```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import tensorflow as tf
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.layers import Dense , GlobalAveragePooling2D
from tensorflow.keras.models import Model
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
```

```
from glob import glob
import os
```

```
data_set_path = r"/content/drive/MyDrive/Colab Notebooks/Processed Images_Fruits"
```

```
fruit_name =[]
Images =[]
labels =[]
Quality_category = ['Bad Quality_Fruits','Good Quality_Fruits','Mixed Qualit_Fruits']
for label, category in enumerate(Quality_category):
    category_path = os.path.join(data_set_path, category)
    for fruit_dir in os.listdir(category_path):
        fruit_path = os.path.join(category_path, fruit_dir)
        images = glob(os.path.join(fruit_path, "*.jpg"))
        Images.extend(images)
        fruit_name.extend([fruit_dir]*len(images))
        labels.extend([label] * len(images))
```

```
from PIL import Image
```

```
data = np.array(labels)
labels = np.array(labels)
```

```
df = pd.DataFrame({ 'image_path': Images, 'fruit_name': fruit_name, 'label': labels})
```

```
df['fruit_name']=df['fruit_name'].apply(lambda x: x.split('_')[0])
```

```
fruit_list=df['fruit_name'].unique()
```

```
from sklearn.utils import shuffle
```

```
def shuffle_and_sample(group, frac=0.05):
   group = shuffle(group)
   return group.sample(frac=frac)
```

```
sampled_df = df.groupby('fruit_name', group_keys=False).apply(shuffle_and_sample)
```

```
<ipython-input-12-3308864faaa8>:1: DeprecationWarning: DataFrameGroupBy.apply operated on the grouping columns. This behavior is deprecated, and in a futu
  sampled_df = df.groupby('fruit_name', group_keys=False).apply(shuffle_and_sample)
```

```
print(f"Total images in original DataFrame: {len(df)}")
print(f"Total images in sampled DataFrame: {len(sampled_df)}")
print(sampled_df.head(20))
```

```
Total images in original DataFrame: 15819
Total images in sampled DataFrame: 791
                                              image_path fruit_name  label
14702  /content/drive/MyDrive/Colab Notebooks/Process...      Apple      1
59     /content/drive/MyDrive/Colab Notebooks/Process...      Apple      0
14226  /content/drive/MyDrive/Colab Notebooks/Process...      Apple      1
14267  /content/drive/MyDrive/Colab Notebooks/Process...      Apple      1
966    /content/drive/MyDrive/Colab Notebooks/Process...      Apple      0
94     /content/drive/MyDrive/Colab Notebooks/Process...      Apple      0
325    /content/drive/MyDrive/Colab Notebooks/Process...      Apple      0
```

```
14052  /content/drive/MyDrive/Colab Notebooks/Process...    Apple    1
840    /content/drive/MyDrive/Colab Notebooks/Process...    Apple    0
774    /content/drive/MyDrive/Colab Notebooks/Process...    Apple    0
385    /content/drive/MyDrive/Colab Notebooks/Process...    Apple    0
14354  /content/drive/MyDrive/Colab Notebooks/Process...    Apple    1
305    /content/drive/MyDrive/Colab Notebooks/Process...    Apple    0
971    /content/drive/MyDrive/Colab Notebooks/Process...    Apple    0
14691  /content/drive/MyDrive/Colab Notebooks/Process...    Apple    1
571    /content/drive/MyDrive/Colab Notebooks/Process...    Apple    0
14526  /content/drive/MyDrive/Colab Notebooks/Process...    Apple    1
13984  /content/drive/MyDrive/Colab Notebooks/Process...    Apple    1
14481  /content/drive/MyDrive/Colab Notebooks/Process...    Apple    1
166    /content/drive/MyDrive/Colab Notebooks/Process...    Apple    0
```

```
X = sampled_df.drop('label',axis=1)
y = sampled_df['label']
```

```
X['Image']= X['image_path'].apply(lambda x: Image.open(x))
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
train= np.array(X_train['Image'])
test= np.array(X_test['Image'])
```

```
def resize_image(images, width, height):
    resized_images = []
    for image in images:
        image = image.convert('RGB')
        resized_image = image.resize((width, height))
        resized_image = np.array(resized_image)/ 255
        resized_images.append(resized_image)
    return np.array(resized_images)
```

```
train = resize_image(train, 224, 224)
test = resize_image(test, 224, 224)
```

```
train = tf.image.resize(train, (96, 96))
test = tf.image.resize(test, (96, 96))
```

```
train.shape
```

```
TensorShape([632, 96, 96, 3])
```

```
def build_cnn_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(96, 96, 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.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(3, activation='softmax')
    ])
    return model
```

```
cnn_model = build_cnn_model()
cnn_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

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

```
cnn_model.fit(train, y_train, epochs=10, validation_data=(test, y_test))
```

```
Epoch 1/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 16s 689ms/step - accuracy: 0.4284 - loss: 1.2666 - val_accuracy: 0.5849 - val_loss: 0.7820
Epoch 2/10
```

```
20/20 ━━━━━━━━━━━━━━━━━━━━ 12s 603ms/step - accuracy: 0.6505 - loss: 0.7383 - val_accuracy: 0.7862 - val_loss: 0.6314
Epoch 3/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 22s 692ms/step - accuracy: 0.7786 - loss: 0.5898 - val_accuracy: 0.6730 - val_loss: 0.7149
Epoch 4/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 19s 581ms/step - accuracy: 0.7945 - loss: 0.5012 - val_accuracy: 0.8365 - val_loss: 0.4653
Epoch 5/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 21s 641ms/step - accuracy: 0.8433 - loss: 0.3741 - val_accuracy: 0.8742 - val_loss: 0.4505
Epoch 6/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 21s 674ms/step - accuracy: 0.8739 - loss: 0.2891 - val_accuracy: 0.8616 - val_loss: 0.4858
Epoch 7/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 20s 664ms/step - accuracy: 0.9117 - loss: 0.2586 - val_accuracy: 0.8742 - val_loss: 0.4710
Epoch 8/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 20s 634ms/step - accuracy: 0.9154 - loss: 0.2201 - val_accuracy: 0.8616 - val_loss: 0.4850
Epoch 9/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 19s 568ms/step - accuracy: 0.9297 - loss: 0.2111 - val_accuracy: 0.8428 - val_loss: 0.7242
Epoch 10/10
20/20 ━━━━━━━━━━━━━━━━━━━━ 22s 678ms/step - accuracy: 0.9305 - loss: 0.1669 - val_accuracy: 0.8679 - val_loss: 0.4880
<keras.src.callbacks.history.History at 0x7d2a74ef39d0>
```

```
cnn_loss,cnn_acc = cnn_model.evaluate(test, y_test)
print(f"CNN Loss: {cnn_loss}")
print(f"CNN Accuracy: {cnn_acc}")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 307ms/step - accuracy: 0.8722 - loss: 0.5256
CNN Loss: 0.4880344271659851
CNN Accuracy: 0.8679245114326477
```

```
y_pred= cnn_model.predict(test)
predicted_labels = np.argmax(y_pred, axis=1)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 1s 258ms/step
```

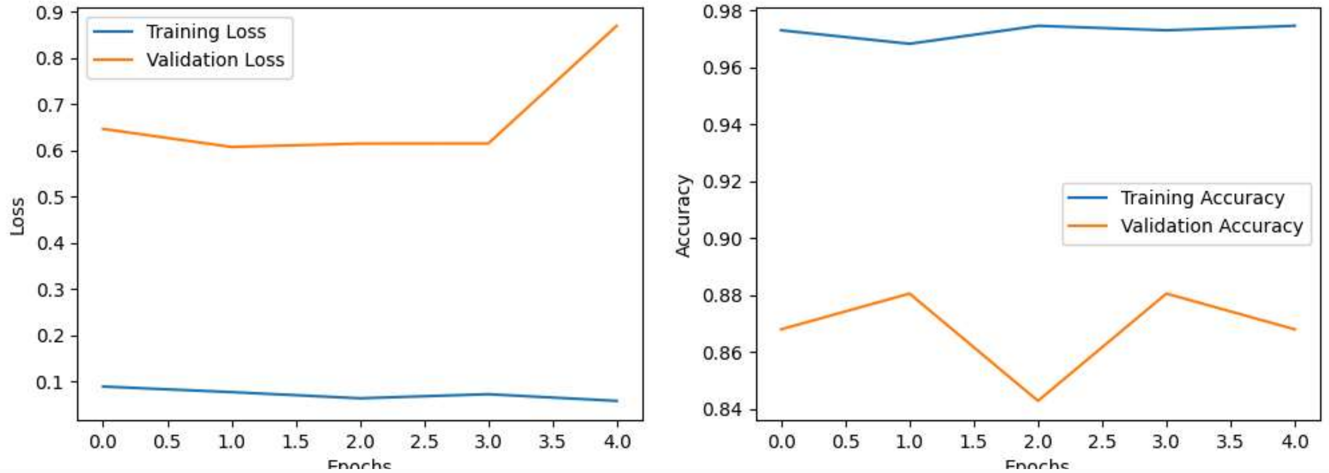
```
predicted_labels
```

```
array([1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 2, 1, 1, 1, 0, 1, 0, 0, 1, 0, 2, 0, 0, 2, 0, 1, 2, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 2, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 2, 0,
       1, 1, 1, 1, 1, 2, 0, 1, 1, 1, 2, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1])
```

```
def plot_training_history(History,title):
  plt.figure(figsize=(12,4))
  plt.subplot(1,2,1)
  plt.plot(History.history['loss'],label='Training Loss')
  plt.plot(History.history['val_loss'],label='Validation Loss')
  plt.xlabel('Epochs')
  plt.ylabel('Loss')
  plt.legend()
  plt.subplot(1,2,2)
  plt.plot(History.history['accuracy'],label='Training Accuracy')
  plt.plot(History.history['val_accuracy'],label='Validation Accuracy')
  plt.xlabel('Epochs')
  plt.ylabel('Accuracy')
  plt.legend()
```

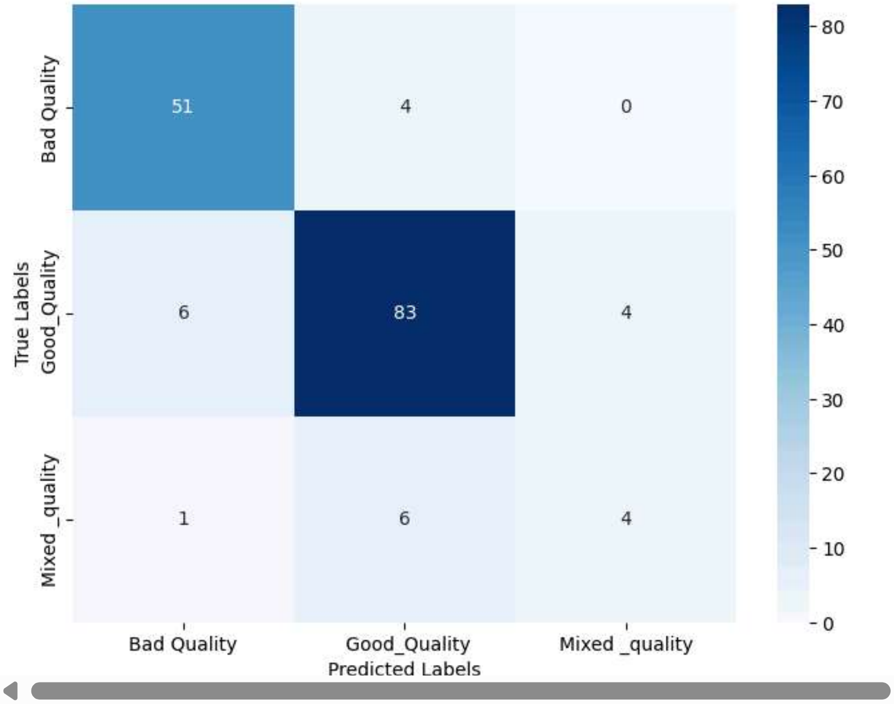
```
plot_training_history(cnn_model.fit(train, y_train, epochs=5, validation_data=(test, y_test)),'CNN')
```

```
Epoch 1/5
20/20 ──────────────────────────── 14s 698ms/step - accuracy: 0.9766 - loss: 0.0905 - val_accuracy: 0.8679 - val_loss: 0.6459
Epoch 2/5
20/20 ──────────────────────────── 15s 738ms/step - accuracy: 0.9670 - loss: 0.0990 - val_accuracy: 0.8805 - val_loss: 0.6069
Epoch 3/5
20/20 ──────────────────────────── 19s 662ms/step - accuracy: 0.9811 - loss: 0.0567 - val_accuracy: 0.8428 - val_loss: 0.6142
Epoch 4/5
20/20 ──────────────────────────── 20s 674ms/step - accuracy: 0.9734 - loss: 0.0689 - val_accuracy: 0.8805 - val_loss: 0.6145
Epoch 5/5
20/20 ──────────────────────────── 14s 680ms/step - accuracy: 0.9829 - loss: 0.0501 - val_accuracy: 0.8679 - val_loss: 0.8691
```



```
cm = confusion_matrix(y_test, predicted_labels)


plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Bad Quality',' Good_Quality','Mixed _quality'], yticklabels=['Bad Quality',' Good_Quality','Mixec
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
```

```
Text(70.58159722222221, 0.5, 'True Labels')
```



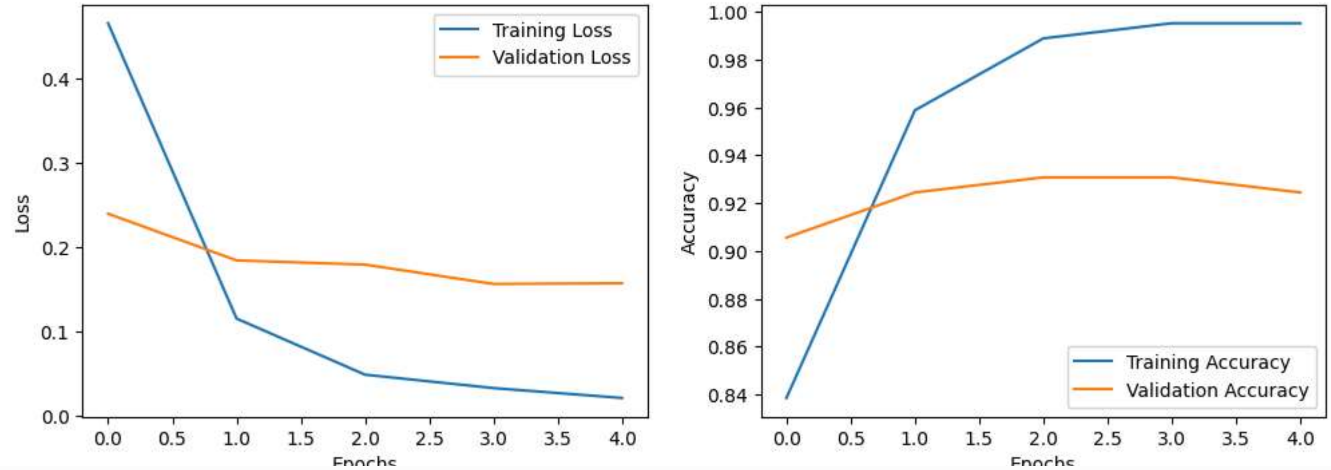
```
print(classification_report(y_test, predicted_labels))
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90        55
           1       0.89      0.89      0.89        93
           2       0.50      0.36      0.42        11
```

```
    accuracy                           0.87       159
   macro avg       0.76      0.73      0.74       159
weighted avg       0.86      0.87      0.86       159
```

```
base_model= MobileNetV2(input_shape=(96,96,3),include_top=False,weights='imagenet')
base_model.trainable = False

x=base_model.output
x=GlobalAveragePooling2D()(x)
x= Dense(128,activation="relu")(x)
output = Dense(3, activation='softmax')(x)
transfer_model = Model(inputs=base_model.input, outputs=output)
transfer_model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
transfer_history = transfer_model.fit(train, y_train, epochs=5, validation_data=(test, y_test))
plot_training_history(transfer_history,'Transfer Learning')
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobilenet_v2/mobilenet_v2_weights_tf_dim_ordering_tf_kernels_1.0_
9406464/9406464 ━━━━━━━━━━━━━━━━━━━━ 0s 0us/step
Epoch 1/5
20/20 ━━━━━━━━━━━━━━━━━━━━ 18s 589ms/step - accuracy: 0.7106 - loss: 0.7342 - val_accuracy: 0.9057 - val_loss: 0.2400
Epoch 2/5
20/20 ━━━━━━━━━━━━━━━━━━━━ 15s 313ms/step - accuracy: 0.9409 - loss: 0.1456 - val_accuracy: 0.9245 - val_loss: 0.1845
Epoch 3/5
20/20 ━━━━━━━━━━━━━━━━━━━━ 5s 254ms/step - accuracy: 0.9864 - loss: 0.0589 - val_accuracy: 0.9308 - val_loss: 0.1796
Epoch 4/5
20/20 ━━━━━━━━━━━━━━━━━━━━ 7s 372ms/step - accuracy: 0.9973 - loss: 0.0290 - val_accuracy: 0.9308 - val_loss: 0.1566
Epoch 5/5
20/20 ━━━━━━━━━━━━━━━━━━━━ 8s 244ms/step - accuracy: 0.9969 - loss: 0.0173 - val_accuracy: 0.9245 - val_loss: 0.1574
```



```
new_model_loss,new_model_acc = transfer_model.evaluate(test, y_test)
print(f"mobile_net Loss: {new_model_loss}")
print(f"mobile_net Accuracy: {new_model_acc}")
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 2s 308ms/step - accuracy: 0.9028 - loss: 0.2192
mobile_net Loss: 0.15740491449832916
mobile_net Accuracy: 0.9245283007621765
```

```
new_pred= transfer_model.predict(test)
new_predicted_labels = np.argmax(y_pred, axis=1)
```

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 4s 527ms/step
```

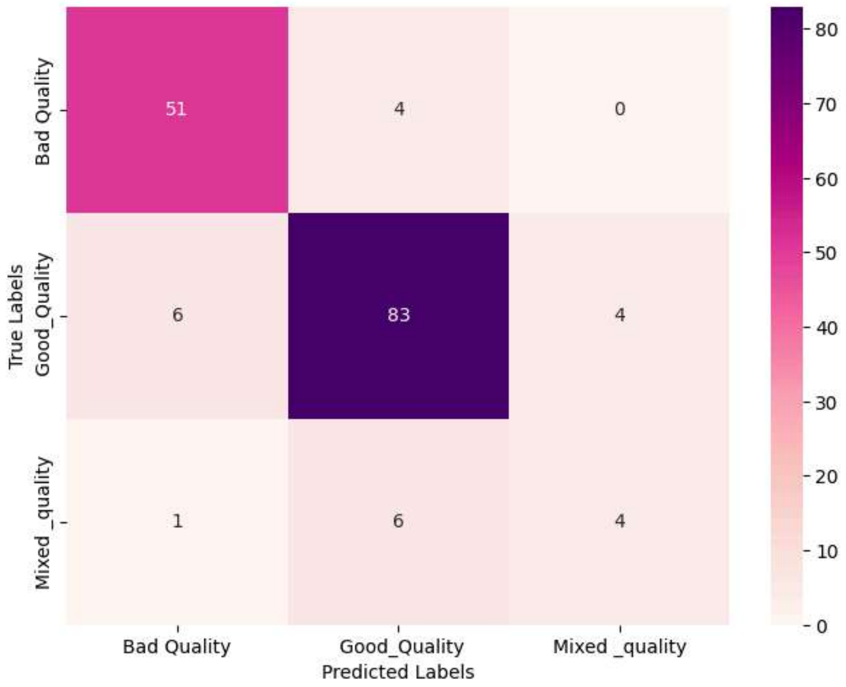
```
new_predicted_labels
```

```
array([1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1,
       1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 0, 1, 1, 1,
       0, 1, 2, 1, 1, 1, 0, 1, 0, 0, 1, 0, 2, 0, 0, 2, 0, 1, 2, 1, 0, 1,
       1, 1, 1, 1, 0, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 2, 1,
       1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 2, 0,
       1, 1, 1, 1, 1, 2, 0, 1, 1, 1, 2, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1,
       0, 0, 1, 1, 1])
```

```
new_cm=confusion_matrix(y_test, new_predicted_labels)
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(new_cm, annot=True, fmt='d', cmap='RdPu', xticklabels=['Bad Quality',' Good_Quality','Mixed _quality'], yticklabels=['Bad Quality',' Good_Quality','M
plt.xlabel('Predicted Labels')
plt.ylabel('True Labels')
```

```
Text(70.58159722222221, 0.5, 'True Labels')
```



```
print(classification_report(y_test, new_predicted_labels))
```

```
              precision    recall  f1-score   support

           0       0.88      0.93      0.90        55
           1       0.89      0.89      0.89        93
           2       0.50      0.36      0.42        11

    accuracy                           0.87       159
   macro avg       0.76      0.73      0.74       159
weighted avg       0.86      0.87      0.86       159
```

```
transfer_model.save('new_transfered_Quality_prediction_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We re
```

```
print(tf.__version__)
```

```
2.17.1
```

```
cnn_model.save('cnn_Quality_prediction_model.h5')
```

```
WARNING:absl:You are saving your model as an HDF5 file via `model.save()` or `keras.saving.save_model(model)`. This file format is considered legacy. We re
```

Start coding or generate with AI.