cifar10

December 9, 2023

[ ]:
```
import tensorflow as tf
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Conv2D, MaxPooling2D, Flatten,␣
↪Dropout
from tensorflow.keras.utils import plot_model
import numpy as np
from sklearn.model_selection import train_test_split
```

[ ]:
```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import regularizers
```

[ ]:
```
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
```

[ ]:
```
x_train, x_val, y_train, y_val = train_test_split(x_train, y_train, test_size=0.
↪2, random_state=42)
```

[ ]:
```
# Display shapes
print("Shape of x_train:", x_train.shape)
print("Shape of y_train:", y_train.shape)
```

```
Shape of x_train: (40000, 32, 32, 3)
Shape of y_train: (40000, 1)
```

[ ]:
```
# Get unique values in y_train using numpy
unique_labels = np.unique(y_train)
print("Unique labels:", unique_labels)
```

```
Unique labels: [0 1 2 3 4 5 6 7 8 9]
```

[ ]:
```
# Plotting 4 images in a row
plt.figure(figsize=(5, 5))
for i in range(4):
    plt.subplot(1, 4, i+1)
    plt.imshow(x_train[i])  # Display the image
    plt.title(f"Label: {y_train[i][0]}")  # Show the label
    plt.axis('off')  # Hide the axes
plt.tight_layout()
plt.show()
```

[ ]:
```
x_train = x_train.astype('float32') / 255.0
x_test = x_test.astype('float32') / 255.0
x_val = x_val.astype('float32') / 255.0
```

[ ]:
```
plt.figure(figsize=(5, 5))
for i in range(4):
    plt.subplot(1, 4, i+1)
    plt.imshow(x_train[i])  # Display the image
    plt.title(f"Label: {y_train[i][0]}")  # Show the label
    plt.axis('off')  # Hide the axes
plt.tight_layout()
plt.show()
```

[ ]:
```
model = Sequential([
    Conv2D(64, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    MaxPooling2D(),

    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D(),

    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D(),

    Flatten(),
```

```
    Dense(512, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    Dropout(0.5),
    Dense(256, activation='relu', kernel_regularizer=regularizers.l2(0.001)),
    Dropout(0.3),
    Dense(10, activation='softmax')
])
```

[ ]:
```
model.summary()
```

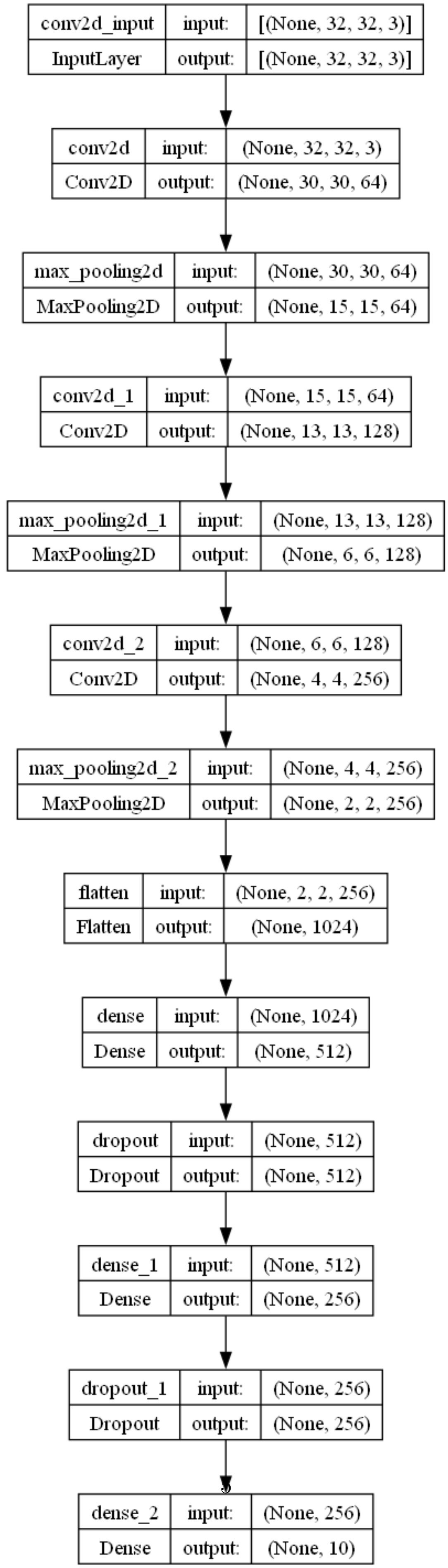
```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         295168

 max_pooling2d_2 (MaxPooling  (None, 2, 2, 256)        0
 2D)

 flatten (Flatten)           (None, 1024)              0
```


```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 64)        1792

 max_pooling2d (MaxPooling2D  (None, 15, 15, 64)       0
 )

 conv2d_1 (Conv2D)           (None, 13, 13, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 6, 6, 128)        0
 2D)

 conv2d_2 (Conv2D)           (None, 4, 4, 256)         295168

 max_pooling2d_2 (MaxPooling  (None, 2, 2, 256)        0
 2D)
```

```
 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 512)               524800

 dropout (Dropout)           (None, 512)               0

 dense_1 (Dense)             (None, 256)               131328

dropout_1 (Dropout)          (None, 256)               0

 dense_2 (Dense)             (None, 10)                2570

=================================================================
Total params: 1,029,514
Trainable params: 1,029,514
Non-trainable params: 0

_________________________________________________________________
```

[ ]:
```
plot_model(model, show_shapes=True, show_layer_names=True, to_file='model.png')
```

[ ]:


conv2d_input
InputLayer
input:
output:
[(None, 32, 32, 3)]
[(None, 32, 32, 3)]
conv2d
Conv2D
input:
output:
(None, 32, 32, 3)
(None, 30, 30, 64)
max_pooling2d
MaxPooling2D
input:
output:
(None, 30, 30, 64)
(None, 15, 15, 64)
conv2d_1
Conv2D
input:
output:
(None, 15, 15, 64)
(None, 13, 13, 128)
max_pooling2d_1
MaxPooling2D
input:
output:
(None, 13, 13, 128)
(None, 6, 6, 128)
conv2d_2
Conv2D
input:
output:
(None, 6, 6, 128)
(None, 4, 4, 256)
max_pooling2d_2
MaxPooling2D
input:
output:
(None, 4, 4, 256)
(None, 2, 2, 256)
flatten
Flatten
input:
output:
(None, 2, 2, 256)
(None, 1024)
dense
Dense
input:
output:
(None, 1024)
(None, 512)
dropout
Dropout
input:
output:
(None, 512)
(None, 512)
dense_1
Dense
input:
output:
(None, 512)
(None, 256)
dropout_1
Dropout
input:
output:
(None, 256)
(None, 256)
dense_2
Dense
input:
output:
(None, 256)
(None, 10)

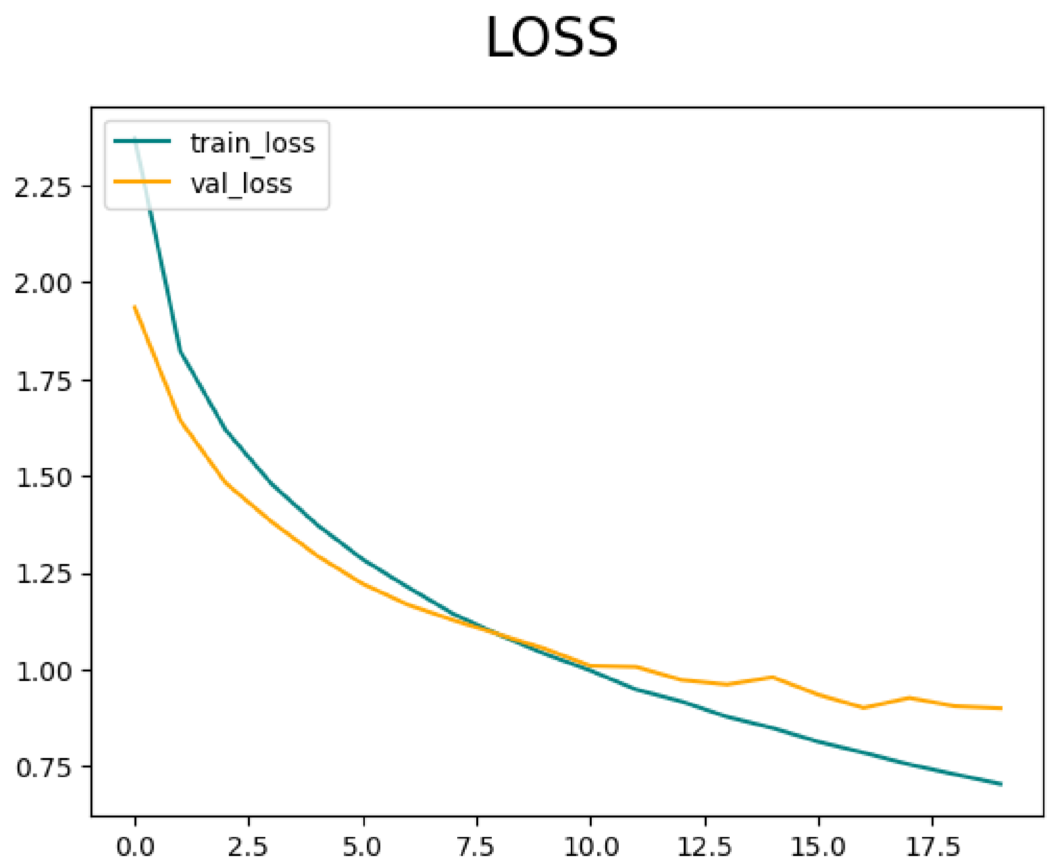
[ ]:
```
optimizer = Adam(learning_rate=0.0001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',␣
↪metrics=['accuracy'])
```

[ ]:
```
history = model.fit(x_train, y_train, epochs=20, validation_data=(x_val, y_val))
```

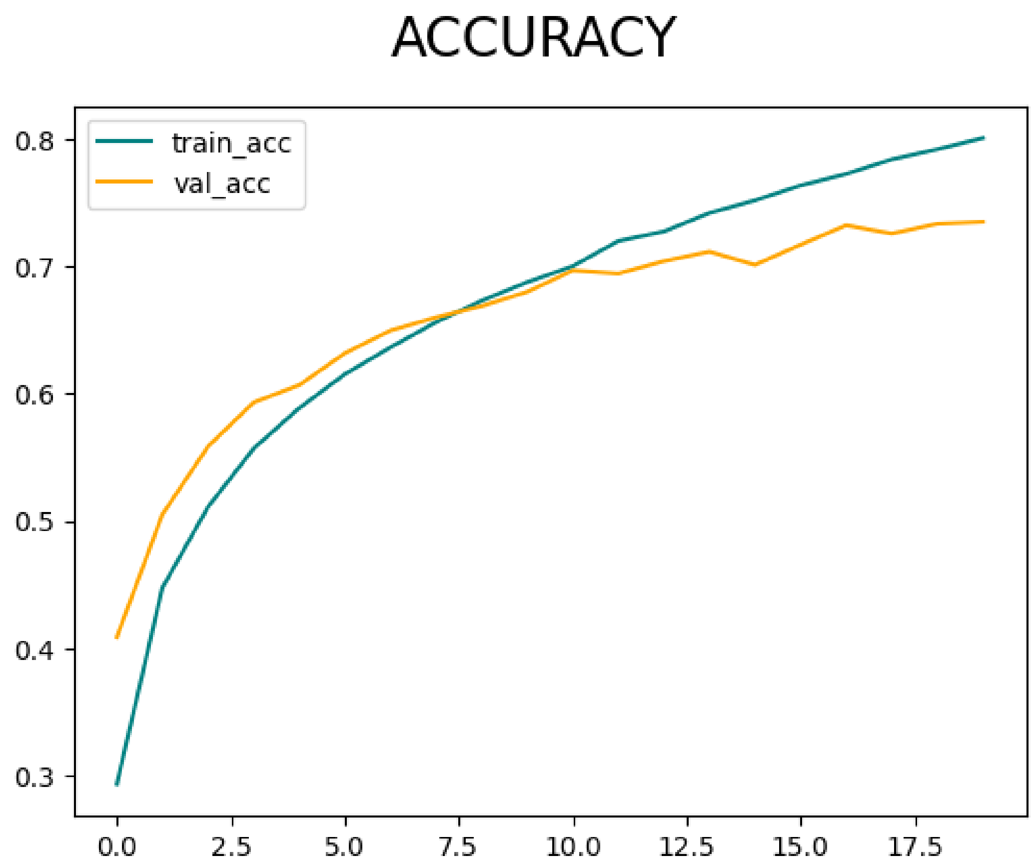
```
Epoch 1/20
1250/1250 [==============================] - 13s 8ms/step - loss: 2.3722 -
accuracy: 0.2933 - val_loss: 1.9353 - val_accuracy: 0.4088
Epoch 2/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.8221 -
accuracy: 0.4476 - val_loss: 1.6432 - val_accuracy: 0.5052
Epoch 3/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.6176 -
accuracy: 0.5110 - val_loss: 1.4810 - val_accuracy: 0.5585
Epoch 4/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.4793 -
accuracy: 0.5569 - val_loss: 1.3817 - val_accuracy: 0.5931
Epoch 5/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.3737 -
accuracy: 0.5886 - val_loss: 1.2938 - val_accuracy: 0.6067
Epoch 6/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.2844 -
accuracy: 0.6152 - val_loss: 1.2211 - val_accuracy: 0.6317
Epoch 7/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.2118 -
accuracy: 0.6363 - val_loss: 1.1666 - val_accuracy: 0.6496
Epoch 8/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.1426 -
accuracy: 0.6561 - val_loss: 1.1268 - val_accuracy: 0.6599
Epoch 9/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.0892 -
accuracy: 0.6730 - val_loss: 1.0901 - val_accuracy: 0.6688
Epoch 10/20
1250/1250 [==============================] - 9s 7ms/step - loss: 1.0399 -
accuracy: 0.6876 - val_loss: 1.0533 - val_accuracy: 0.6799
Epoch 11/20
1250/1250 [==============================] - 9s 7ms/step - loss: 0.9968 -
accuracy: 0.7001 - val_loss: 1.0090 - val_accuracy: 0.6968
Epoch 12/20
1250/1250 [==============================] - 9s 8ms/step - loss: 0.9480 -
accuracy: 0.7199 - val_loss: 1.0062 - val_accuracy: 0.6944
Epoch 13/20
1250/1250 [==============================] - 9s 7ms/step - loss: 0.9170 -
accuracy: 0.7274 - val_loss: 0.9727 - val_accuracy: 0.7041
```

```
Epoch 14/20
1250/1250 [==============================] - 9s 8ms/step - loss: 0.8773 -
accuracy: 0.7420 - val_loss: 0.9608 - val_accuracy: 0.7115
Epoch 15/20
1250/1250 [==============================] - 9s 7ms/step - loss: 0.8483 -
accuracy: 0.7519 - val_loss: 0.9797 - val_accuracy: 0.7012
Epoch 16/20
1250/1250 [==============================] - 9s 8ms/step - loss: 0.8131 -
accuracy: 0.7636 - val_loss: 0.9346 - val_accuracy: 0.7170
Epoch 17/20
1250/1250 [==============================] - 9s 7ms/step - loss: 0.7845 -
accuracy: 0.7726 - val_loss: 0.9005 - val_accuracy: 0.7323
Epoch 18/20
1250/1250 [==============================] - 9s 7ms/step - loss: 0.7543 -
accuracy: 0.7841 - val_loss: 0.9263 - val_accuracy: 0.7257
Epoch 19/20
1250/1250 [==============================] - 9s 7ms/step - loss: 0.7288 -
accuracy: 0.7921 - val_loss: 0.9048 - val_accuracy: 0.7335
Epoch 20/20
1250/1250 [==============================] - 10s 8ms/step - loss: 0.7040 -
accuracy: 0.8008 - val_loss: 0.8994 - val_accuracy: 0.7350
```

[ ]:
```
fig = plt.figure()
plt.plot(history.history["loss"], color='teal', label='train_loss')
plt.plot(history.history["val_loss"], color='orange', label='val_loss')
plt.suptitle('LOSS',fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



[ ]:
```
fig = plt.figure()
plt.plot(history.history['accuracy'],color='teal', label='train_acc')
plt.plot(history.history['val_accuracy'],color='orange', label='val_acc')
plt.suptitle('ACCURACY',fontsize=20)
plt.legend(loc="upper left")
plt.show()
```



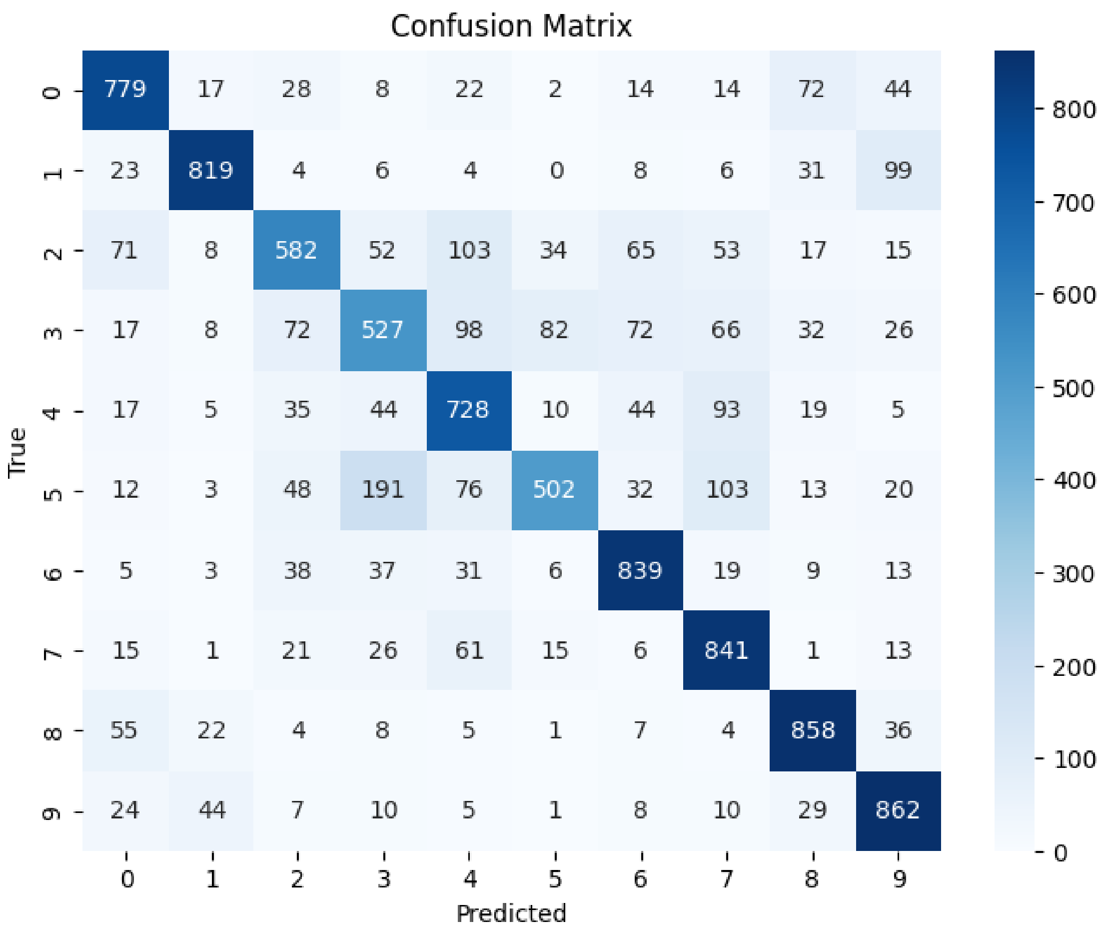
[ ]:
```
# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(x_test, y_test)

print(f"Test Accuracy: {test_accuracy*100:.2f}%")
print(f"Test Loss: {test_loss}")
```

```
313/313 [==============================] - 1s 3ms/step - loss: 0.9063 -
accuracy: 0.7337
Test Accuracy: 73.37%
Test Loss: 0.9063293933868408
```

[ ]:
```
import seaborn as sns
from sklearn.metrics import confusion_matrix
import numpy as np

# Assuming you have predictions for x_test stored in y_pred
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)  # Get the predicted classes

# Calculate the confusion matrix
```

```
conf_matrix = confusion_matrix(y_test, y_pred_classes)

# Define class labels (if available)
class_labels = unique_labels

# Create a heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, cmap='Blues', fmt='d',␣
↪xticklabels=class_labels, yticklabels=class_labels)
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

```
313/313 [==============================] - 1s 2ms/step
```



True \ Predicted	0	1	2	3	4	5	6	7	8	9
0	779	17	28	8	22	2	14	14	72	44
1	23	819	4	6	4	0	8	6	31	99
2	71	8	582	52	103	34	65	53	17	15
3	17	8	72	527	98	82	72	66	32	26
4	17	5	35	44	728	10	44	93	19	5
5	12	3	48	191	76	502	32	103	13	20
6	5	3	38	37	31	6	839	19	9	13
7	15	1	21	26	61	15	6	841	1	13
8	55	22	4	8	5	1	7	4	858	36
9	24	44	7	10	5	1	8	10	29	862

[ ]:
```
# Select a single image from the test set (e.g., the first image)
import random
```

```
image_index = random.randint(1, 499)  # Change this index to select a different image
selected_image = x_test[image_index]
true_label = y_test[image_index]


# Make a prediction on the selected image
prediction = model.predict(np.expand_dims(selected_image, axis=0))
predicted_label = np.argmax(prediction)

# Display the selected image and prediction
plt.figure(figsize=(5, 5))
plt.imshow(selected_image)
plt.title(f"True Label: {true_label}, Predicted Label: {predicted_label}")
plt.axis('off')
plt.show()
```

```
1/1 [==============================] - 0s 18ms/step
```

True Label: [6], Predicted Label: 6