```
# Install required libraries (if not already installed)
# !pip install tensorflow scipy sklearn

# 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")

# 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
])

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

# Model summary
model.summary()

# 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))

# 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)

# 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))

# 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')
```

```
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)
```

```
/usr/local/lib/python3.10/dist-packages/keras/src/layers/convolutional/base_conv.py:107: UserWar
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

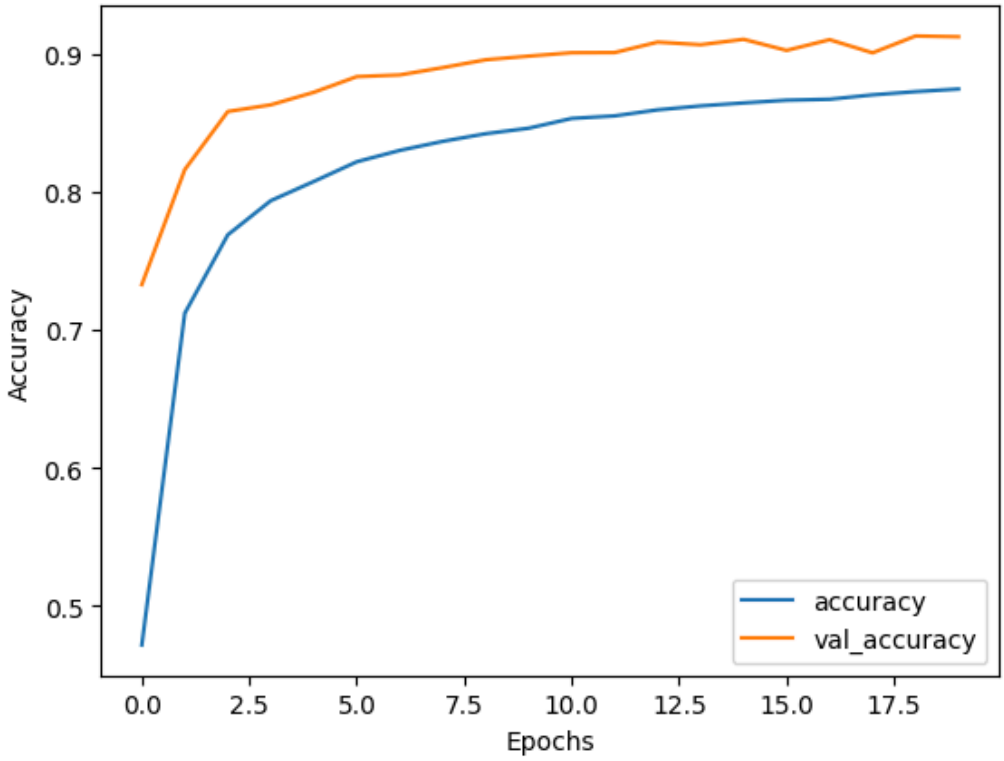
Model: "sequential"

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)

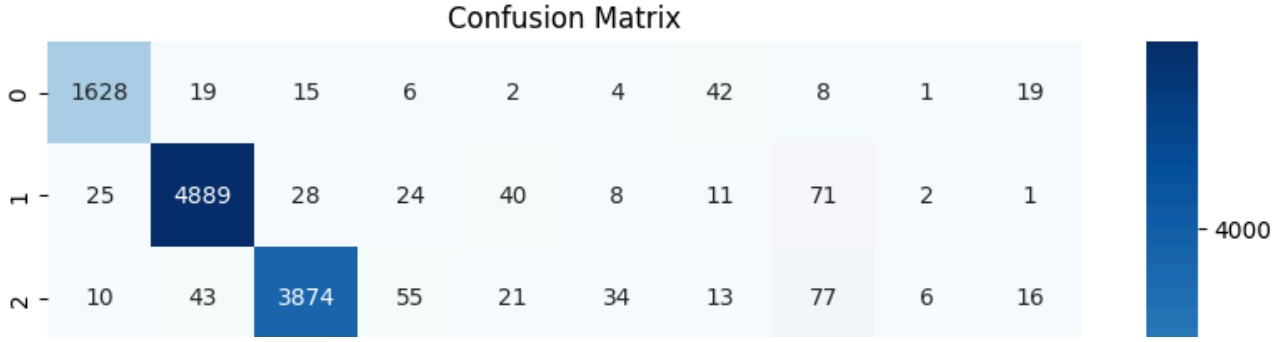
```
Epoch 1/20
/usr/local/lib/python3.10/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:1
  self._warn_if_super_not_called()
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 149s 127ms/step - accuracy: 0.3232 - loss: 1.9098 - val_accuracy
Epoch 2/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 196s 123ms/step - accuracy: 0.6922 - loss: 0.9522 - val_accuracy
Epoch 3/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 141s 122ms/step - accuracy: 0.7597 - loss: 0.7618 - val_accuracy
Epoch 4/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 155s 135ms/step - accuracy: 0.7874 - loss: 0.6715 - val_accuracy
Epoch 5/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 148s 129ms/step - accuracy: 0.8068 - loss: 0.6110 - val_accuracy
Epoch 6/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 198s 126ms/step - accuracy: 0.8194 - loss: 0.5766 - val_accuracy
Epoch 7/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 149s 130ms/step - accuracy: 0.8302 - loss: 0.5414 - val_accuracy
Epoch 8/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 200s 128ms/step - accuracy: 0.8359 - loss: 0.5314 - val_accuracy
Epoch 9/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 197s 124ms/step - accuracy: 0.8396 - loss: 0.5099 - val_accuracy
Epoch 10/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 141s 123ms/step - accuracy: 0.8465 - loss: 0.4894 - val_accuracy
Epoch 11/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 144s 126ms/step - accuracy: 0.8526 - loss: 0.4773 - val_accuracy
Epoch 12/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 198s 122ms/step - accuracy: 0.8531 - loss: 0.4713 - val_accuracy
Epoch 13/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 142s 124ms/step - accuracy: 0.8568 - loss: 0.4594 - val_accuracy
Epoch 14/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 142s 124ms/step - accuracy: 0.8640 - loss: 0.4441 - val_accuracy
Epoch 15/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 142s 124ms/step - accuracy: 0.8642 - loss: 0.4428 - val_accuracy
Epoch 16/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 147s 128ms/step - accuracy: 0.8686 - loss: 0.4257 - val_accuracy
Epoch 17/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 142s 124ms/step - accuracy: 0.8675 - loss: 0.4214 - val_accuracy
```

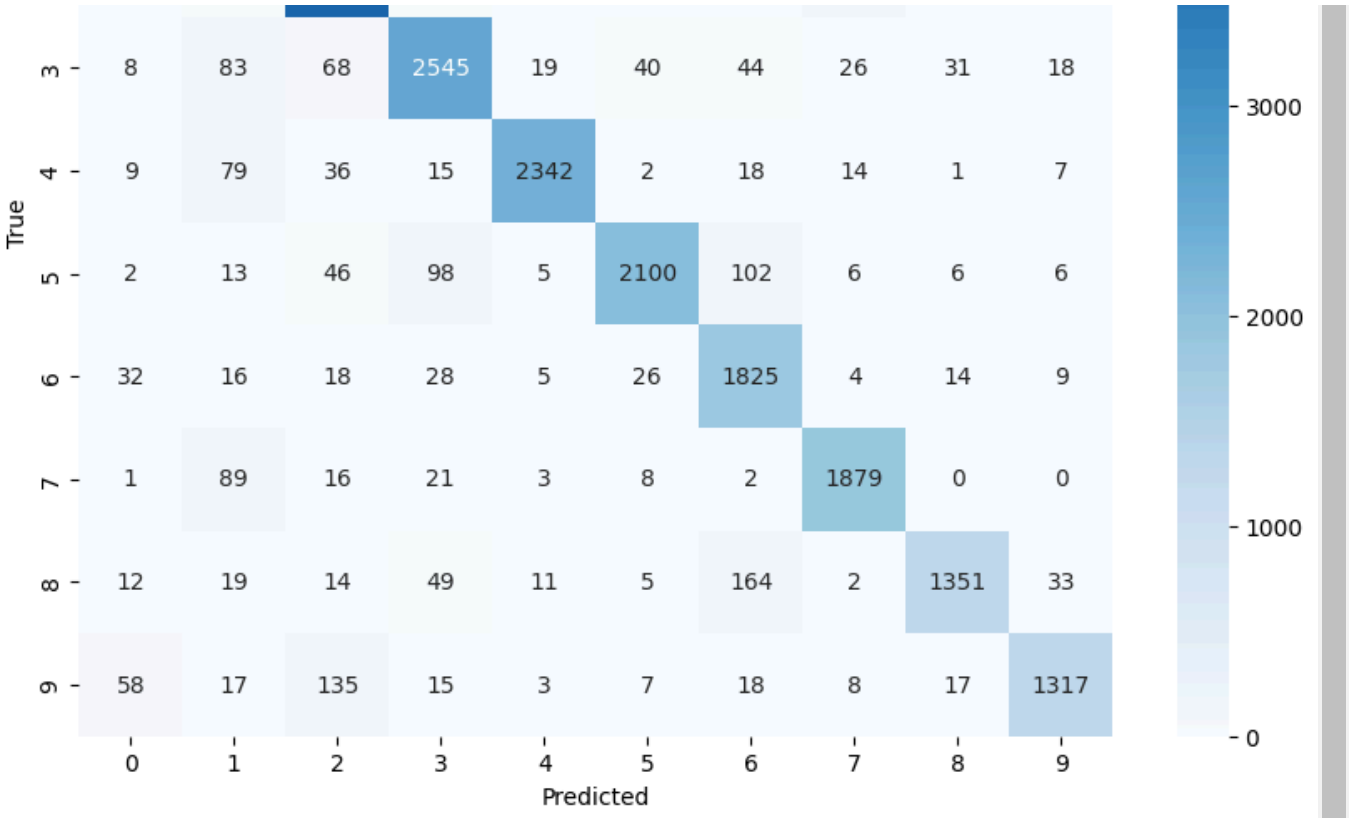
```
Epoch 18/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 142s 123ms/step - accuracy: 0.8712 - loss: 0.4167 - val_accuracy
Epoch 19/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 146s 127ms/step - accuracy: 0.8720 - loss: 0.4187 - val_accuracy
Epoch 20/20
1145/1145 ━━━━━━━━━━━━━━━━━━━━ 198s 124ms/step - accuracy: 0.8719 - loss: 0.4151 - val_accuracy
```

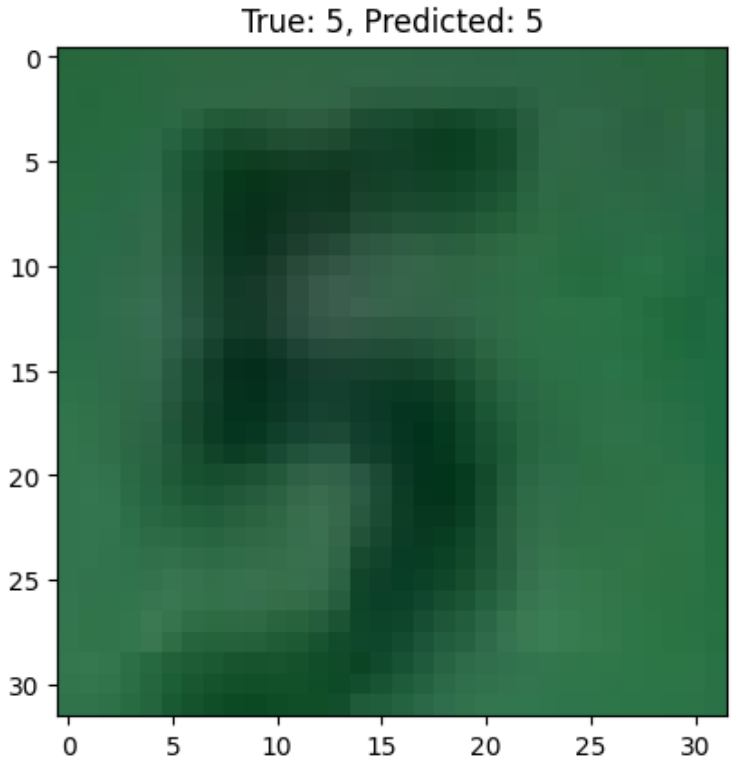


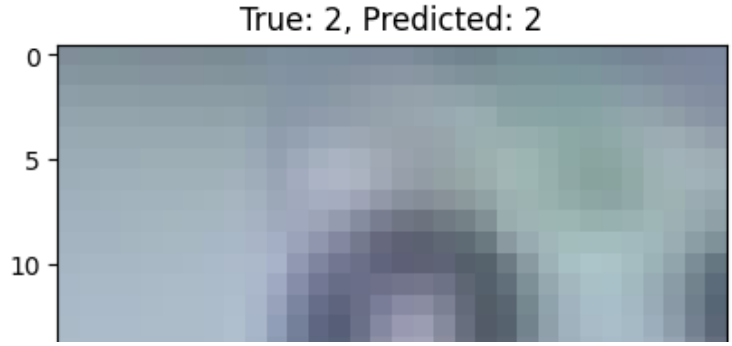

```
814/814 ━━━━━━━━━━━━━━━━━━━━ 11s 14ms/step - accuracy: 0.9107 - loss: 0.3107
Test accuracy: 0.9123386740684509
814/814 ━━━━━━━━━━━━━━━━━━━━ 11s 13ms/step
```

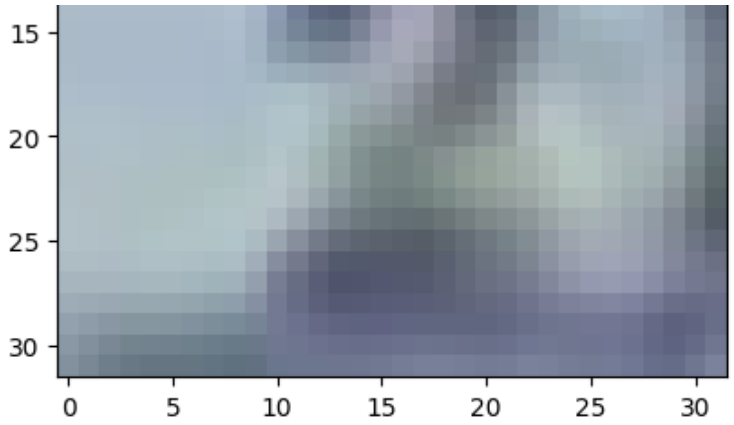
	precision	recall	f1-score	support
0	0.91	0.93	0.92	1744
1	0.93	0.96	0.94	5099
2	0.91	0.93	0.92	4149
3	0.89	0.88	0.89	2882
4	0.96	0.93	0.94	2523
5	0.94	0.88	0.91	2384
6	0.82	0.92	0.87	1977
7	0.90	0.93	0.91	2019
8	0.95	0.81	0.87	1660
9	0.92	0.83	0.87	1595
accuracy			0.91	26032
macro avg	0.91	0.90	0.91	26032
weighted avg	0.91	0.91	0.91	26032



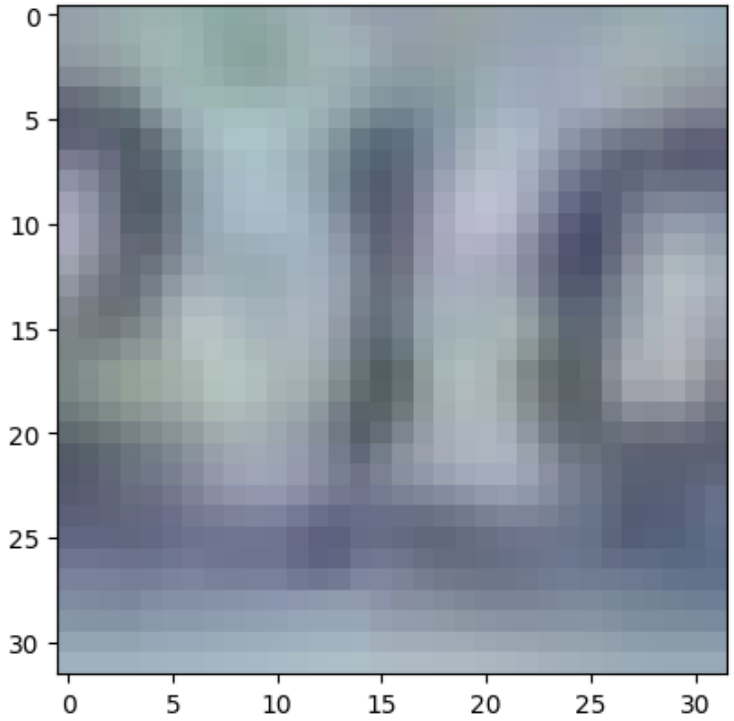

True
Predicted
3 8 83 68 2545 19 40 44 26 31 18
4 9 79 36 15 2342 2 18 14 1 7
5 2 13 46 98 5 2100 102 6 6 6
6 32 16 18 28 5 26 1825 4 14 9
7 1 89 16 21 3 8 2 1879 0 0
8 12 19 14 49 11 5 164 2 1351 33
9 58 17 135 15 3 7 18 8 17 1317
0 1 2 3 4 5 6 7 8 9
3000
2000
1000
0


True: 5, Predicted: 5
0
5
10
15
20
25
30

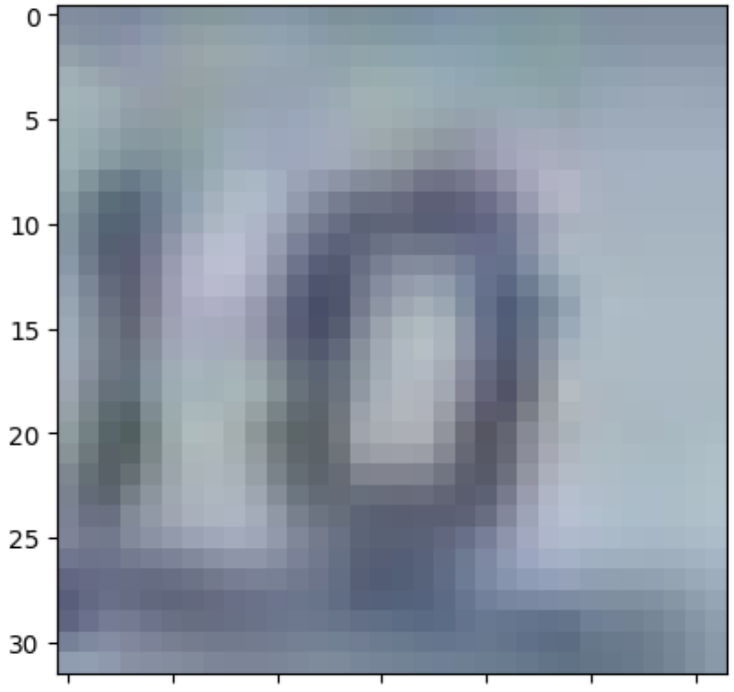

True: 2, Predicted: 2
0
5
10

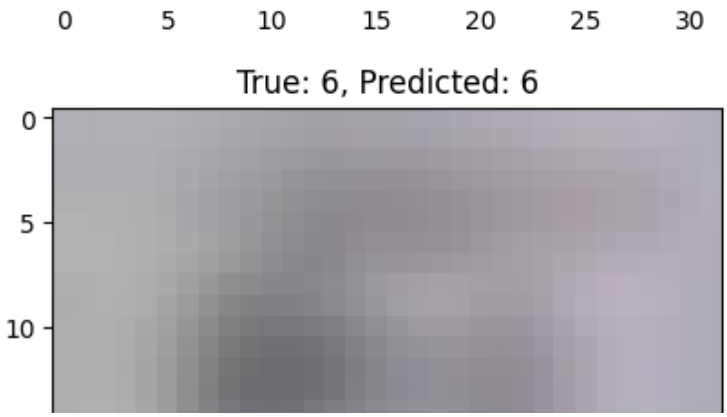


True: 1, Predicted: 1



True: 0, Predicted: 0




0
5
10
15
20
25
30
True: 6, Predicted: 6
0
5
10