Convolutional Neural Networks (CNNs)

This notebook demonstrates how to build a Convolutional Neural Network (CNN) to classify images from the Cifar10 dataset. We then evaluate the model on test data and report the performance metrics (accuracy, precision, recall).

```
#load necessary libraries
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import itertools

import keras
from keras.datasets import cifar10
from tensorflow.keras import layers
from keras.models import Sequential
from sklearn.metrics import confusion_matrix, classification_report

%matplotlib inline
```

```
# load and preprocess data
(train_images, train_labels),(test_images,test_labels) = cifar10.lo

# normalize pixel values
train_images, test_images = train_images/255.0, test_images/255.0

# reshape image data for CNN input
train_images = train_images.reshape(train_images.shape[0], 32, 32,
test_images = test_images.reshape(test_images.shape[0], 32, 32, 3)

# one-hot encode labels
train_labels = keras.utils.to_categorical(train_labels, 10)
test_labels = keras.utils.to_categorical(test_labels, 10)

# verify shapes
train_images.shape, test_images.shape
```

```
((50000, 32, 32, 3), (10000, 32, 32, 3))
```

```
# Build the CNN model
model = Sequential([
    # First convolutional layer
    layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 3
    layers.MaxPooling2D((2, 2)),

    # Second convolutional layer
    layers.Conv2D(64, (3, 3), activation='relu'),
    layers.MaxPooling2D((2, 2)),

    # Third convolutional layer
    layers.Conv2D(128, (3, 3), activation='relu'),

    # Flatten the 3D output to 1D
    layers.Flatten(),

    #Dense (fully connected) layer
    layers.Dense(128, activation='relu'),

    # Output layer
    layers.Dense(10, activation='softmax')
])
```

```
/usr/local/lib/python3.12/dist-packages/keras/src/layers/convolution
  super().__init__(activity_regularizer=activity_regularizer, **kwar
```

```
# Compile the model
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Train the model
history = model.fit(train_images, train_labels, epochs=10, batch_siz
                    validation_data=(test_images, test_labels))
```

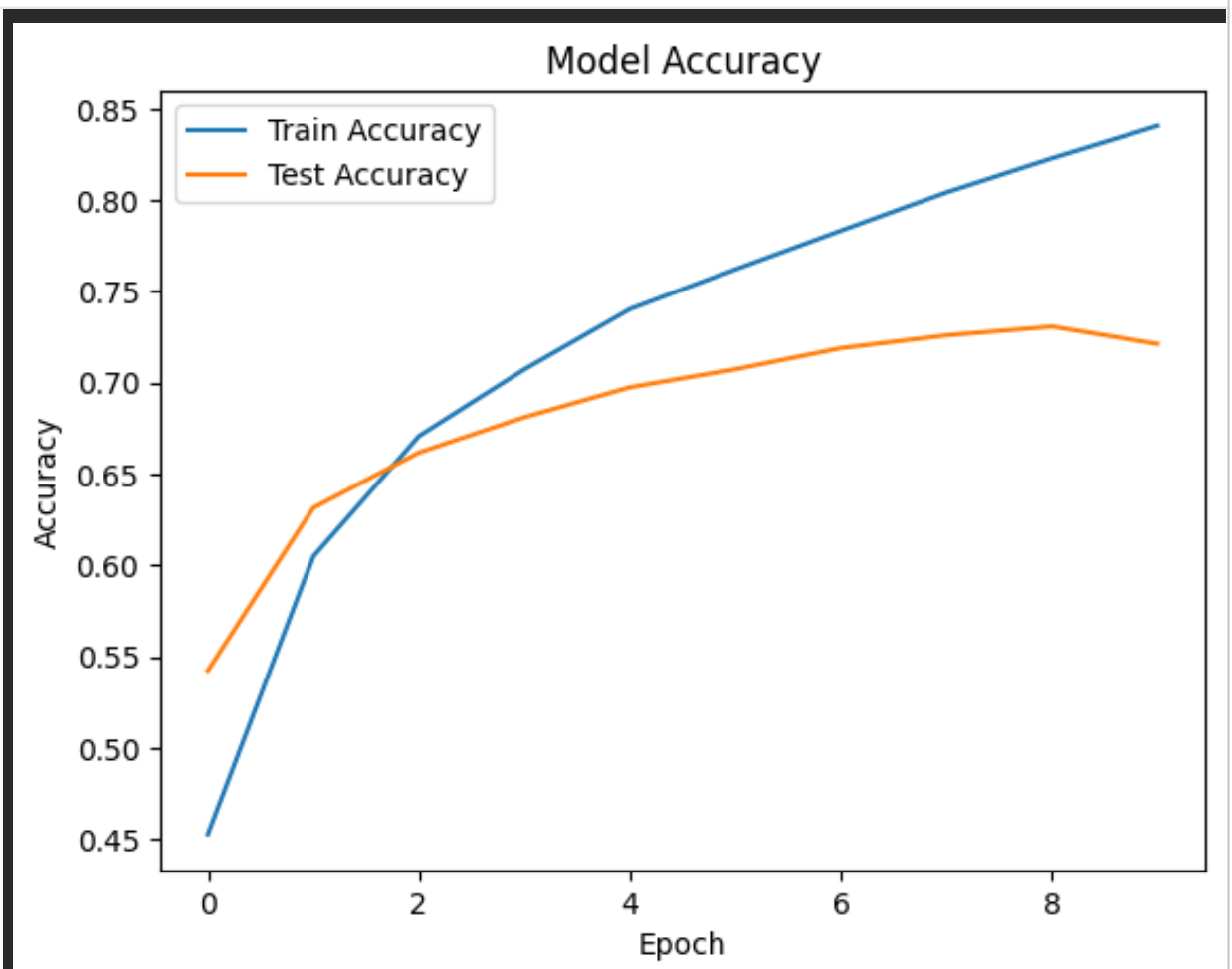
```
Epoch 1/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 72s 89ms/step - accuracy: 0.3639 - loss
Epoch 2/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 68s 87ms/step - accuracy: 0.5844 - loss
Epoch 3/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 85s 90ms/step - accuracy: 0.6675 - loss
Epoch 4/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 67s 86ms/step - accuracy: 0.7030 - loss
Epoch 5/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 84s 88ms/step - accuracy: 0.7438 - loss
Epoch 6/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 69s 89ms/step - accuracy: 0.7614 - loss
Epoch 7/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 69s 88ms/step - accuracy: 0.7884 - loss
Epoch 8/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 68s 86ms/step - accuracy: 0.8092 - loss
Epoch 9/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 83s 88ms/step - accuracy: 0.8310 - loss
Epoch 10/10
782/782 ━━━━━━━━━━━━━━━━━━━━ 67s 86ms/step - accuracy: 0.8519 - loss
```

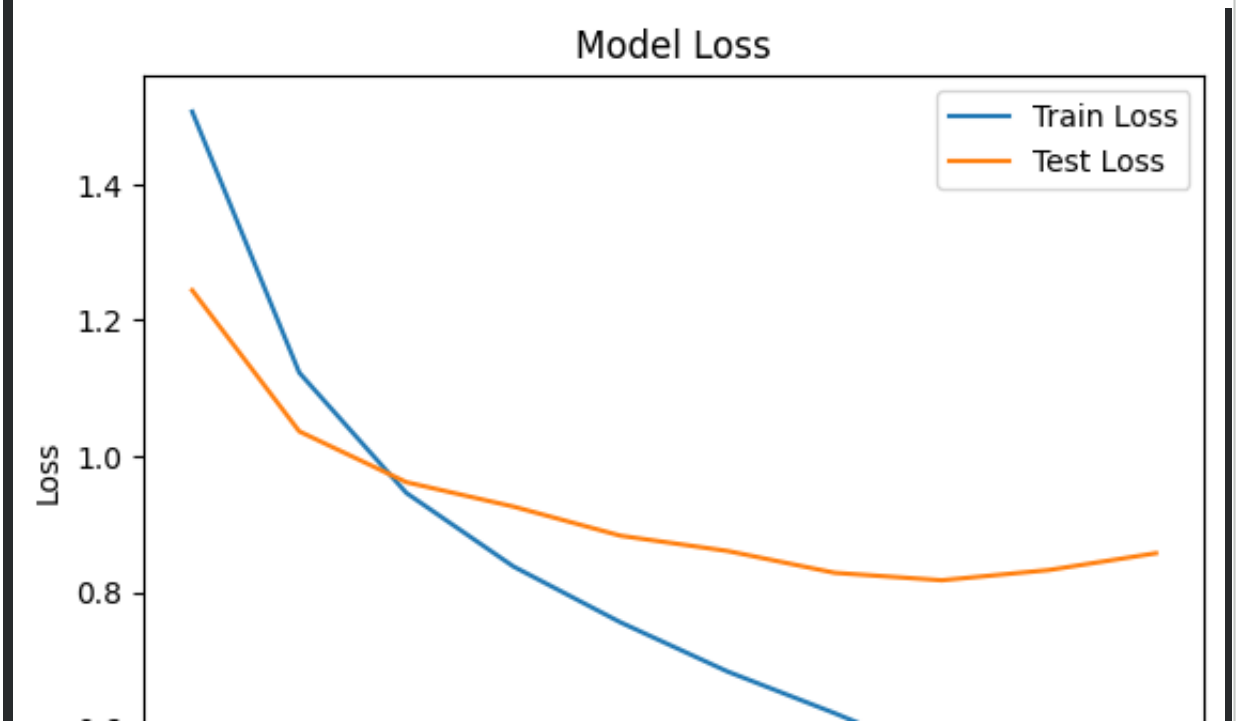
```
#Evaluate the model
test_loss, test_acc = model.evaluate(test_images, test_labels, verbo
print('\nTest accuracy:', test_acc)
```

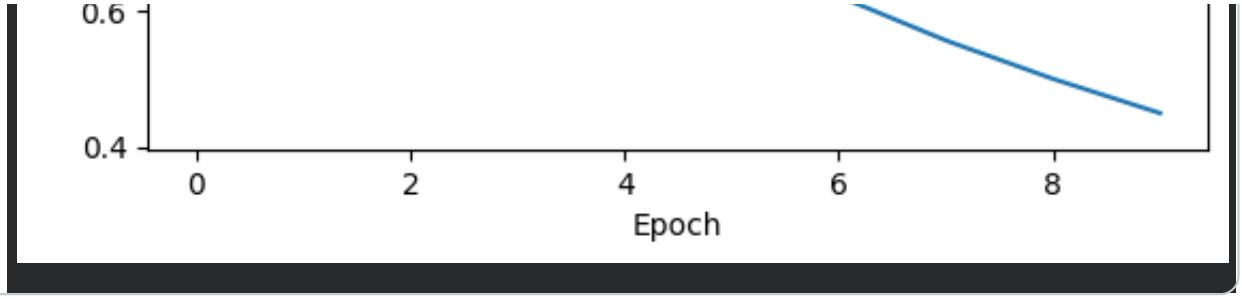
```
313/313 - 4s - 12ms/step - accuracy: 0.7213 - loss: 0.8571

Test accuracy: 0.7213000059127808
```

```
# plot training & validation accuracy values
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Test Accuracy')
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.show()

# plot training & validation loss values
plt.plot(history.history['loss'], label='Train Loss')
```

```
plt.plot(history.history['val_loss'], label='Test Loss')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend()
plt.show()
```

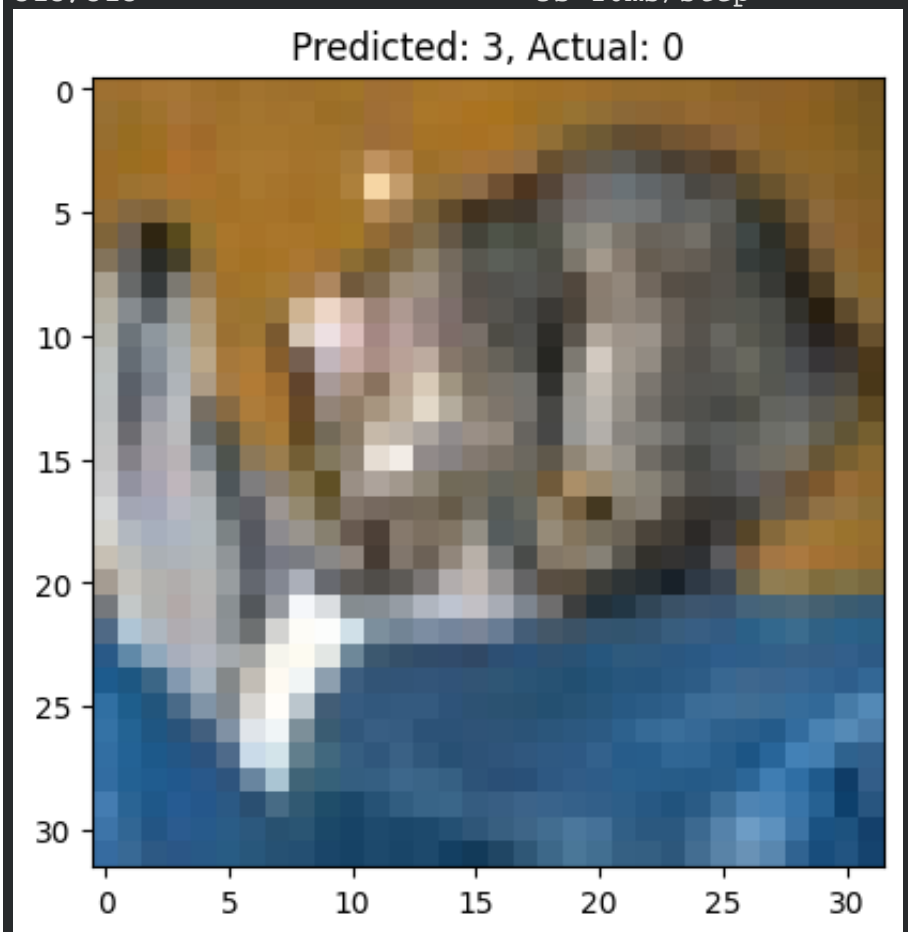


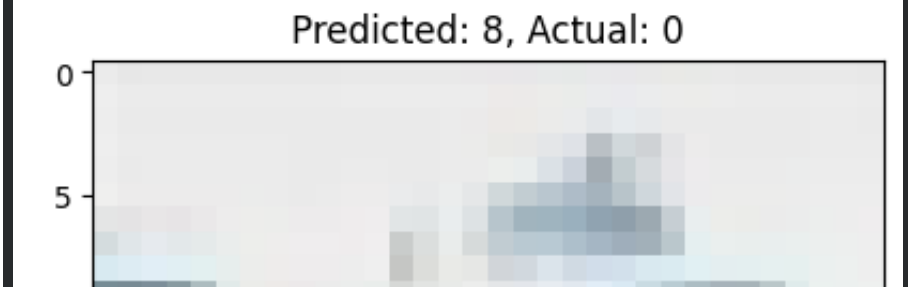


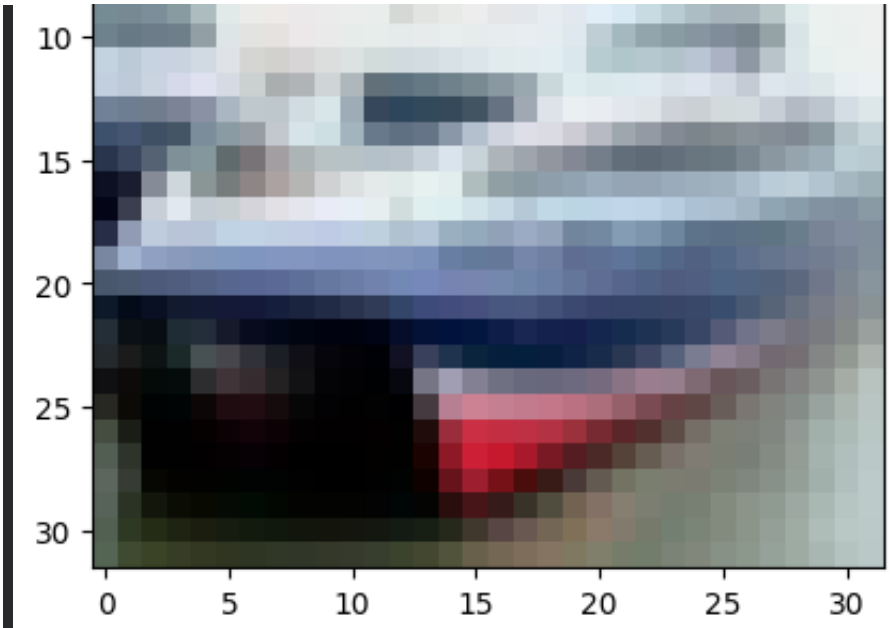



```
# make predictions
predictions = model.predict(test_images)

#display the first 5 predictions
for i in range(5):
    plt.imshow(test_images[i].reshape(32, 32, 3), cmap='gray')
    plt.title(f'Predicted: {predictions[i].argmax()}, Actual: {y_tes
    plt.show()
```

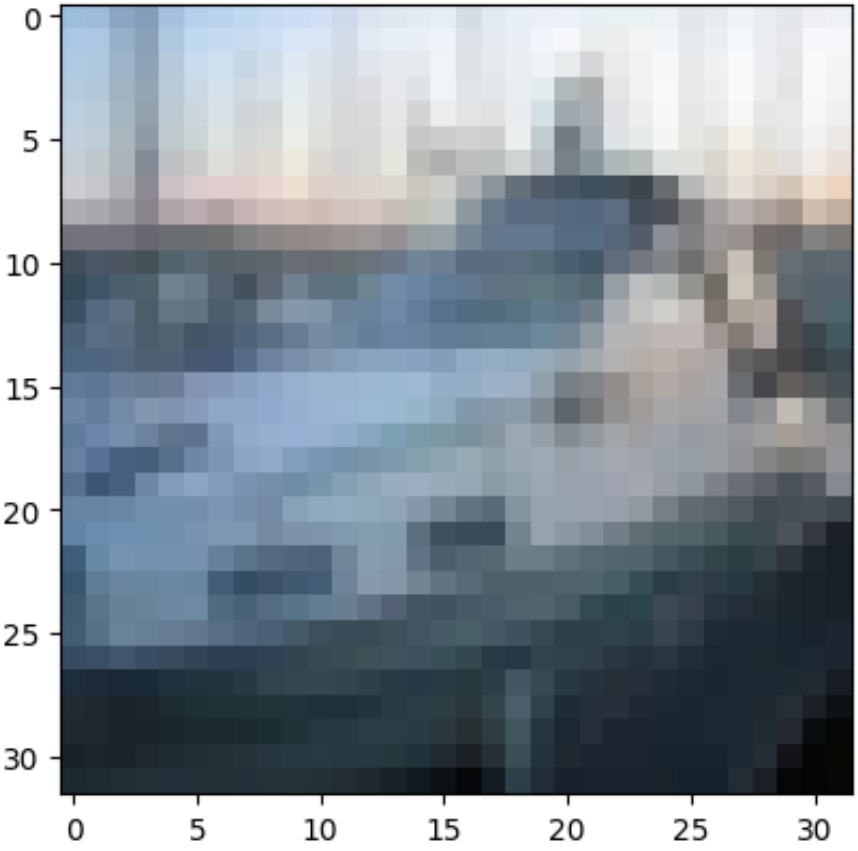
```
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 16ms/step
```



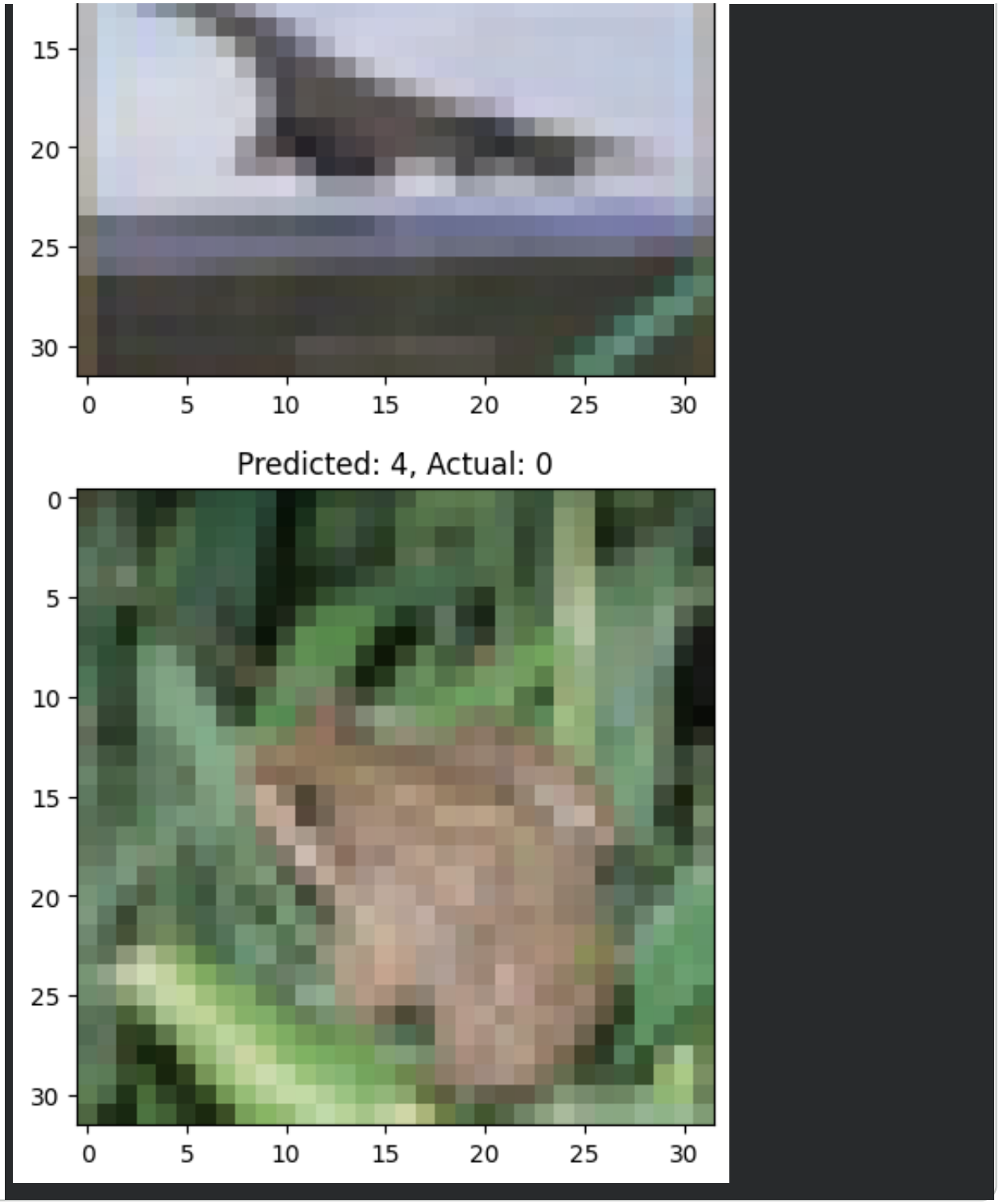




Predicted: 8, Actual: 0



Predicted: 0, Actual: 0


15
20
25
30
0
5
10
15
20
25
30
Predicted: 4, Actual: 0
0
5
10
15
20
25
30
0
5
10
15
20
25
30