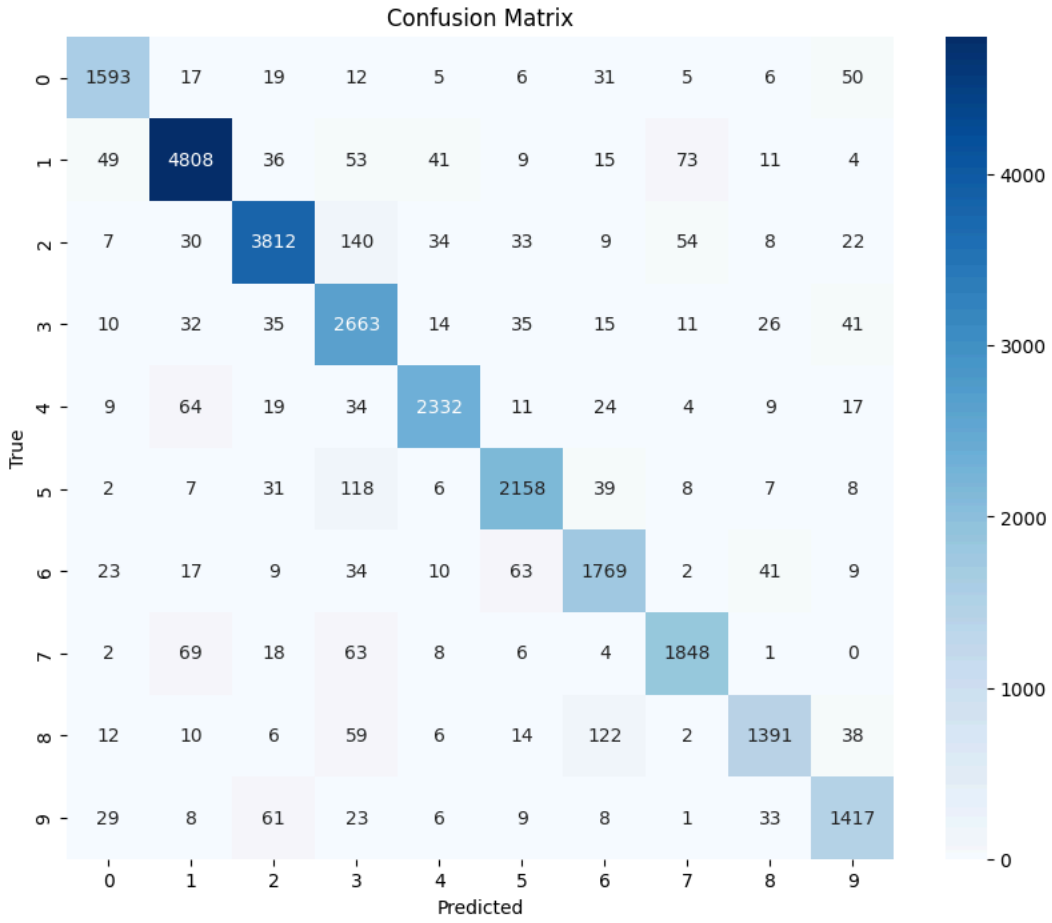
```
814/814 ━━━━━━━━━━━━━━━━━━━━ 28s 34ms/step - accuracy: 0.9101 - loss: 0.3229
Test accuracy: 0.9139136672019958
```

```
# Predict test labels
predictions = model.predict(X_test)

# Classification report
y_test_labels = np.argmax(y_test, axis=1)
y_pred_labels = np.argmax(predictions, axis=1)
print(classification_report(y_test_labels, y_pred_labels))
```

```
814/814 ━━━━━━━━━━━━━━━━━━━━ 15s 18ms/step
              precision    recall  f1-score   support

           0       0.92      0.91      0.92      1744
           1       0.95      0.94      0.95      5099
           2       0.94      0.92      0.93      4149
           3       0.83      0.92      0.88      2882
           4       0.95      0.92      0.94      2523
           5       0.92      0.91      0.91      2384
           6       0.87      0.89      0.88      1977
           7       0.92      0.92      0.92      2019
           8       0.91      0.84      0.87      1660
```

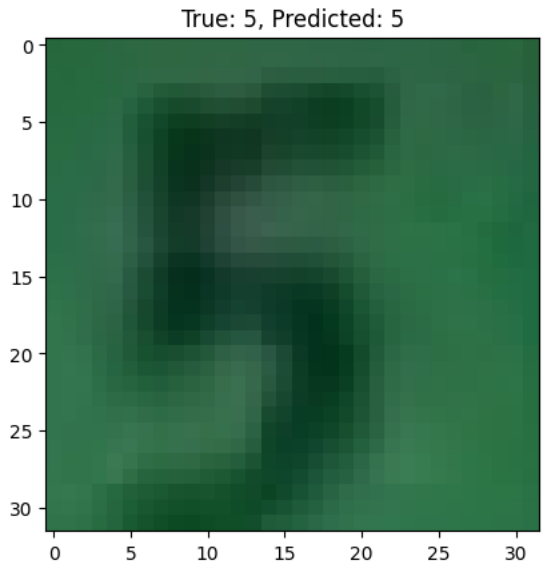
```
              9       0.88      0.89      0.89      1595

       accuracy                           0.91     26032
      macro avg       0.91      0.91      0.91     26032
   weighted avg       0.92      0.91      0.91     26032
```

```
# Confusion matrix
conf_matrix = confusion_matrix(y_test_labels, y_pred_labels)
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues')
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

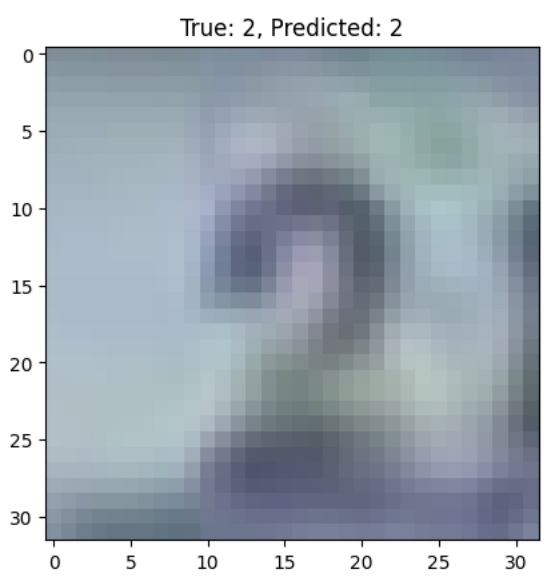



```
# Function to plot a test sample and its predicted label
def plot_sample(X, y_true, y_pred, index):
    plt.imshow(X[index])
    plt.title(f"True: {y_true[index]}, Predicted: {y_pred[index]}")
    plt.show()
```

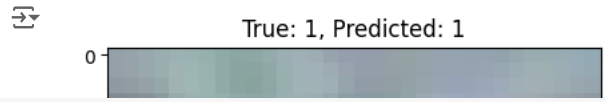
```
# # Display first 5 test samples with their true and predicted labels
# for i in range(5):
#     plot_sample(X_test, y_test_labels, y_pred_labels, i)
# Display 1st test sample
plot_sample(X_test, y_test_labels, y_pred_labels, 0)
```



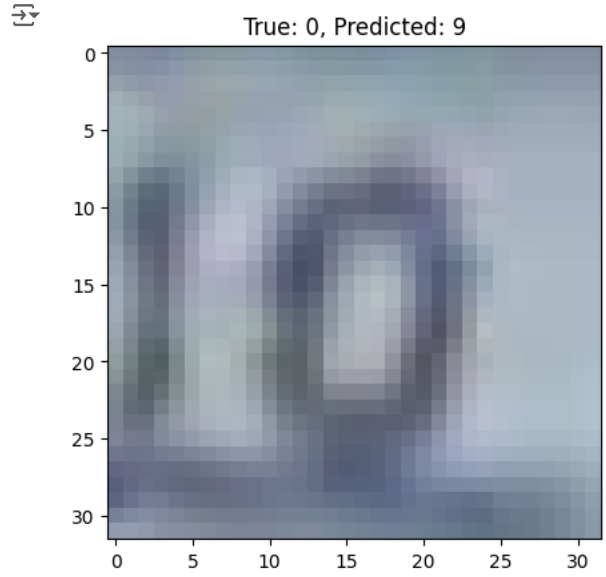

```
# Display 2nd test sample
plot_sample(X_test, y_test_labels, y_pred_labels, 1)
```




```
# Display 3rd test sample
plot_sample(X_test, y_test_labels, y_pred_labels, 2)
```




```
# Display 4th test sample
plot_sample(X_test, y_test_labels, y_pred_labels, 3)
```




```
# Display 5th test sample
plot_sample(X_test, y_test_labels, y_pred_labels, 4)
```