```
import numpy as np
import os
import pandas as pd
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Drive already mounted at /content/gdrive; to attempt to forcibly remount, call drive.mount("/content/gdrive", force_remount=True).
```

```
os.listdir("/content/gdrive/MyDrive/archive")
```

```
['skin']
```

```
os.listdir("/content/gdrive/MyDrive/archive/skin/train_set")
```

```
['benign', 'malignant']
```

```
os.listdir("/content/gdrive/MyDrive/archive/skin/test_set")
```

```
['benign', 'malignant']
```

```
import tensorflow as tf
import os
from keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Conv2D
from tensorflow.keras.layers import Dropout
from tensorflow.keras import Model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.optimizers import Adam
from tensorflow.keras import layers
```

```
from tensorflow.keras.applications.resnet50 import ResNet50
```

```
train_dir="/content/gdrive/MyDrive/archive/skin/train_set"
```

```
test__dir="/content/gdrive/MyDrive/archive/skin/test_set"
```

```
label=["'malignant","benign"]
```

```
print("class")
for i in range(len(label)):
  print(i,end=" ")
  print(label[i])
```

```
class
0 'malignant
1 benign
```

```
print("number of classes:",len(label))
```

```
number of classes: 2
```

```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import confusion_matrix
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Model
```

```
# Define data directories
train_dir ="/content/gdrive/MyDrive/archive/skin/train_set"
test_dir = "/content/gdrive/MyDrive/archive/skin/test_set"
```

```
# Define data generators for training and testing data
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True,
```

```
                                    validation_split=0.2,)
test_datagen = ImageDataGenerator(rescale=1./255)

train_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(224, 224),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    subset='training')

val_generator = train_datagen.flow_from_directory(train_dir,
                                                    target_size=(224, 224),
                                                    batch_size=32,
                                                    class_mode='categorical',
                                                    subset='validation')
test_generator = test_datagen.flow_from_directory(test_dir,
                                                  target_size=(224, 224),
                                                  batch_size=800,
                                                  class_mode='categorical')
```

```
Found 2110 images belonging to 2 classes.
Found 527 images belonging to 2 classes.
Found 662 images belonging to 2 classes.
```

```
# Load the ResNet50 model with pre-trained ImageNet weights
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
```

```
# Add custom layers on top of the pre-trained model
x = base_model.output
x = Flatten()(x)
x = Dense(512, activation='relu')(x)
x = Dropout(0.5)(x)
#output layer fully connected dance layer with two neuron
predictions = Dense(2, activation='softmax')(x)
```

```
# Combine the base ResNet50 model with the custom layers
model = Model(inputs=base_model.input, outputs=predictions)
#model.summary()
```

```
# Freeze all layers in the base ResNet50 model
for layer in base_model.layers[5:]:
    layer.trainable = False
# Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

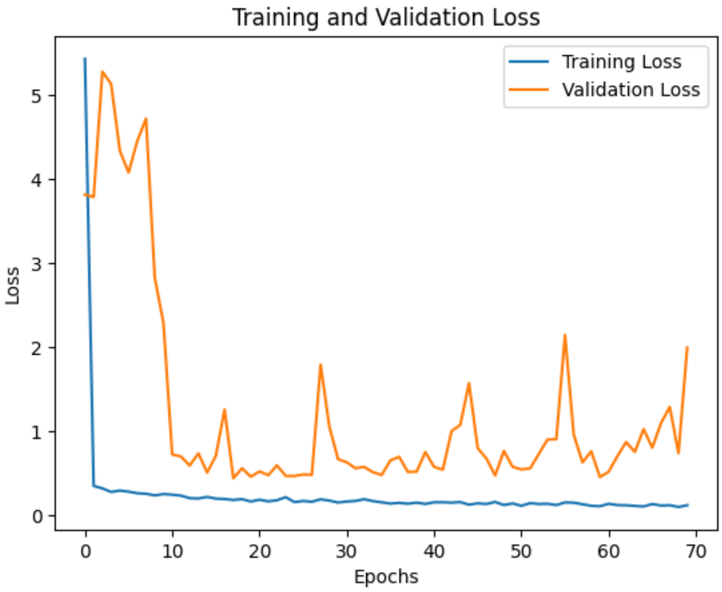
```
 ization)

 conv5_block3_add (Add)         (None, 7, 7, 2048)   0           ['conv5_block2_out[0][0]',
                                                                  'conv5_block3_3_bn[0][0]']

 conv5_block3_out (Activation)  (None, 7, 7, 2048)   0           ['conv5_block3_add[0][0]']

 flatten_1 (Flatten)            (None, 100352)       0           ['conv5_block3_out[0][0]']

 dense_2 (Dense)                (None, 512)          51380736    ['flatten_1[0][0]']

 dropout_1 (Dropout)            (None, 512)          0           ['dense_2[0][0]']

 dense_3 (Dense)                (None, 2)            1026        ['dropout_1[0][0]']

==================================================================================================
Total params: 74,969,474
Trainable params: 51,391,362
Non-trainable params: 23,578,112
__________________________________________________________________________________________________
```

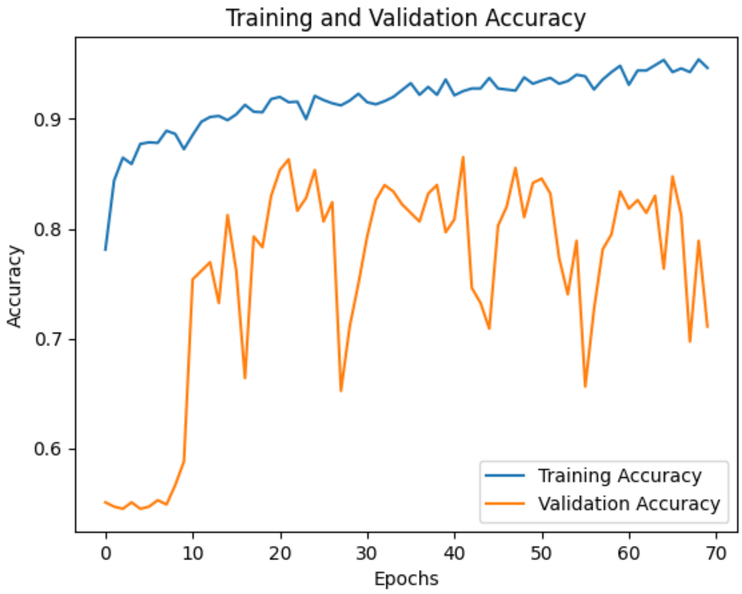
```
# Train the model on the training data
history = model.fit_generator(train_generator,
                              steps_per_epoch=train_generator.n // train_generator.batch_size,
                              epochs=70,
                              validation_data=val_generator,
                              validation_steps=val_generator.n // val_generator.batch_size)
```

```
65/65 [==============================] - 44s 677ms/step - loss: 0.1606 - accuracy: 0.9254 - val_loss: 0.5479 - val_accuracy: 0.8
Epoch 43/70
65/65 [==============================] - 45s 697ms/step - loss: 0.1553 - accuracy: 0.9278 - val_loss: 1.0050 - val_accuracy: 0.7
Epoch 44/70
65/65 [==============================] - 46s 714ms/step - loss: 0.1624 - accuracy: 0.9278 - val_loss: 1.0791 - val_accuracy: 0.7
Epoch 45/70
65/65 [==============================] - 45s 689ms/step - loss: 0.1313 - accuracy: 0.9374 - val_loss: 1.5758 - val_accuracy: 0.7
Epoch 46/70
65/65 [==============================] - 46s 709ms/step - loss: 0.1474 - accuracy: 0.9278 - val_loss: 0.8029 - val_accuracy: 0.8
Epoch 47/70
65/65 [==============================] - 46s 699ms/step - loss: 0.1393 - accuracy: 0.9269 - val_loss: 0.6788 - val_accuracy: 0.8
Epoch 48/70
65/65 [==============================] - 46s 705ms/step - loss: 0.1638 - accuracy: 0.9259 - val_loss: 0.4790 - val_accuracy: 0.8
Epoch 49/70
65/65 [==============================] - 45s 693ms/step - loss: 0.1271 - accuracy: 0.9379 - val_loss: 0.7701 - val_accuracy: 0.8
Epoch 50/70
65/65 [==============================] - 46s 709ms/step - loss: 0.1459 - accuracy: 0.9321 - val_loss: 0.5810 - val_accuracy: 0.8
Epoch 51/70
65/65 [==============================] - 44s 671ms/step - loss: 0.1186 - accuracy: 0.9350 - val_loss: 0.5508 - val_accuracy: 0.8
Epoch 52/70
65/65 [==============================] - 47s 718ms/step - loss: 0.1492 - accuracy: 0.9374 - val_loss: 0.5616 - val_accuracy: 0.8
Epoch 53/70
65/65 [==============================] - 44s 671ms/step - loss: 0.1400 - accuracy: 0.9321 - val_loss: 0.7350 - val_accuracy: 0.7
Epoch 54/70
65/65 [==============================] - 44s 675ms/step - loss: 0.1424 - accuracy: 0.9346 - val_loss: 0.9061 - val_accuracy: 0.7
Epoch 55/70
65/65 [==============================] - 46s 709ms/step - loss: 0.1264 - accuracy: 0.9404 - val_loss: 0.9091 - val_accuracy: 0.7
Epoch 56/70
65/65 [==============================] - 46s 705ms/step - loss: 0.1582 - accuracy: 0.9389 - val_loss: 2.1475 - val_accuracy: 0.6
Epoch 57/70
65/65 [==============================] - 44s 680ms/step - loss: 0.1551 - accuracy: 0.9269 - val_loss: 0.9618 - val_accuracy: 0.7
Epoch 58/70
65/65 [==============================] - 47s 722ms/step - loss: 0.1366 - accuracy: 0.9360 - val_loss: 0.6343 - val_accuracy: 0.7
Epoch 59/70
65/65 [==============================] - 45s 693ms/step - loss: 0.1182 - accuracy: 0.9427 - val_loss: 0.7658 - val_accuracy: 0.7
Epoch 60/70
65/65 [==============================] - 47s 729ms/step - loss: 0.1130 - accuracy: 0.9485 - val_loss: 0.4589 - val_accuracy: 0.8
Epoch 61/70
65/65 [==============================] - 47s 714ms/step - loss: 0.1421 - accuracy: 0.9312 - val_loss: 0.5229 - val_accuracy: 0.8
Epoch 62/70
65/65 [==============================] - 48s 731ms/step - loss: 0.1268 - accuracy: 0.9442 - val_loss: 0.7078 - val_accuracy: 0.8
Epoch 63/70
65/65 [==============================] - 46s 703ms/step - loss: 0.1244 - accuracy: 0.9442 - val_loss: 0.8753 - val_accuracy: 0.8
Epoch 64/70
65/65 [==============================] - 44s 677ms/step - loss: 0.1169 - accuracy: 0.9490 - val_loss: 0.7585 - val_accuracy: 0.8
Epoch 65/70
65/65 [==============================] - 45s 692ms/step - loss: 0.1103 - accuracy: 0.9538 - val_loss: 1.0303 - val_accuracy: 0.7
Epoch 66/70
65/65 [==============================] - 46s 710ms/step - loss: 0.1390 - accuracy: 0.9427 - val_loss: 0.8088 - val_accuracy: 0.8
Epoch 67/70
65/65 [==============================] - 46s 701ms/step - loss: 0.1209 - accuracy: 0.9461 - val_loss: 1.1016 - val_accuracy: 0.8
Epoch 68/70
65/65 [==============================] - 47s 715ms/step - loss: 0.1247 - accuracy: 0.9427 - val_loss: 1.2932 - val_accuracy: 0.6
Epoch 69/70
65/65 [==============================] - 44s 674ms/step - loss: 0.1038 - accuracy: 0.9543 - val_loss: 0.7441 - val_accuracy: 0.7
Epoch 70/70
65/65 [==============================] - 44s 674ms/step - loss: 0.1246 - accuracy: 0.9466 - val_loss: 1.9975 - val_accuracy: 0.7
```

```
# Plot the loss vs val_loss
plt.plot(history.history['loss'], label='Training Loss')
```

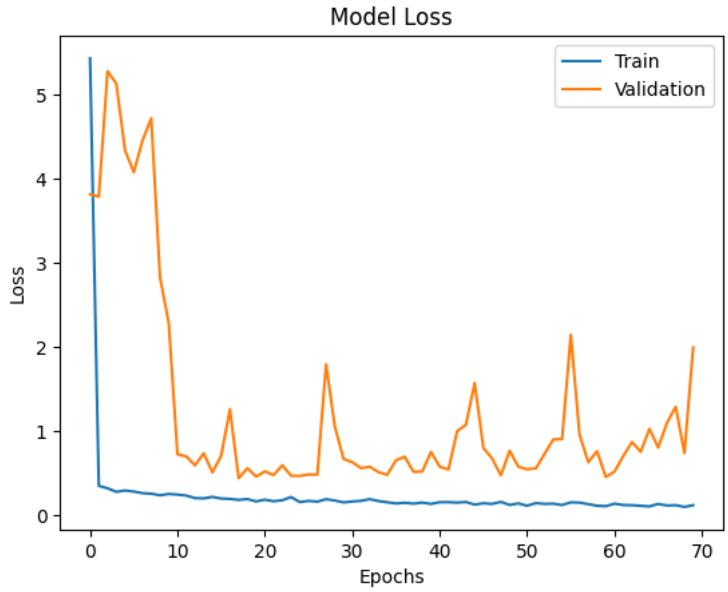
```
plt.plot(history.history['val_loss'], label='Validation Loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()
plt.show()

# Plot the accuracy vs val_accuracy
plt.plot(history.history['accuracy'], label='Training Accuracy')
plt.plot(history.history['val_accuracy'], label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```





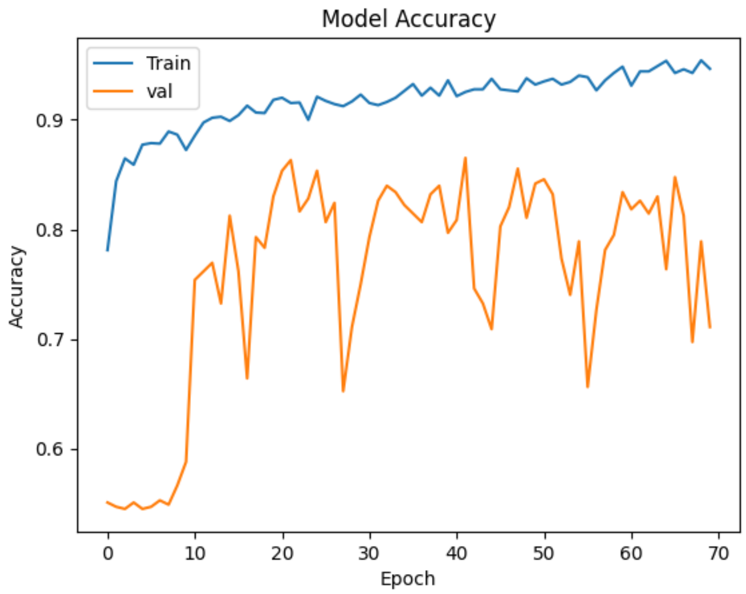

```
# Evaluate the model on the testing data
test_loss, test_acc = model.evaluate(test_generator, verbose=2)
print('Test Accuracy:', test_acc)
# Print the train and test loss
print('Train Loss:', history.history['loss'][-1])
print('Test Loss:', test_loss)
print('Test Accuracy:', test_acc)
# Plot the train and validation loss over epochs
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend(['Train', 'Validation'])
plt.show()
```

```
1/1 - 22s - loss: 1.0056 - accuracy: 0.8263 - 22s/epoch - 22s/step
Test Accuracy: 0.8262839913368225
Train Loss: 0.1245984137058258
Test Loss: 1.0056229829788208
Test Accuracy: 0.8262839913368225
```




```
# Evaluate the model on the testing data
test_loss, test_acc = model.evaluate(test_generator, verbose=2)
print('Test Accuracy:', test_acc)

# Plot the training and testing accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(['Train', 'val'], loc='upper left')
plt.show()
```

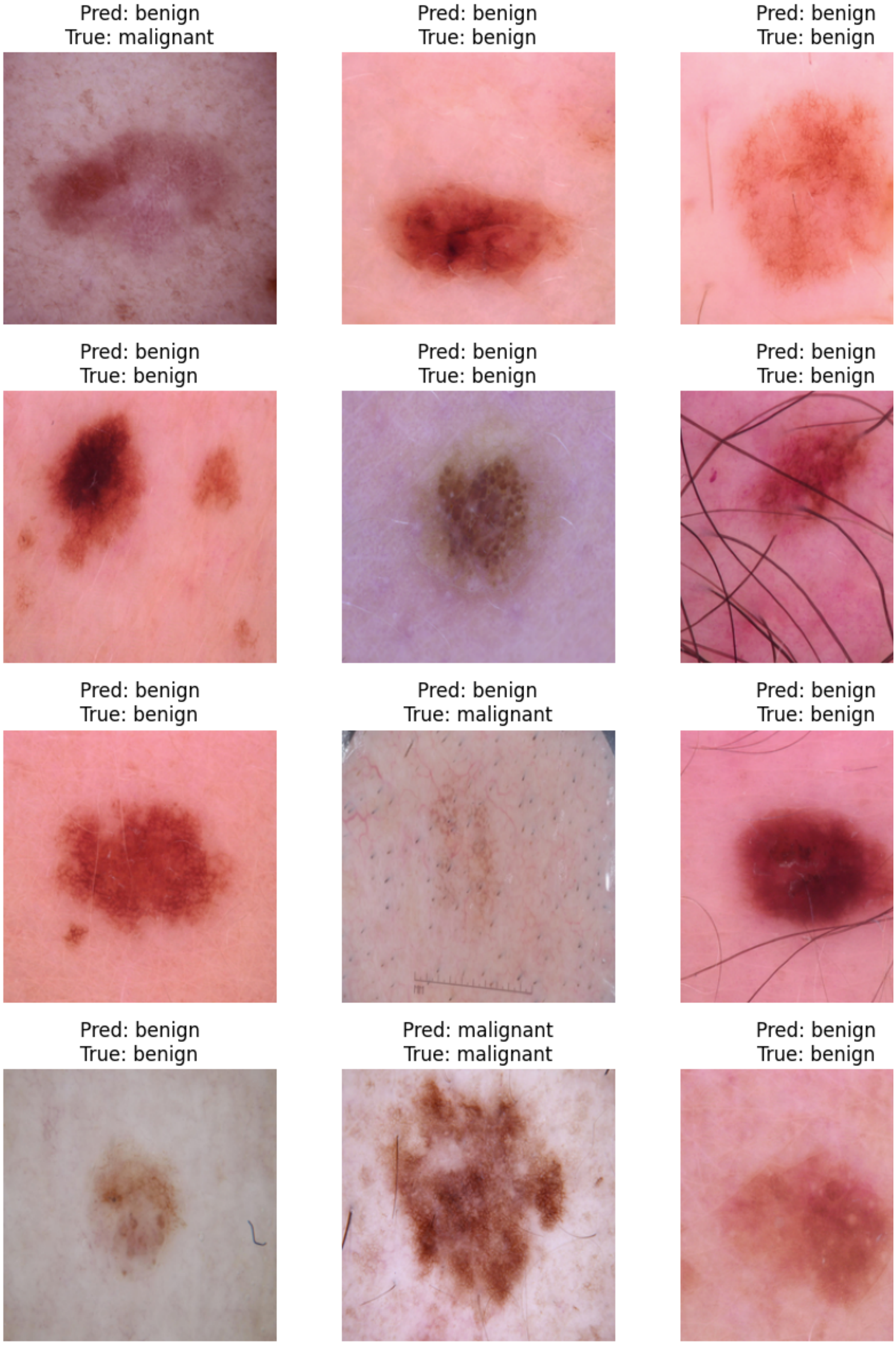
```
1/1 - 5s - loss: 1.0056 - accuracy: 0.8263 - 5s/epoch - 5s/step
Test Accuracy: 0.8262839913368225
```




```
import matplotlib.pyplot as plt

# Get the next batch of images from the test generator
x_test, y_test = next(test_generator)

# Predict the classes of the testing data
y_pred = model.predict(x_test)
y_pred_classes = np.argmax(y_pred, axis=1)
y_true_classes = np.argmax(y_test, axis=1)
label_dict = {0: 'benign', 1: 'malignant'}

# Plot the images along with their predicted and actual labels
fig, axes = plt.subplots(nrows=4, ncols=4, figsize=(12,12))
```

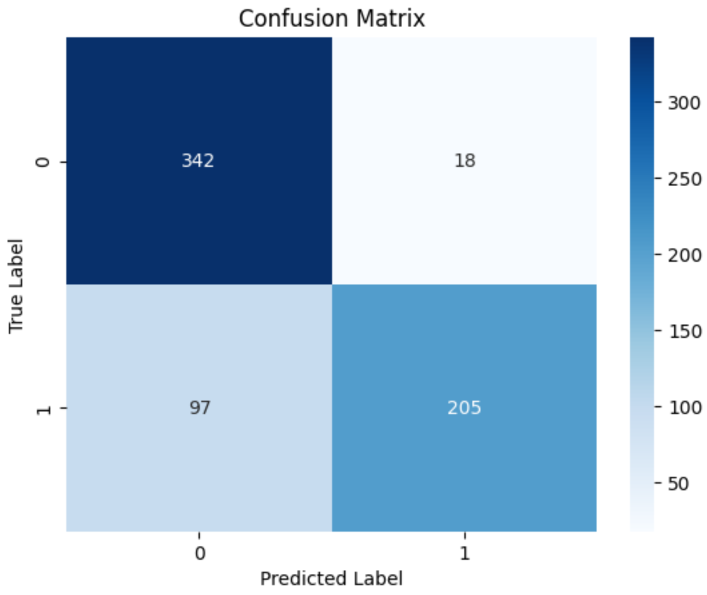
```
for i, ax in enumerate(axes.flat):
    ax.imshow(x_test[i])
    pred_label = label_dict[y_pred_classes[i]]
    true_label = label_dict[y_true_classes[i]]
    ax.set_title("Pred: {}\nTrue: {}".format(pred_label, true_label))
    ax.axis('off')
plt.tight_layout()
plt.show()


```

```
21/21 [==============================] - 4s 138ms/step
```




```
# Create confusion matrix
cm = confusion_matrix(y_true_classes, y_pred_classes)

# Plot confusion matrix
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues')
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
```

```
7 plt.xlabel('Predicted Label')
8 plt.ylabel('True Label')
9 plt.show()
```




```
1
```