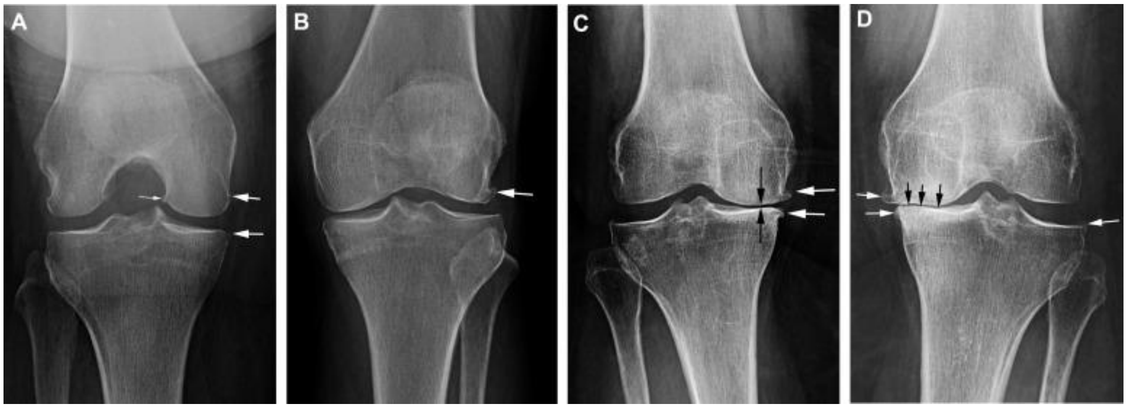
Knee Osteoarthritis Classification using Xception, MobileNet, Attention, Sqeeze and Excitation



```
import numpy as np
import pandas as pd
```

```
base_path = "/kaggle/input/knee-osteoarthritis-classification-224224/Knee Osteoarthritis Classification/train"
categories = ["Normal","Osteopenia", "Osteoporosis"]

image_paths = []
labels = []

for category in categories:
    category_path = os.path.join(base_path, category)
    for image_name in os.listdir(category_path):
        image_path = os.path.join(category_path, image_name)
        image_paths.append(image_path)
        labels.append(category)

df = pd.DataFrame({
    "image_path": image_paths,
    "label": labels
})
```

```
df.head()
```

```
                                          image_path   label
0  /kaggle/input/knee-osteoarthritis-classificati...  Normal
1  /kaggle/input/knee-osteoarthritis-classificati...  Normal
2  /kaggle/input/knee-osteoarthritis-classificati...  Normal
3  /kaggle/input/knee-osteoarthritis-classificati...  Normal
4  /kaggle/input/knee-osteoarthritis-classificati...  Normal
```

```
df.tail()
```

```
                                             image_path         label
3775  /kaggle/input/knee-osteoarthritis-classificati...  Osteoporosis
3776  /kaggle/input/knee-osteoarthritis-classificati...  Osteoporosis
3777  /kaggle/input/knee-osteoarthritis-classificati...  Osteoporosis
3778  /kaggle/input/knee-osteoarthritis-classificati...  Osteoporosis
3779  /kaggle/input/knee-osteoarthritis-classificati...  Osteoporosis
```

```
df.shape
```

```
(3780, 2)
```

```
df.columns
```

```
Index(['image_path', 'label'], dtype='object')
```

```
df.duplicated().sum()
```

```
0
```

```
df.isnull().sum()
```

```
image_path    0
label         0
dtype: int64
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3780 entries, 0 to 3779
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   image_path  3780 non-null   object
 1   label       3780 non-null   object
dtypes: object(2)
memory usage: 59.2+ KB
```

```
df['label'].unique()
```

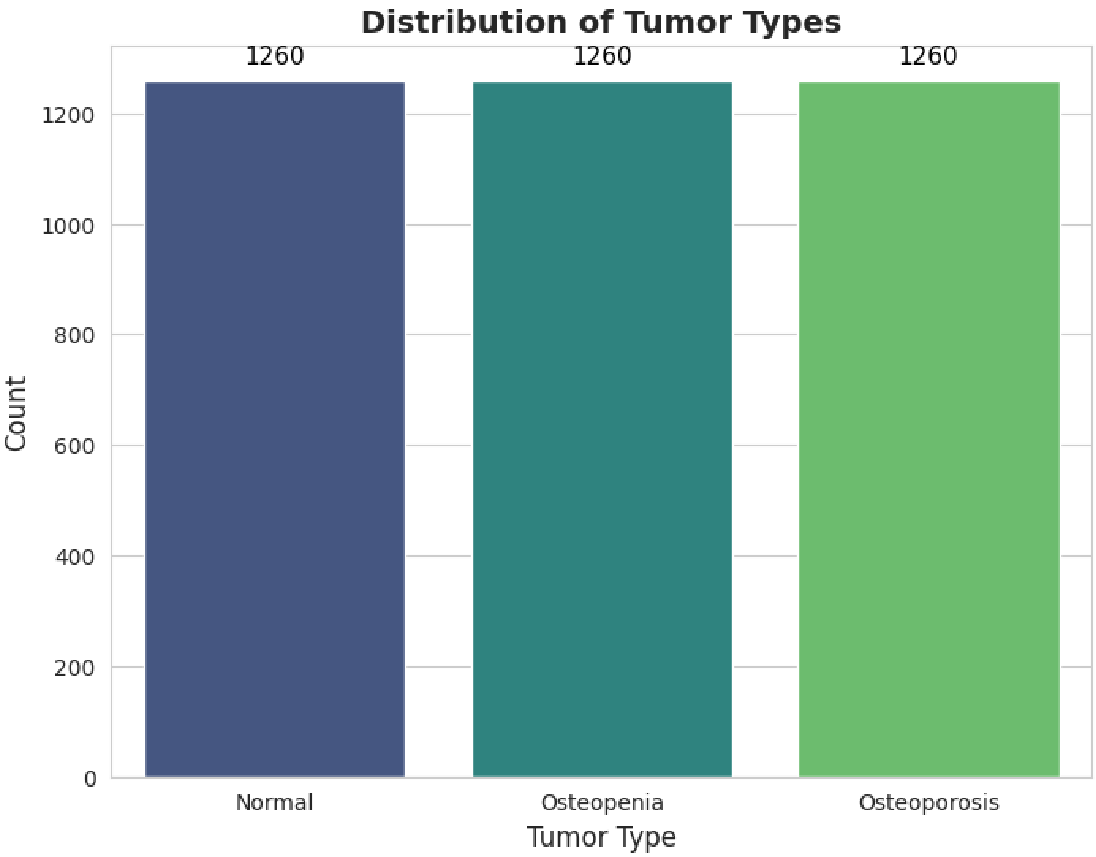
```
array(['Normal', 'Osteopenia', 'Osteoporosis'], dtype=object)
```

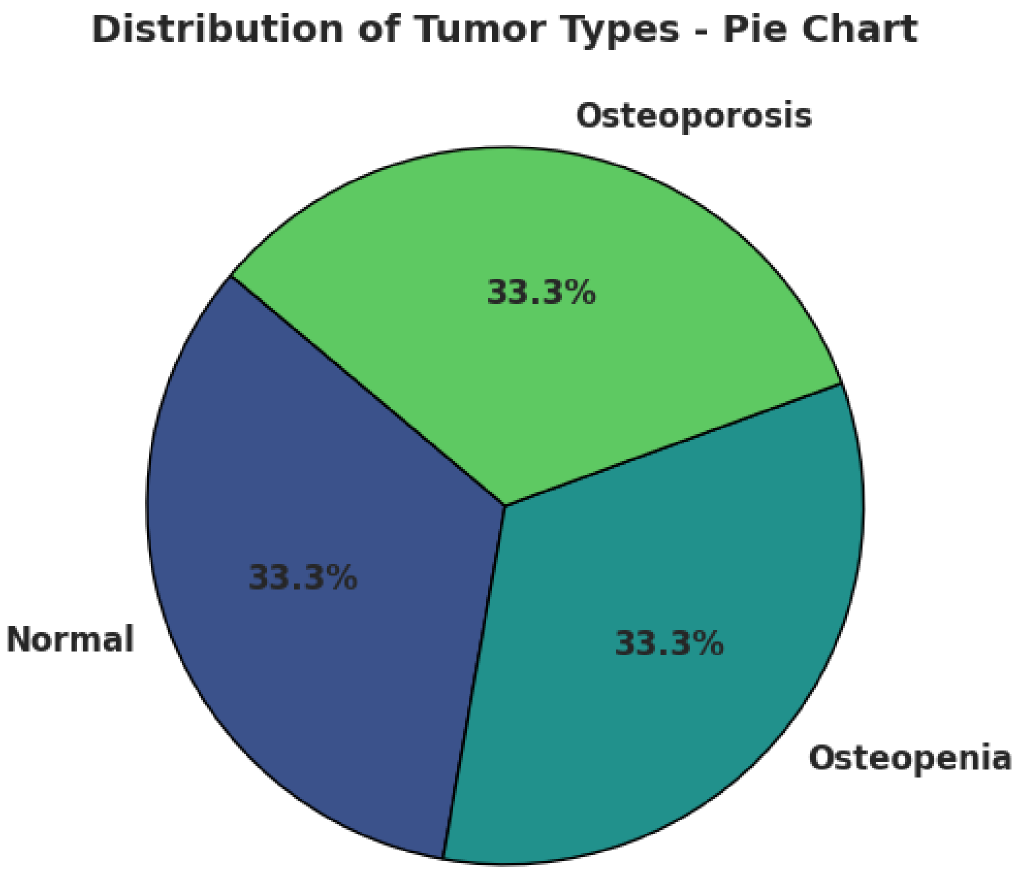
```
df['label'].value_counts()
```

```
label
Normal          1260
Osteopenia      1260
Osteoporosis    1260
Name: count, dtype: int64
```

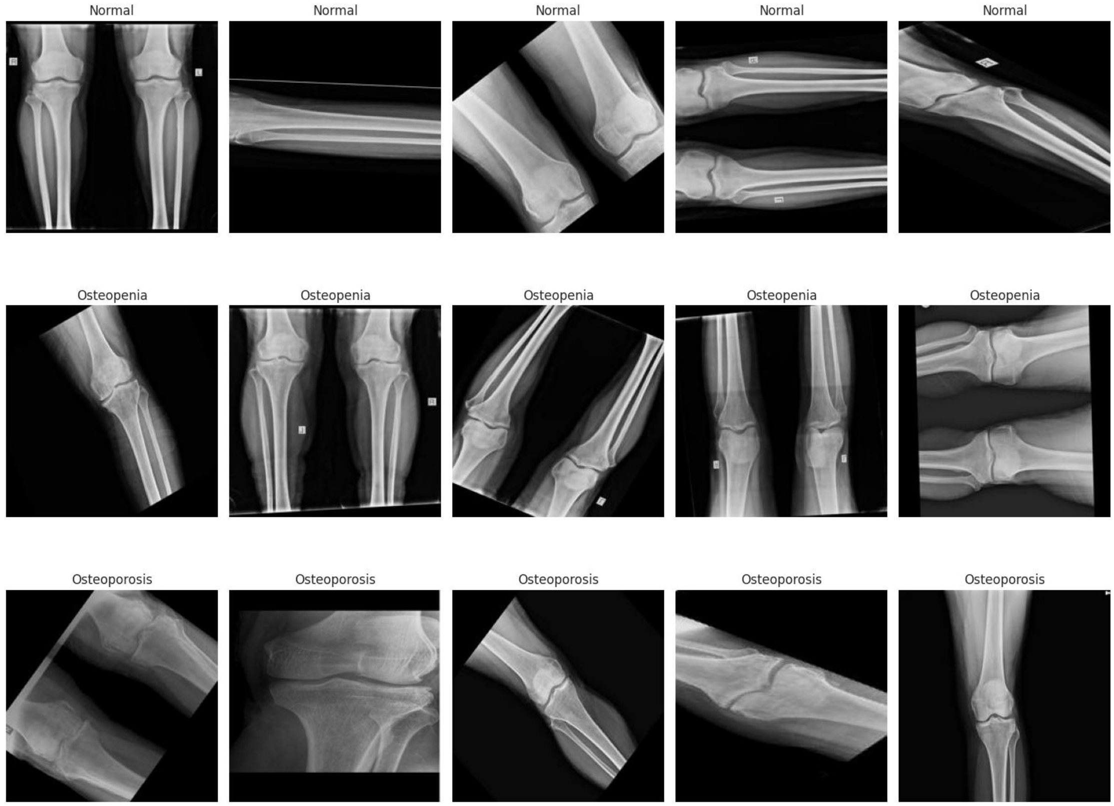
```
import seaborn as sns
import matplotlib.pyplot as plt

sns.set_style("whitegrid")
```

```
fig, ax = plt.subplots(figsize=(8, 6))
sns.countplot(data=df, x="label", palette="viridis", ax=ax)

ax.set_title("Distribution of Tumor Types", fontsize=14, fontweight='bold')
ax.set_xlabel("Tumor Type", fontsize=12)
ax.set_ylabel("Count", fontsize=12)

for p in ax.patches:
    ax.annotate(f'{int(p.get_height())}',
                (p.get_x() + p.get_width() / 2., p.get_height()),
                ha='center', va='bottom', fontsize=11, color='black',
                xytext=(0, 5), textcoords='offset points')

plt.show()

label_counts = df["label"].value_counts()

fig, ax = plt.subplots(figsize=(8, 6))
colors = sns.color_palette("viridis", len(label_counts))

ax.pie(label_counts, labels=label_counts.index, autopct='%1.1f%%',
       startangle=140, colors=colors, textprops={'fontsize': 12, 'weight':
'bold'},
       wedgeprops={'edgecolor': 'black', 'linewidth': 1})

ax.set_title("Distribution of Tumor Types - Pie Chart", fontsize=14,
fontweight='bold')

plt.show()
```


Distribution of Tumor Types
1260
1260
1260
1200
1000
800
600
400
200
0
Count
Normal
Osteopenia
Osteoporosis
Tumor Type



```
import cv2

num_images = 5

plt.figure(figsize=(15, 12))

for i, category in enumerate(categories):
    category_images = df[df['label'] ==
category]['image_path'].iloc[:num_images]

    for j, img_path in enumerate(category_images):

        img = cv2.imread(img_path)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        plt.subplot(len(categories), num_images, i * num_images + j + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(category)
plt.tight_layout()
plt.show()
```



```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()

df['category_encoded'] = label_encoder.fit_transform(df['label'])

df = df[['image_path', 'category_encoded']]

from imblearn.over_sampling import RandomOverSampler

ros = RandomOverSampler(random_state=42)
X_resampled, y_resampled = ros.fit_resample(df[['image_path']],
df['category_encoded'])

df_resampled = pd.DataFrame(X_resampled, columns=['image_path'])
df_resampled['category_encoded'] = y_resampled

print("\nClass distribution after oversampling:")
print(df_resampled['category_encoded'].value_counts())
```

```

Class distribution after oversampling:
category_encoded
0    1260
```

```
1    1260
2    1260
Name: count, dtype: int64
```

```
df_resampled
```

```
                                             image_path  category_encoded
0     /kaggle/input/knee-osteoarthritis-classificati...                 0
1     /kaggle/input/knee-osteoarthritis-classificati...                 0
2     /kaggle/input/knee-osteoarthritis-classificati...                 0
3     /kaggle/input/knee-osteoarthritis-classificati...                 0
4     /kaggle/input/knee-osteoarthritis-classificati...                 0
...                                                 ...               ...
3775  /kaggle/input/knee-osteoarthritis-classificati...                 2
3776  /kaggle/input/knee-osteoarthritis-classificati...                 2
3777  /kaggle/input/knee-osteoarthritis-classificati...                 2
3778  /kaggle/input/knee-osteoarthritis-classificati...                 2
3779  /kaggle/input/knee-osteoarthritis-classificati...                 2

[3780 rows x 2 columns]
```

```
df_resampled['category_encoded'] =
df_resampled['category_encoded'].astype(str)
```

```
import time
import shutil
import pathlib
import itertools
from PIL import Image

import cv2
import seaborn as sns
sns.set_style('darkgrid')
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report

import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,
Activation, Dropout, BatchNormalization
from tensorflow.keras import regularizers

import warnings
warnings.filterwarnings("ignore")

print ('check')
```

check

```
train_df_new, temp_df_new = train_test_split(
    df_resampled,
    train_size=0.8,
    shuffle=True,
    random_state=42,
    stratify=df_resampled['category_encoded']
)

valid_df_new, test_df_new = train_test_split(
    temp_df_new,
    test_size=0.5,
    shuffle=True,
    random_state=42,
    stratify=temp_df_new['category_encoded']
)

batch_size = 16
img_size = (224, 224)
channels = 3
img_shape = (img_size[0], img_size[1], channels)

tr_gen = ImageDataGenerator(rescale=1./255)
ts_gen = ImageDataGenerator(rescale=1./255)

train_gen_new = tr_gen.flow_from_dataframe(
    train_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

valid_gen_new = ts_gen.flow_from_dataframe(
    valid_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=True,
    batch_size=batch_size
)

test_gen_new = ts_gen.flow_from_dataframe(
    test_df_new,
    x_col='image_path',
    y_col='category_encoded',
    target_size=img_size,
    class_mode='sparse',
    color_mode='rgb',
    shuffle=False,
    batch_size=batch_size
)
```

```
Found 3024 validated image filenames belonging to 3 classes.
Found 378 validated image filenames belonging to 3 classes.
Found 378 validated image filenames belonging to 3 classes.
```

```
print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))
```

```
Num GPUs Available:  2
```

```
 gpus = tf.config.list_physical_devices('GPU')
if gpus:
    try:
        for gpu in gpus:
            tf.config.experimental.set_memory_growth(gpu, True)
        print("GPU is set for TensorFlow")
    except RuntimeError as e:
        print(e)
```

```
GPU is set for TensorFlow
```

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint

early_stopping = EarlyStopping(monitor='val_loss', patience=5,
restore_best_weights=True)

from tensorflow.keras.applications import Xception
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, GlobalAveragePooling2D, Dense, Dropout,
    BatchNormalization, GaussianNoise, MultiHeadAttention, Reshape
)
from tensorflow.keras.optimizers import Adam

def create_xception_model(input_shape, num_classes=8, learning_rate=1e-4):
    inputs = Input(shape=input_shape, name="Input_Layer")

    base_model = Xception(weights="imagenet", input_tensor=inputs,
include_top=False)
    base_model.trainable = False

    x = base_model.output
    height, width, channels = x.shape[1], x.shape[2], x.shape[3]

    x = Reshape((height * width, channels), name="Reshape_to_Sequence")(x)
    x = MultiHeadAttention(num_heads=8, key_dim=channels,
name="Multi_Head_Attention")(x, x)
    x = Reshape((height, width, channels), name="Reshape_to_Spatial")(x)

    x = GaussianNoise(0.25, name="Gaussian_Noise")(x)

    x = GlobalAveragePooling2D(name="Global_Avg_Pooling")(x)
    x = Dense(512, activation="relu", name="FC_512")(x)
    x = BatchNormalization(name="Batch_Normalization")(x)
    x = Dropout(0.25, name="Dropout")(x)

    outputs = Dense(num_classes, activation="softmax",
name="Output_Layer")(x)
    model = Model(inputs=inputs, outputs=outputs,
name="Xception_with_Attention")
    model.compile(
        optimizer=Adam(learning_rate=learning_rate),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"]
    )

    return model

input_shape = (224, 224, 3)
cnn_model = create_xception_model(input_shape, num_classes=3,
learning_rate=1e-4)

def ppo_loss(y_true, y_pred):
    epsilon = 0.2
    y_true_one_hot = tf.one_hot(tf.cast(y_true, tf.int32),
depth=y_pred.shape[-1])
    prob_ratio = tf.reduce_sum(y_pred * y_true_one_hot, axis=-1) / (
        tf.reduce_sum(tf.stop_gradient(y_pred) * y_true_one_hot, axis=-1) +
1e-10
    )
    clipped_ratio = tf.clip_by_value(prob_ratio, 1 - epsilon, 1 + epsilon)
    loss = -tf.reduce_mean(tf.minimum(prob_ratio * y_true_one_hot,
clipped_ratio * y_true_one_hot))
    return loss

history = cnn_model.fit(
    train_gen_new,
    validation_data=valid_gen_new,
    epochs=10,
    callbacks=[early_stopping],
    verbose=1
)
```

```
Epoch 1/10
189/189 ━━━━━━━━━━━━━━━━━━━━56s 251ms/step - accuracy: 0.5291 - loss: 1.1050
- val_accuracy: 0.5053 - val_loss: 1.8048
Epoch 2/10
189/189 ━━━━━━━━━━━━━━━━━━━━42s 222ms/step - accuracy: 0.6063 - loss: 0.8506
- val_accuracy: 0.3333 - val_loss: 8.2265
Epoch 3/10
189/189 ━━━━━━━━━━━━━━━━━━━━43s 227ms/step - accuracy: 0.6780 - loss: 0.7560
- val_accuracy: 0.6508 - val_loss: 1.1354
Epoch 4/10
189/189 ━━━━━━━━━━━━━━━━━━━━43s 227ms/step - accuracy: 0.7196 - loss: 0.6676
- val_accuracy: 0.6508 - val_loss: 0.8521
Epoch 5/10
189/189 ━━━━━━━━━━━━━━━━━━━━42s 223ms/step - accuracy: 0.7322 - loss: 0.6309
- val_accuracy: 0.3783 - val_loss: 2.8962
Epoch 6/10
189/189 ━━━━━━━━━━━━━━━━━━━━43s 227ms/step - accuracy: 0.7569 - loss: 0.5762
- val_accuracy: 0.7063 - val_loss: 0.8060
Epoch 7/10
189/189 ━━━━━━━━━━━━━━━━━━━━42s 223ms/step - accuracy: 0.7967 - loss: 0.4922
- val_accuracy: 0.4577 - val_loss: 3.5726
Epoch 8/10
189/189 ━━━━━━━━━━━━━━━━━━━━42s 223ms/step - accuracy: 0.8134 - loss: 0.4838
- val_accuracy: 0.6349 - val_loss: 1.1248
Epoch 9/10
189/189 ━━━━━━━━━━━━━━━━━━━━42s 223ms/step - accuracy: 0.8271 - loss: 0.4514
- val_accuracy: 0.6005 - val_loss: 1.6182
Epoch 10/10
189/189 ━━━━━━━━━━━━━━━━━━━━42s 223ms/step - accuracy: 0.8408 - loss: 0.4000
- val_accuracy: 0.7381 - val_loss: 0.8623
```

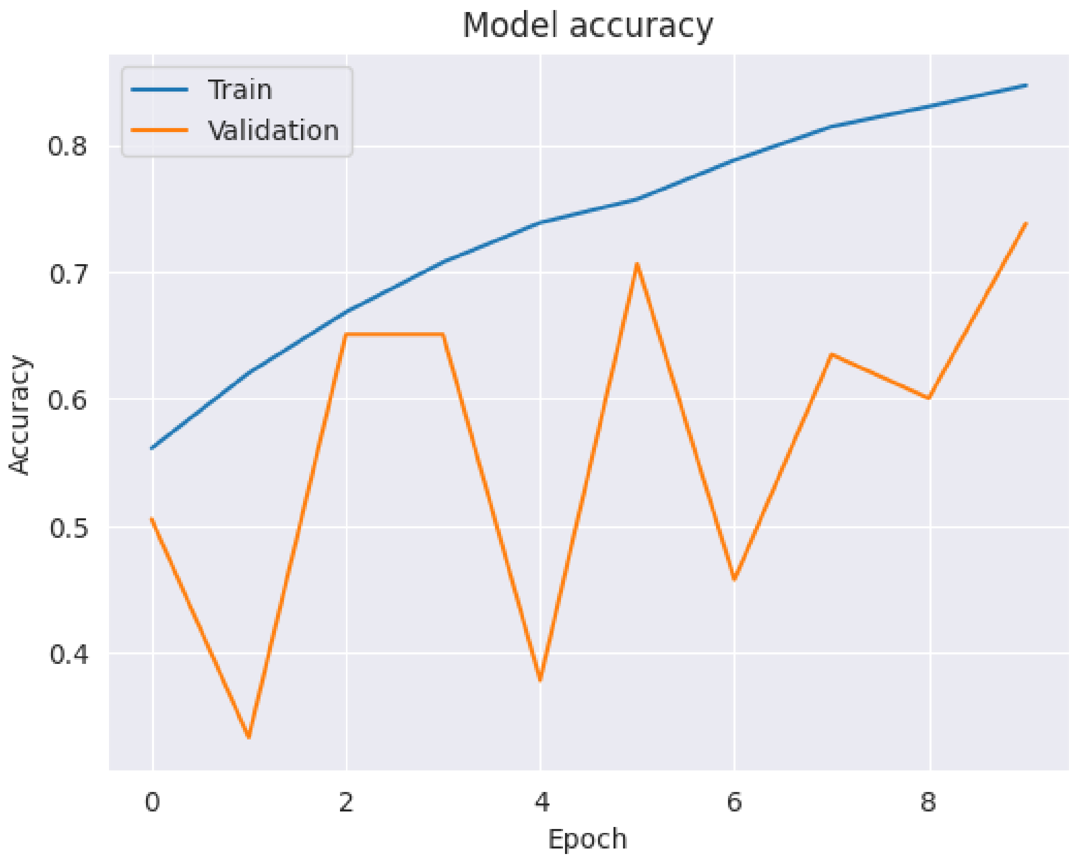
```
y_pred = cnn_model.predict(valid_gen_new)
y_true = valid_gen_new.labels
```

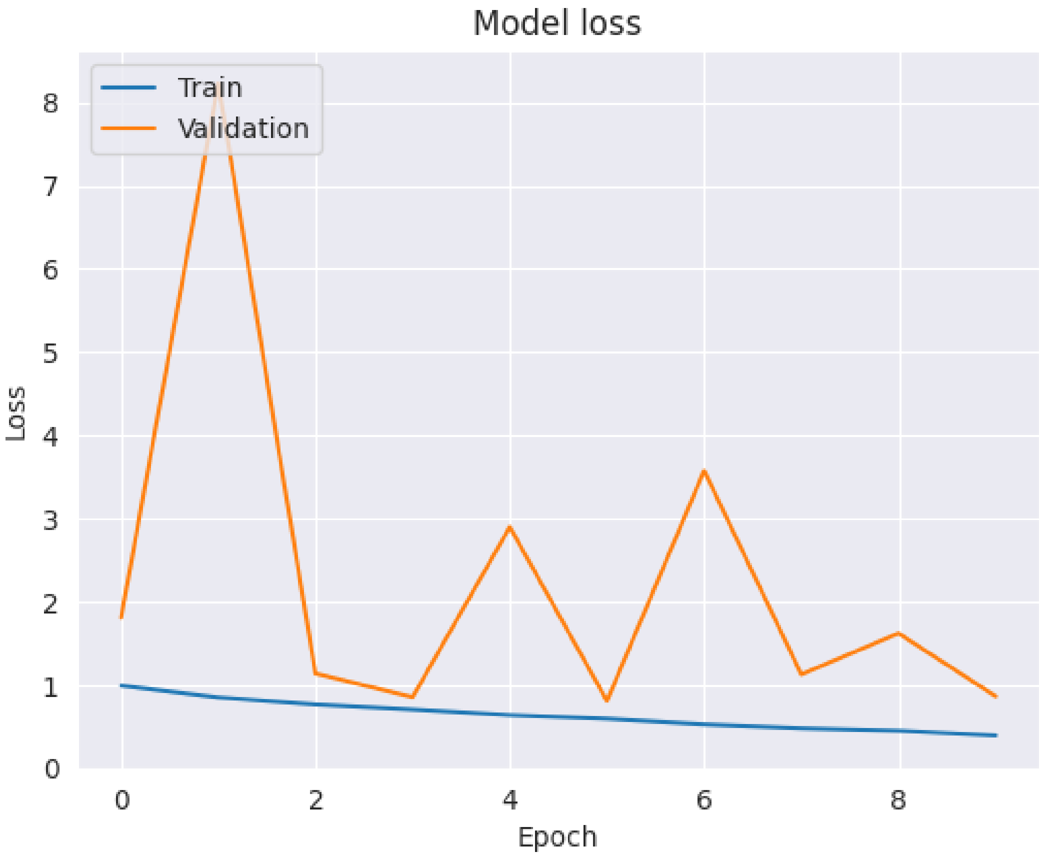
```
24/24 ━━━━━━━━━━━━━━━━━━━━7s 208ms/step
```

```
def ppo_loss(y_true, y_pred):
    epsilon = 0.2
    y_true_one_hot = tf.one_hot(tf.cast(y_true, tf.int32),
depth=tf.shape(y_pred)[-1])
    selected_probs = tf.reduce_sum(y_pred * y_true_one_hot, axis=-1)
    old_selected_probs = tf.reduce_sum(tf.stop_gradient(y_pred) *
y_true_one_hot, axis=-1)
    ratio = selected_probs / (old_selected_probs + 1e-10)
    clipped_ratio = tf.clip_by_value(ratio, 1 - epsilon, 1 + epsilon)
    loss = -tf.reduce_mean(tf.minimum(ratio, clipped_ratio))
    return loss

ppo_loss_value = ppo_loss(y_true, y_pred)
print("PPO Loss on Validation Data:", ppo_loss_value.numpy())
```

```
PPO Loss on Validation Data: -0.9999998
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





```
test_labels = test_gen_new.classes
predictions = cnn_model.predict(test_gen_new)
predicted_classes = np.argmax(predictions, axis=1)
```

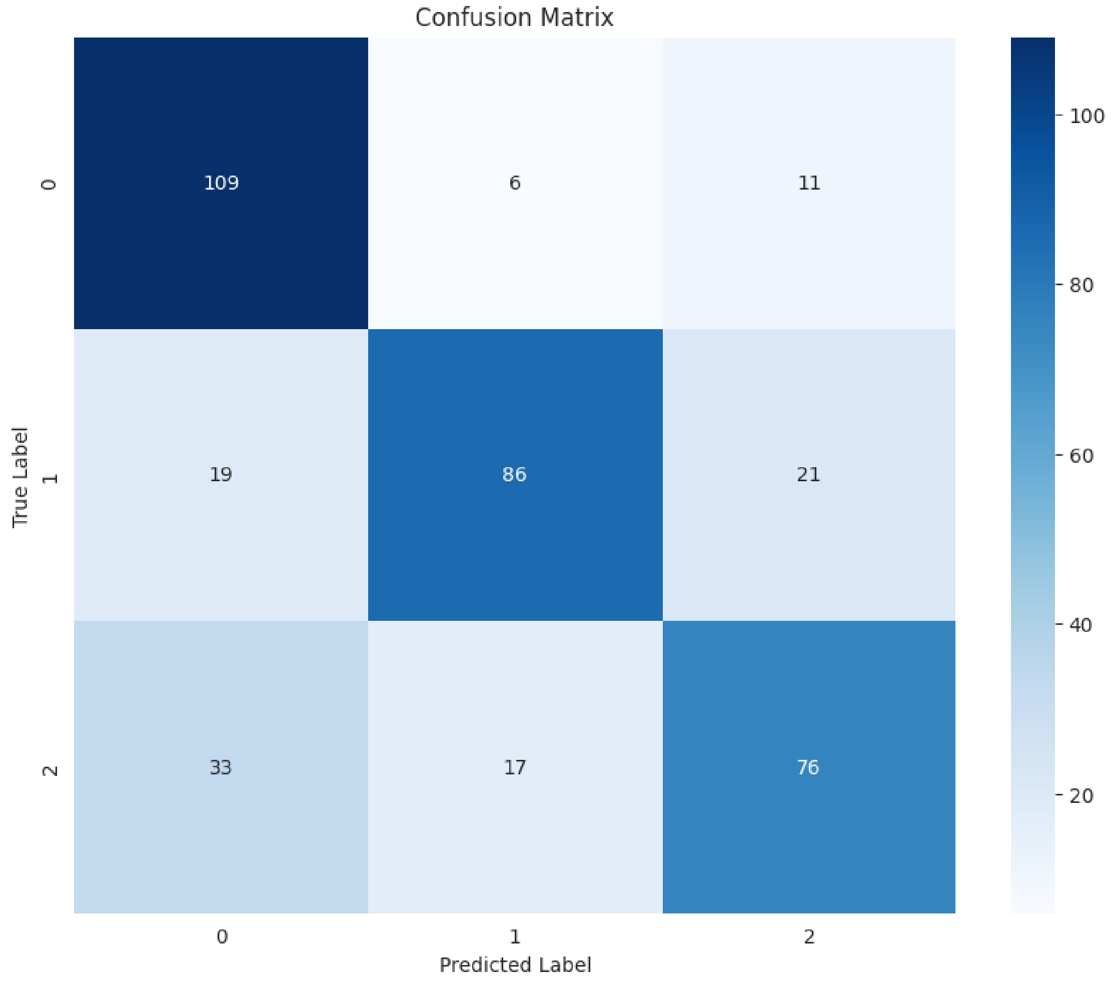
```
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 113ms/step
```

```
report = classification_report(test_labels, predicted_classes,
target_names=list(test_gen_new.class_indices.keys()))
print(report)
```

```
              precision    recall  f1-score   support

           0       0.68      0.87      0.76       126
           1       0.79      0.68      0.73       126
           2       0.70      0.60      0.65       126

    accuracy                           0.72       378
   macro avg       0.72      0.72      0.71       378
weighted avg       0.72      0.72      0.71       378
```

```
conf_matrix = confusion_matrix(test_labels, predicted_classes)
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=list(test_gen_new.class_indices.keys()),
yticklabels=list(test_gen_new.class_indices.keys()))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```



```
from tensorflow.keras.applications import MobileNet
from tensorflow.keras.models import Model
from tensorflow.keras.layers import (
    Input, GlobalAveragePooling2D, Dense, Dropout,
    BatchNormalization, GaussianNoise, MultiHeadAttention,
    Reshape, Conv2D, Multiply
)
from tensorflow.keras.optimizers import Adam

def squeeze_and_excitation_block(input_tensor, ratio=16):
    channels = input_tensor.shape[-1]
    se = GlobalAveragePooling2D(name="SE_Squeeze")(input_tensor)
    se = Dense(channels // ratio, activation="relu",
name="SE_Excitation_1")(se)
    se = Dense(channels, activation="sigmoid", name="SE_Excitation_2")(se)
    se = Multiply(name="SE_Scale")([input_tensor, se])
    return se

def create_mobilenet_model(input_shape, num_classes=8, learning_rate=1e-4):
    inputs = Input(shape=input_shape, name="Input_Layer")

    base_model = MobileNet(weights="imagenet", input_tensor=inputs,
include_top=False)
    base_model.trainable = False

    x = base_model.output
    height, width, channels = x.shape[1], x.shape[2], x.shape[3]

    x = squeeze_and_excitation_block(x)

    x = Reshape((height * width, channels), name="Reshape_to_Sequence")(x)
    x = MultiHeadAttention(num_heads=8, key_dim=channels,
name="Multi_Head_Attention")(x, x)
    x = Reshape((height, width, channels), name="Reshape_to_Spatial")(x)
    x = GaussianNoise(0.25, name="Gaussian_Noise")(x)
    x = GlobalAveragePooling2D(name="Global_Avg_Pooling")(x)
    x = Dense(512, activation="relu", name="FC_512")(x)
    x = BatchNormalization(name="Batch_Normalization")(x)
    x = Dropout(0.25, name="Dropout")(x)

    outputs = Dense(num_classes, activation="softmax",
name="Output_Layer")(x)
    model = Model(inputs=inputs, outputs=outputs,
name="MobileNet_with_Attention_SE")
    model.compile(
        optimizer=Adam(learning_rate=learning_rate),
        loss="sparse_categorical_crossentropy",
        metrics=["accuracy"]
    )

    return model

input_shape = (224, 224, 3)
cnn_model = create_mobilenet_model(input_shape, num_classes=3,
learning_rate=1e-4)

history = cnn_model.fit(
    train_gen_new,
    validation_data=valid_gen_new,
    epochs=10,
    callbacks=[early_stopping],
    verbose=1
)
```

```
Epoch 1/10
189/189 ━━━━━━━━━━━━━━━━━━━━26s 86ms/step - accuracy: 0.5505 - loss: 1.0899
- val_accuracy: 0.5397 - val_loss: 1.0545
Epoch 2/10
189/189 ━━━━━━━━━━━━━━━━━━━━13s 69ms/step - accuracy: 0.6727 - loss: 0.7495
- val_accuracy: 0.6481 - val_loss: 0.9467
Epoch 3/10
189/189 ━━━━━━━━━━━━━━━━━━━━14s 71ms/step - accuracy: 0.7301 - loss: 0.6401
- val_accuracy: 0.6958 - val_loss: 0.7450
Epoch 4/10
189/189 ━━━━━━━━━━━━━━━━━━━━14s 71ms/step - accuracy: 0.7938 - loss: 0.5361
- val_accuracy: 0.7619 - val_loss: 0.7175
Epoch 5/10
189/189 ━━━━━━━━━━━━━━━━━━━━13s 69ms/step - accuracy: 0.8158 - loss: 0.4654
- val_accuracy: 0.6878 - val_loss: 0.9441
Epoch 6/10
189/189 ━━━━━━━━━━━━━━━━━━━━13s 68ms/step - accuracy: 0.8586 - loss: 0.3344
- val_accuracy: 0.6958 - val_loss: 1.0429
Epoch 7/10
189/189 ━━━━━━━━━━━━━━━━━━━━13s 69ms/step - accuracy: 0.8718 - loss: 0.3135
- val_accuracy: 0.7302 - val_loss: 0.9113
Epoch 8/10
189/189 ━━━━━━━━━━━━━━━━━━━━13s 68ms/step - accuracy: 0.8989 - loss: 0.2500
- val_accuracy: 0.6905 - val_loss: 0.9734
Epoch 9/10
189/189 ━━━━━━━━━━━━━━━━━━━━13s 69ms/step - accuracy: 0.9194 - loss: 0.2280
- val_accuracy: 0.7381 - val_loss: 1.1187
```

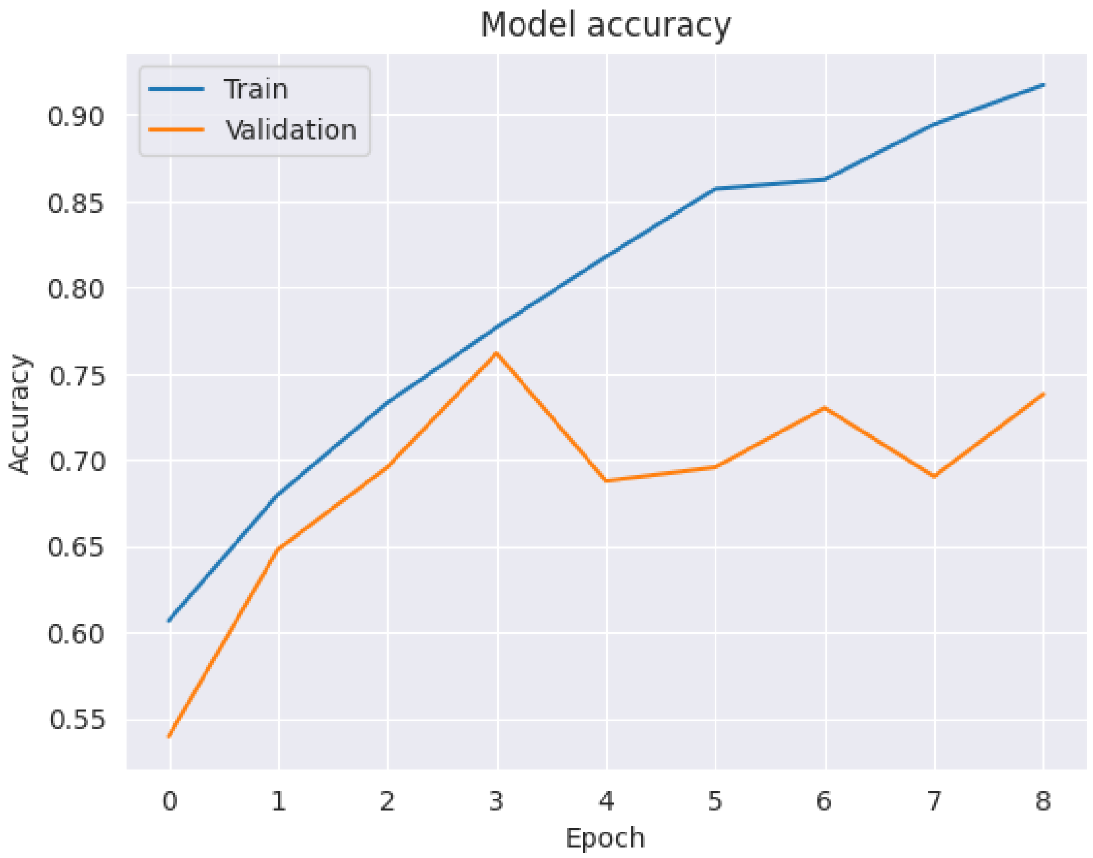
```
y_pred = cnn_model.predict(valid_gen_new)
y_true = valid_gen_new.labels
```

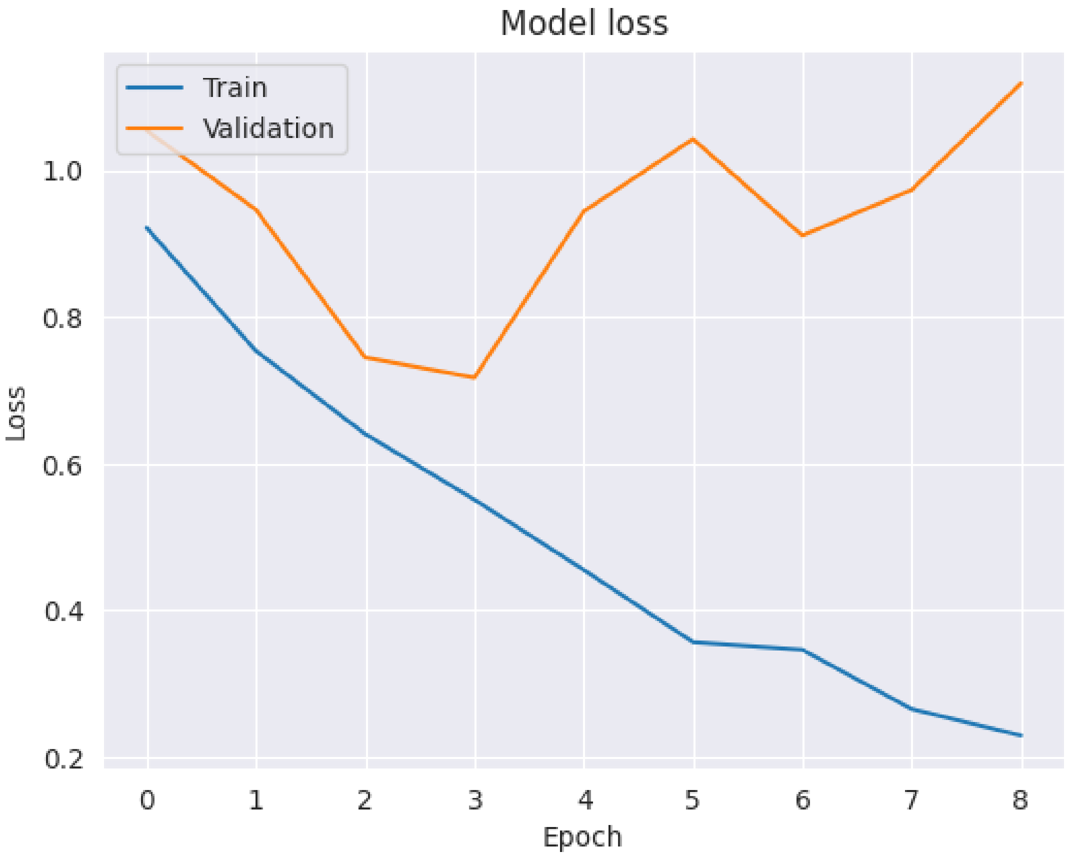
```
24/24 ━━━━━━━━━━━━━━━━━━━━4s 102ms/step
```

```
ppo_loss_value = ppo_loss(y_true, y_pred)
print("PPO Loss on Validation Data:", ppo_loss_value.numpy())
```

```
PPO Loss on Validation Data: -0.9999713
```

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')
plt.show()
```





```
test_labels = test_gen_new.classes
predictions = cnn_model.predict(test_gen_new)
predicted_classes = np.argmax(predictions, axis=1)
```

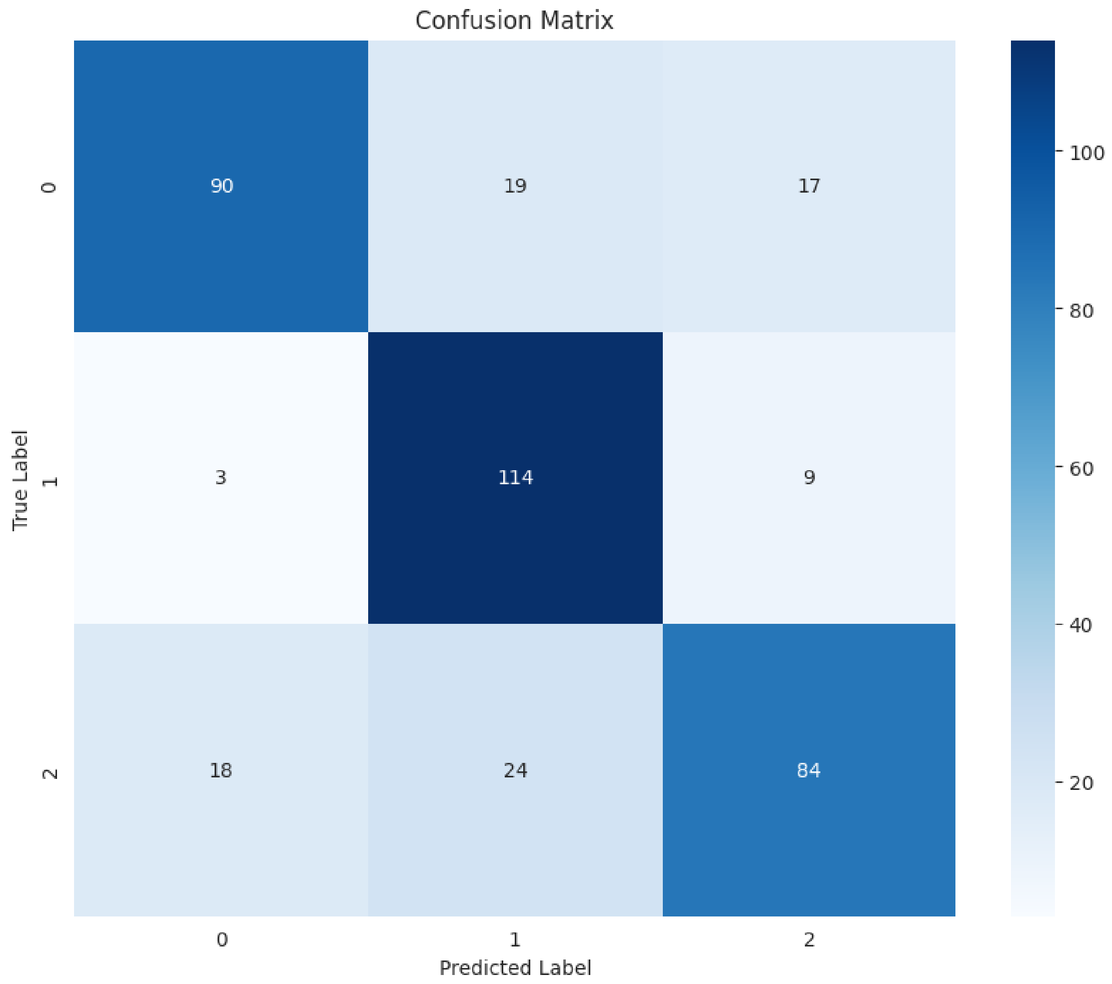
```
24/24 ━━━━━━━━━━━━━━━━━━━━━1s 33ms/step
```

```
report = classification_report(test_labels, predicted_classes,
target_names=list(test_gen_new.class_indices.keys()))
print(report)
```

```
              precision    recall  f1-score   support

           0       0.81      0.71      0.76       126
           1       0.73      0.90      0.81       126
           2       0.76      0.67      0.71       126

    accuracy                           0.76       378
   macro avg       0.77      0.76      0.76       378
weighted avg       0.77      0.76      0.76       378
```

```
conf_matrix = confusion_matrix(test_labels, predicted_classes)
```

```
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=list(test_gen_new.class_indices.keys()),
yticklabels=list(test_gen_new.class_indices.keys()))
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```


Confusion Matrix
True Label
Predicted Label