Libraries and Configurations

```
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow.keras.layers as L
import tensorflow_addons as tfa
import glob, random, os, warnings
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns

print('TensorFlow Version ' + tf.__version__)

import warnings
warnings.filterwarnings("ignore")
```

```
TensorFlow Version 2.6.4
```

```
image_size = 224
batch_size = 16
n_classes = 3
EPOCHS = 30

train_path = '/kaggle/input/pavement-crack-datasetinfrared-only/train'

classes = {1 : "High Crack",
           2 : "Low Crack",
           3 : "Medium Crack",
           4 : "No Crack"}
```

Data Augmentations

```
def data_augment(image):
    p_spatial = tf.random.uniform([], 0, 1.0, dtype = tf.float32)
    p_rotate = tf.random.uniform([], 0, 1.0, dtype = tf.float32)
    p_pixel_1 = tf.random.uniform([], 0, 1.0, dtype = tf.float32)
    p_pixel_2 = tf.random.uniform([], 0, 1.0, dtype = tf.float32)
    p_pixel_3 = tf.random.uniform([], 0, 1.0, dtype = tf.float32)

    # Flips
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_flip_up_down(image)

    if p_spatial > .75:
        image = tf.image.transpose(image)
```

```
    # Rotates
    if p_rotate > .75:
        image = tf.image.rot90(image, k = 3) # rotate 270º
    elif p_rotate > .5:
        image = tf.image.rot90(image, k = 2) # rotate 180º
    elif p_rotate > .25:
        image = tf.image.rot90(image, k = 1) # rotate 90º

    # Pixel-level transforms
    # if p_pixel_1 >= .4:
    #     image = tf.image.random_saturation(image, lower = .7, upper
= 1.3)
    # if p_pixel_2 >= .4:
    #     image = tf.image.random_contrast(image, lower = .8, upper =
1.2)
    # if p_pixel_3 >= .4:
    #     image = tf.image.random_brightness(image, max_delta = .1)

    return image
```

Data Generator

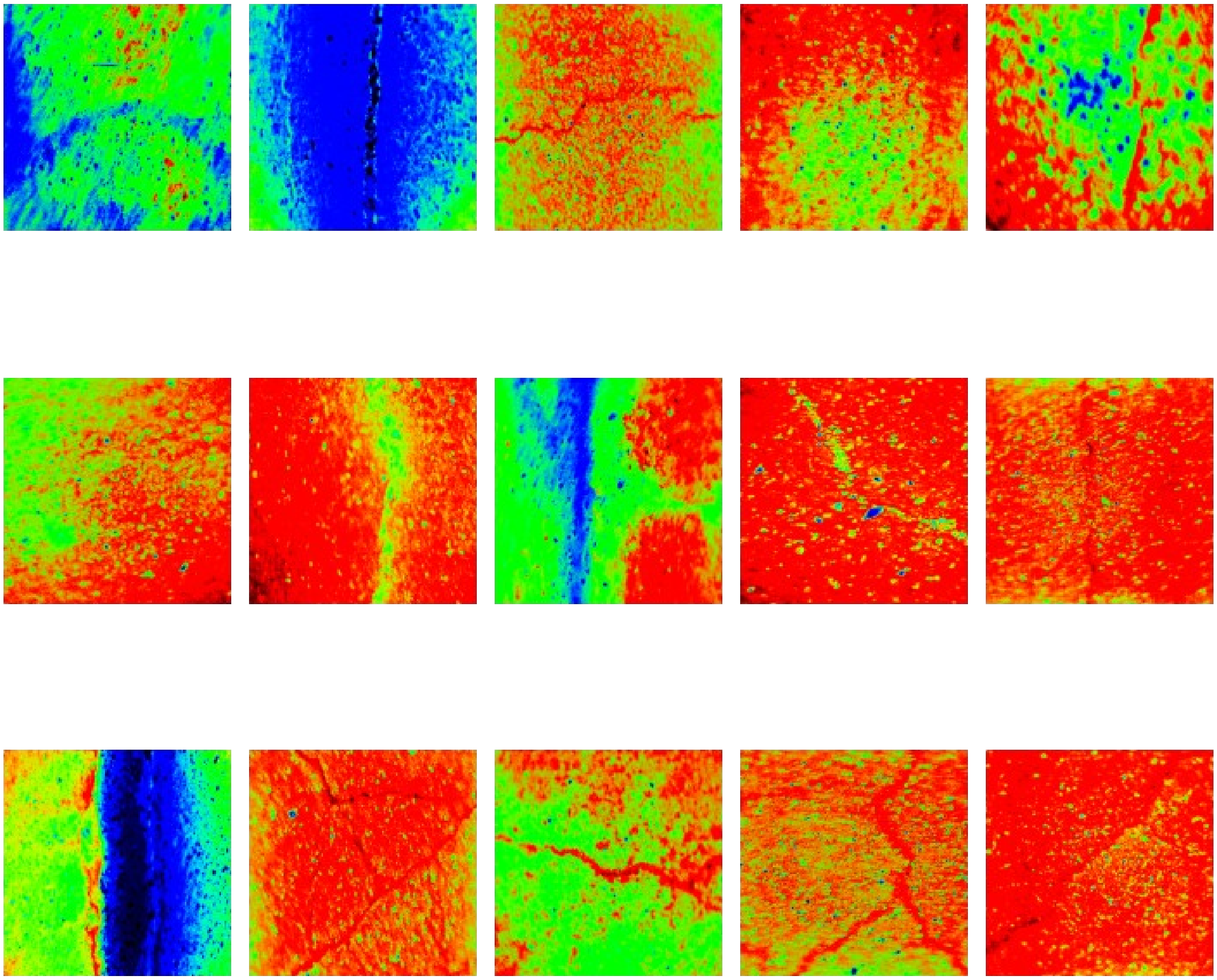
```
datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale =
1./255,

samplewise_center = True,

samplewise_std_normalization = True,

preprocessing_function = data_augment)


# set as training data

train_datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=30,
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.2,
    zoom_range=0.3,
    horizontal_flip=True,
    vertical_flip=True,
)

test_datagen =
tf.keras.preprocessing.image.ImageDataGenerator(rescale=1./255)
```

```
train_gen  = datagen.flow_from_directory(
    train_path,
    target_size=(224, 224),
    batch_size = batch_size,
    seed = 1,
    color_mode = 'rgb',
    shuffle = True,
    class_mode='categorical',
    )

# same directory as training data

valid_gen  = test_datagen.flow_from_directory(
    "/kaggle/input/pavement-crack-datasetinfrared-only/test",
    target_size=(224, 224),
    batch_size = batch_size,
    seed = 1,
    color_mode = 'rgb',
    shuffle = False,
    class_mode='categorical'
    )
```

```
Found 1855 images belonging to 4 classes.
Found 461 images belonging to 4 classes.
```

Sample Image Visualization

```
warnings.filterwarnings("ignore")

images = [train_gen[0][0][i] for i in range(16)]
fig, axes = plt.subplots(3, 5, figsize = (10, 10))

axes = axes.flatten()

for img, ax in zip(images, axes):
    ax.imshow(img.reshape(image_size, image_size, 3))
    ax.axis('off')

plt.tight_layout()
plt.show()
```



Building the Model

```
!pip install vit_keras
```

```
Collecting vit_keras
  Downloading vit_keras-0.1.2-py3-none-any.whl (24 kB)
Collecting validators
  Downloading validators-0.20.0.tar.gz (30 kB)
  Preparing metadata (setup.py) ... ent already satisfied: scipy in
/opt/conda/lib/python3.7/site-packages (from vit_keras) (1.7.3)
Requirement already satisfied: numpy<1.23.0,>=1.16.5 in
/opt/conda/lib/python3.7/site-packages (from scipy->vit_keras)
(1.21.6)
Requirement already satisfied: decorator>=3.4.0 in
/opt/conda/lib/python3.7/site-packages (from validators->vit_keras)
(5.1.1)
Building wheels for collected packages: validators
  Building wheel for validators (setup.py) ... e=validators-0.20.0-
```

```
py3-none-any.whl size=19582
sha256=6cff2e8546df2ee65d3ed6aaa9d25eba9b1f516c5704bcd6f12109e21935e71
1
  Stored in directory:
/root/.cache/pip/wheels/5f/55/ab/36a76989f7f88d9ca7b1f68da6d94252bb6a8
d6ad4f18e04e9
Successfully built validators
Installing collected packages: validators, vit_keras
Successfully installed validators-0.20.0 vit_keras-0.1.2
WARNING: Running pip as the 'root' user can result in broken
permissions and conflicting behaviour with the system package manager.
It is recommended to use a virtual environment instead:
https://pip.pypa.io/warnings/venv
```

1 - ViT B16 Model

```
from vit_keras import vit

vit_model = vit.vit_b16(
        image_size = image_size,
        activation = 'softmax',
        pretrained = True,
        include_top = False,
        pretrained_top = False,
        classes = 4)
```

```
Downloading data from
https://github.com/faustomorales/vit-keras/releases/download/dl/ViT-
B_16_imagenet21k+imagenet2012.npz
347504640/347502902 [==============================] - 3s 0us/step
347512832/347502902 [==============================] - 3s 0us/step
```

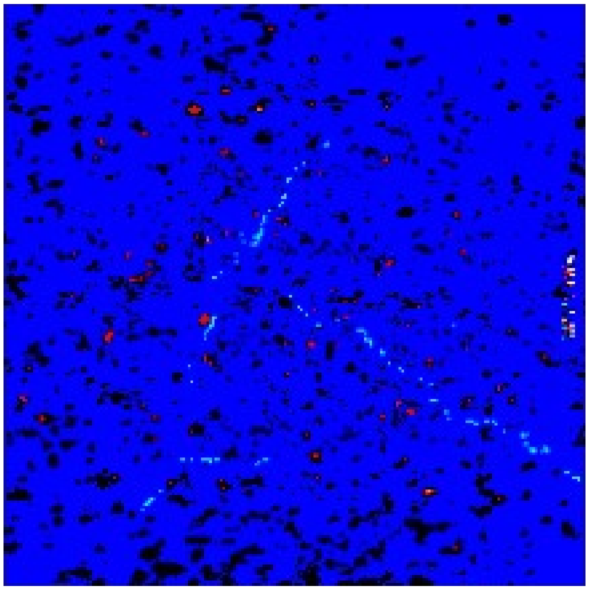
```
class Patches(L.Layer):
    def __init__(self, patch_size):
        super(Patches, self).__init__()
        self.patch_size = patch_size

    def call(self, images):
        batch_size = tf.shape(images)[0]
        patches = tf.image.extract_patches(
            images = images,
            sizes = [1, self.patch_size, self.patch_size, 1],
            strides = [1, self.patch_size, self.patch_size, 1],
            rates = [1, 1, 1, 1],
            padding = 'VALID',
        )
        patch_dims = patches.shape[-1]
```

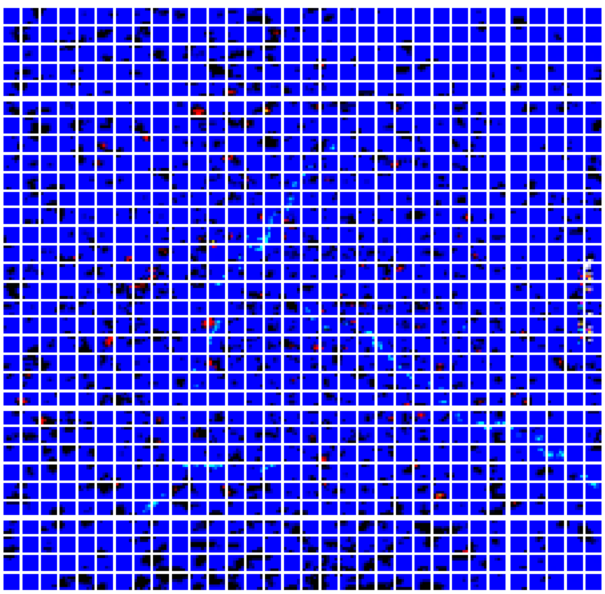
```
        patches = tf.reshape(patches, [batch_size, -1, patch_dims])
        return patches
```

Visualizing Attention Maps of Sample Test Image

```
plt.figure(figsize=(4, 4))
batch_size = 16
patch_size = 7  # Size of the patches to be extract from the input
images
num_patches = (image_size // patch_size) ** 2

x = train_gen.next()
image = x[0][0]

plt.imshow(image.astype('uint8'))
plt.axis('off')

resized_image = tf.image.resize(
    tf.convert_to_tensor([image]), size = (image_size, image_size)
)

patches = Patches(patch_size)(resized_image)
print(f'Image size: {image_size} X {image_size}')
print(f'Patch size: {patch_size} X {patch_size}')
print(f'Patches per image: {patches.shape[1]}')
print(f'Elements per patch: {patches.shape[-1]}')

n = int(np.sqrt(patches.shape[1]))
plt.figure(figsize=(4, 4))

for i, patch in enumerate(patches[0]):
    ax = plt.subplot(n, n, i + 1)
    patch_img = tf.reshape(patch, (patch_size, patch_size, 3))
    plt.imshow(patch_img.numpy().astype('uint8'))
    plt.axis('off')
```

```
Image size: 224 X 224
Patch size: 7 X 7
Patches per image: 1024
Elements per patch: 147
```





ViT Model Architecture

```
model = tf.keras.Sequential([
        vit_model,
        tf.keras.layers.Flatten(),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(128, activation = tfa.activations.gelu),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(64, activation = tfa.activations.gelu),
        tf.keras.layers.Dense(32, activation = tfa.activations.gelu),
        tf.keras.layers.Dense(4, 'softmax')
    ],
```

```
    name = 'vision_transformer')

model.summary()
```

```
Model: "vision_transformer"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
vit-b16 (Functional)         (None, 768)               85798656  
_________________________________________________________________
flatten_1 (Flatten)          (None, 768)               0         
_________________________________________________________________
batch_normalization_2 (Batch (None, 768)               3072      
_________________________________________________________________
dense_4 (Dense)              (None, 128)               98432     
_________________________________________________________________
batch_normalization_3 (Batch (None, 128)               512       
_________________________________________________________________
dense_5 (Dense)              (None, 64)                8256      
_________________________________________________________________
dense_6 (Dense)              (None, 32)                2080      
_________________________________________________________________
dense_7 (Dense)              (None, 4)                 132       
=================================================================
Total params: 85,911,140
Trainable params: 85,909,348
Non-trainable params: 1,792
_________________________________________________________________
```

Training the Model

```
warnings.filterwarnings("ignore")

learning_rate = 1e-4

optimizer = tfa.optimizers.RectifiedAdam(learning_rate = 
learning_rate)

model.compile(optimizer = optimizer, 
              loss = 
tf.keras.losses.CategoricalCrossentropy(label_smoothing = 0.2), 
              metrics = ['accuracy'])

STEP_SIZE_TRAIN = train_gen.n // train_gen.batch_size
STEP_SIZE_VALID = valid_gen.n // valid_gen.batch_size



early_stopping_callbacks = tf.keras.callbacks.EarlyStopping(patience = 
15, restore_best_weights = True, verbose = 1)
```

```
model.fit(x = train_gen,
          steps_per_epoch = STEP_SIZE_TRAIN,
          validation_data = valid_gen,
          validation_steps = STEP_SIZE_VALID,
          epochs = EPOCHS,
          callbacks = early_stopping_callbacks)
```

```
Epoch 1/30
115/115 [==============================] - 128s 756ms/step - loss:
1.2583 - accuracy: 0.5095 - val_loss: 1.1746 - val_accuracy: 0.5603
Epoch 2/30
115/115 [==============================] - 70s 608ms/step - loss:
1.0260 - accuracy: 0.7009 - val_loss: 1.1565 - val_accuracy: 0.5603
Epoch 3/30
115/115 [==============================] - 70s 606ms/step - loss:
0.9324 - accuracy: 0.7781 - val_loss: 1.1931 - val_accuracy: 0.5804
Epoch 4/30
115/115 [==============================] - 70s 605ms/step - loss:
0.8840 - accuracy: 0.8140 - val_loss: 1.0552 - val_accuracy: 0.6763
Epoch 5/30
115/115 [==============================] - 70s 612ms/step - loss:
0.8508 - accuracy: 0.8287 - val_loss: 1.0878 - val_accuracy: 0.6607
Epoch 6/30
115/115 [==============================] - 70s 605ms/step - loss:
0.8329 - accuracy: 0.8407 - val_loss: 0.9854 - val_accuracy: 0.7455
Epoch 7/30
115/115 [==============================] - 70s 608ms/step - loss:
0.8041 - accuracy: 0.8673 - val_loss: 0.9169 - val_accuracy: 0.7835
Epoch 8/30
115/115 [==============================] - 70s 606ms/step - loss:
0.7928 - accuracy: 0.8635 - val_loss: 0.9503 - val_accuracy: 0.7545
Epoch 9/30
115/115 [==============================] - 70s 608ms/step - loss:
0.7742 - accuracy: 0.8755 - val_loss: 0.9459 - val_accuracy: 0.7612
Epoch 10/30
115/115 [==============================] - 69s 602ms/step - loss:
0.7581 - accuracy: 0.8880 - val_loss: 1.0175 - val_accuracy: 0.7299
Epoch 11/30
115/115 [==============================] - 70s 604ms/step - loss:
0.7591 - accuracy: 0.8907 - val_loss: 0.9173 - val_accuracy: 0.7812
Epoch 12/30
115/115 [==============================] - 70s 606ms/step - loss:
0.7466 - accuracy: 0.8978 - val_loss: 0.9606 - val_accuracy: 0.7746
Epoch 13/30
115/115 [==============================] - 70s 611ms/step - loss:
0.7259 - accuracy: 0.9125 - val_loss: 0.9557 - val_accuracy: 0.7567
Epoch 14/30
115/115 [==============================] - 70s 609ms/step - loss:
0.7141 - accuracy: 0.9233 - val_loss: 0.9600 - val_accuracy: 0.7589
```

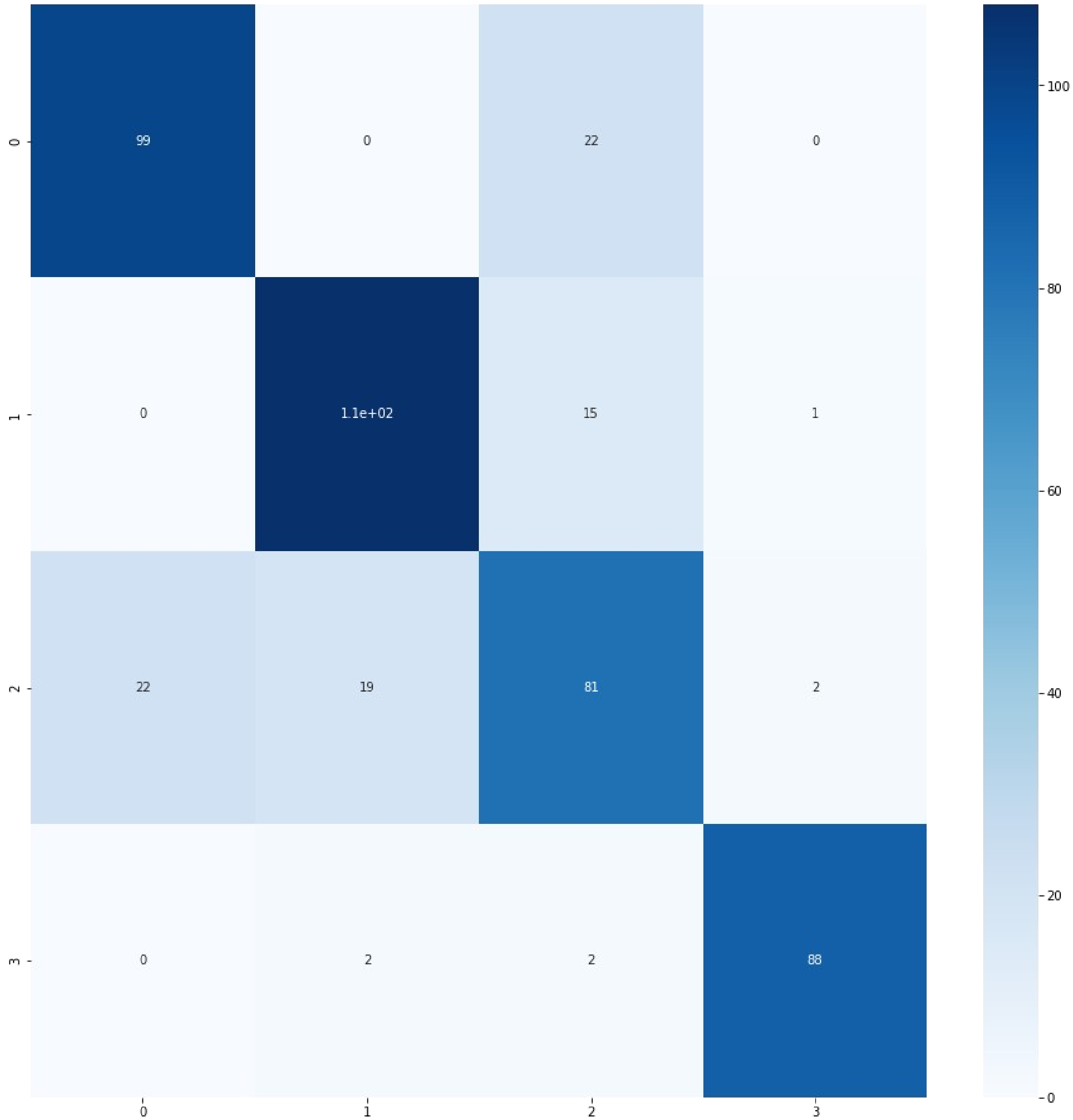
```
Epoch 15/30
115/115 [==============================] - 70s 607ms/step - loss: 
0.7379 - accuracy: 0.8972 - val_loss: 0.9464 - val_accuracy: 0.7545
Epoch 16/30
115/115 [==============================] - 70s 607ms/step - loss: 
0.7098 - accuracy: 0.9233 - val_loss: 0.9352 - val_accuracy: 0.7790
Epoch 17/30
115/115 [==============================] - 70s 606ms/step - loss: 
0.7079 - accuracy: 0.9239 - val_loss: 0.8557 - val_accuracy: 0.8527
Epoch 18/30
115/115 [==============================] - 70s 611ms/step - loss: 
0.7025 - accuracy: 0.9282 - val_loss: 0.8681 - val_accuracy: 0.8103
Epoch 19/30
115/115 [==============================] - 70s 608ms/step - loss: 
0.6855 - accuracy: 0.9456 - val_loss: 0.8968 - val_accuracy: 0.8036
Epoch 20/30
115/115 [==============================] - 70s 608ms/step - loss: 
0.6887 - accuracy: 0.9380 - val_loss: 0.8516 - val_accuracy: 0.8237
Epoch 21/30
115/115 [==============================] - 70s 606ms/step - loss: 
0.6756 - accuracy: 0.9516 - val_loss: 0.9013 - val_accuracy: 0.8170
Epoch 22/30
115/115 [==============================] - 70s 611ms/step - loss: 
0.6692 - accuracy: 0.9565 - val_loss: 0.9370 - val_accuracy: 0.8036
Epoch 23/30
115/115 [==============================] - 70s 606ms/step - loss: 
0.6760 - accuracy: 0.9489 - val_loss: 0.8954 - val_accuracy: 0.8103
Epoch 24/30
115/115 [==============================] - 70s 606ms/step - loss: 
0.6772 - accuracy: 0.9511 - val_loss: 0.9165 - val_accuracy: 0.7991
Epoch 25/30
115/115 [==============================] - 70s 605ms/step - loss: 
0.6718 - accuracy: 0.9511 - val_loss: 0.9096 - val_accuracy: 0.8214
Epoch 26/30
115/115 [==============================] - 70s 610ms/step - loss: 
0.6802 - accuracy: 0.9424 - val_loss: 0.9591 - val_accuracy: 0.7679
Epoch 27/30
115/115 [==============================] - 70s 605ms/step - loss: 
0.6620 - accuracy: 0.9608 - val_loss: 0.9634 - val_accuracy: 0.7612
Epoch 28/30
115/115 [==============================] - 70s 607ms/step - loss: 
0.6813 - accuracy: 0.9451 - val_loss: 0.9122 - val_accuracy: 0.8013
Epoch 29/30
115/115 [==============================] - 69s 602ms/step - loss: 
0.6687 - accuracy: 0.9521 - val_loss: 1.0262 - val_accuracy: 0.7254
Epoch 30/30
115/115 [==============================] - 70s 608ms/step - loss: 
0.6601 - accuracy: 0.9581 - val_loss: 0.8990 - val_accuracy: 0.8103

<keras.callbacks.History at 0x7aa797b0af50>
```

```python
# Save The Model

model.save('ViT_model.h5')
```

ViT Model Result

```python
predicted_classes = np.argmax(model.predict(valid_gen, steps =
valid_gen.n // valid_gen.batch_size + 1), axis = 1)
true_classes = valid_gen.classes
class_labels = list(valid_gen.class_indices.keys())

confusionmatrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize = (16, 16))
sns.heatmap(confusionmatrix, cmap = 'Blues', annot = True, cbar =
True)

print(classification_report(true_classes, predicted_classes))
```

```
              precision    recall  f1-score   support

           0       0.82      0.82      0.82       121
           1       0.84      0.87      0.85       124
           2       0.68      0.65      0.66       124
           3       0.97      0.96      0.96        92

    accuracy                           0.82       461
   macro avg       0.82      0.82      0.82       461
weighted avg       0.81      0.82      0.81       461
```




```
model.evaluate(valid_gen)

29/29 [==============================] - 5s 156ms/step - loss: 0.8906
- accuracy: 0.8156

[0.8906439542770386, 0.8156182169914246]
```

2 - ResNet50

```
model2 = tf.keras.Sequential()

base1= tf.keras.applications.ResNet50V2(include_top=False,
                   input_shape=(224,224,3),
                   pooling='avg',classes=4,
                   weights='imagenet')

model2.add(base1)

model2.add(tf.keras.layers.Flatten())
model2.add(tf.keras.layers.BatchNormalization())
model2.add(tf.keras.layers.Dense(128, activation =
tfa.activations.gelu))
model2.add(tf.keras.layers.BatchNormalization())
model2.add(tf.keras.layers.Dense(64, activation =
tfa.activations.gelu))
model2.add(tf.keras.layers.Dropout(0.5))
model2.add(tf.keras.layers.Dense(32, activation =
tfa.activations.gelu))
model2.add(tf.keras.layers.Dense(4, 'softmax'))

for layer in base1.layers:
    layer.trainable = False

model2.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
resnet50v2 (Functional)      (None, 2048)              23564800
_________________________________________________________________
flatten_6 (Flatten)          (None, 2048)              0
_________________________________________________________________
batch_normalization_12 (Batc (None, 2048)              8192
_________________________________________________________________
dense_24 (Dense)             (None, 128)               262272
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512
_________________________________________________________________
dense_25 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_26 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_27 (Dense)             (None, 4)                 132
=================================================================
```

```
Total params: 23,846,244
Trainable params: 277,092
Non-trainable params: 23,569,152
_________________________________________________________________
```

Training The Model

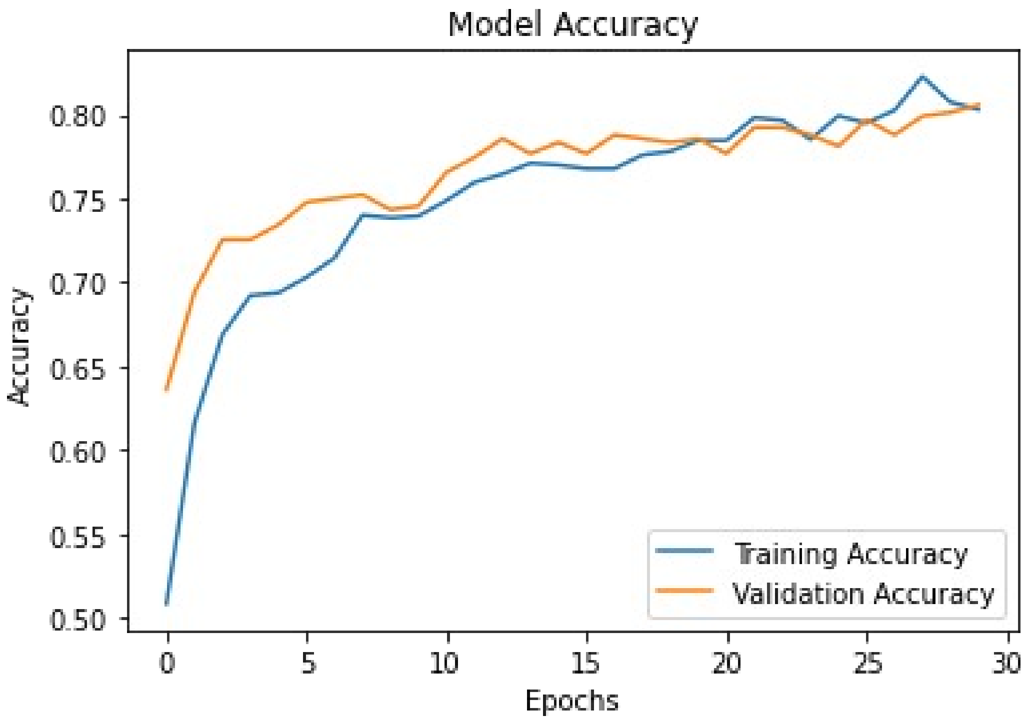
```
model2.compile(optimizer = optimizer,
              loss = 
tf.keras.losses.CategoricalCrossentropy(label_smoothing = 0.2),
              metrics = ['accuracy'])

history = model2.fit(x = train_gen,
          steps_per_epoch = STEP_SIZE_TRAIN,
          validation_data = valid_gen,
          validation_steps = STEP_SIZE_VALID,
          epochs = EPOCHS,
          callbacks = early_stopping_callbacks)

Epoch 1/30
115/115 [==============================] - 36s 281ms/step - loss: 
1.2918 - accuracy: 0.5084 - val_loss: 1.1641 - val_accuracy: 0.6362
Epoch 2/30
115/115 [==============================] - 31s 271ms/step - loss: 
1.1255 - accuracy: 0.6161 - val_loss: 1.0899 - val_accuracy: 0.6942
Epoch 3/30
115/115 [==============================] - 32s 275ms/step - loss: 
1.0732 - accuracy: 0.6688 - val_loss: 1.0489 - val_accuracy: 0.7254
Epoch 4/30
115/115 [==============================] - 32s 278ms/step - loss: 
1.0472 - accuracy: 0.6922 - val_loss: 1.0374 - val_accuracy: 0.7254
Epoch 5/30
115/115 [==============================] - 32s 276ms/step - loss: 
1.0391 - accuracy: 0.6939 - val_loss: 1.0098 - val_accuracy: 0.7344
Epoch 6/30
115/115 [==============================] - 32s 276ms/step - loss: 
1.0299 - accuracy: 0.7031 - val_loss: 0.9939 - val_accuracy: 0.7478
Epoch 7/30
115/115 [==============================] - 31s 271ms/step - loss: 
1.0121 - accuracy: 0.7145 - val_loss: 0.9845 - val_accuracy: 0.7500
Epoch 8/30
115/115 [==============================] - 31s 272ms/step - loss: 
0.9870 - accuracy: 0.7401 - val_loss: 0.9793 - val_accuracy: 0.7522
Epoch 9/30
115/115 [==============================] - 32s 274ms/step - loss: 
0.9893 - accuracy: 0.7384 - val_loss: 0.9648 - val_accuracy: 0.7433
Epoch 10/30
115/115 [==============================] - 32s 276ms/step - loss: 
0.9786 - accuracy: 0.7395 - val_loss: 0.9772 - val_accuracy: 0.7455
```

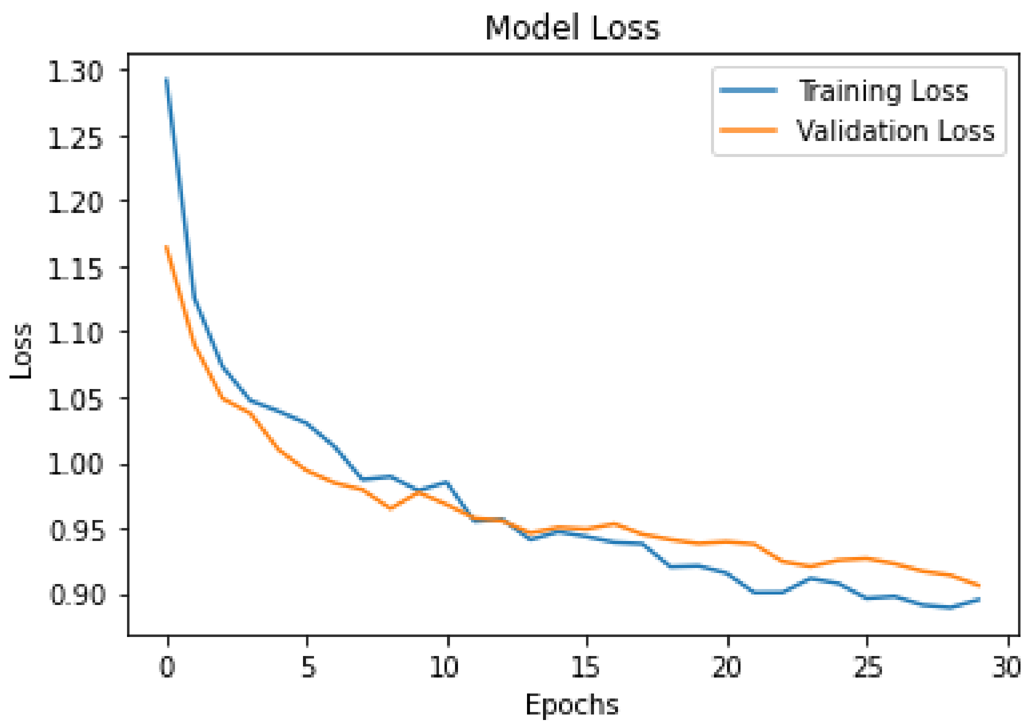
```
Epoch 11/30
115/115 [==============================] - 32s 275ms/step - loss:
0.9850 - accuracy: 0.7488 - val_loss: 0.9681 - val_accuracy: 0.7656
Epoch 12/30
115/115 [==============================] - 31s 273ms/step - loss:
0.9552 - accuracy: 0.7597 - val_loss: 0.9575 - val_accuracy: 0.7746
Epoch 13/30
115/115 [==============================] - 32s 275ms/step - loss:
0.9566 - accuracy: 0.7645 - val_loss: 0.9555 - val_accuracy: 0.7857
Epoch 14/30
115/115 [==============================] - 31s 273ms/step - loss:
0.9411 - accuracy: 0.7711 - val_loss: 0.9462 - val_accuracy: 0.7768
Epoch 15/30
115/115 [==============================] - 32s 274ms/step - loss:
0.9472 - accuracy: 0.7700 - val_loss: 0.9505 - val_accuracy: 0.7835
Epoch 16/30
115/115 [==============================] - 31s 273ms/step - loss:
0.9435 - accuracy: 0.7678 - val_loss: 0.9492 - val_accuracy: 0.7768
Epoch 17/30
115/115 [==============================] - 31s 272ms/step - loss:
0.9391 - accuracy: 0.7678 - val_loss: 0.9532 - val_accuracy: 0.7879
Epoch 18/30
115/115 [==============================] - 32s 274ms/step - loss:
0.9380 - accuracy: 0.7760 - val_loss: 0.9452 - val_accuracy: 0.7857
Epoch 19/30
115/115 [==============================] - 31s 273ms/step - loss:
0.9204 - accuracy: 0.7781 - val_loss: 0.9412 - val_accuracy: 0.7835
Epoch 20/30
115/115 [==============================] - 31s 273ms/step - loss:
0.9211 - accuracy: 0.7847 - val_loss: 0.9382 - val_accuracy: 0.7857
Epoch 21/30
115/115 [==============================] - 31s 273ms/step - loss:
0.9156 - accuracy: 0.7847 - val_loss: 0.9396 - val_accuracy: 0.7768
Epoch 22/30
115/115 [==============================] - 31s 270ms/step - loss:
0.9006 - accuracy: 0.7983 - val_loss: 0.9377 - val_accuracy: 0.7924
Epoch 23/30
115/115 [==============================] - 31s 273ms/step - loss:
0.9007 - accuracy: 0.7966 - val_loss: 0.9245 - val_accuracy: 0.7924
Epoch 24/30
115/115 [==============================] - 31s 272ms/step - loss:
0.9115 - accuracy: 0.7852 - val_loss: 0.9207 - val_accuracy: 0.7879
Epoch 25/30
115/115 [==============================] - 31s 270ms/step - loss:
0.9077 - accuracy: 0.7993 - val_loss: 0.9254 - val_accuracy: 0.7812
Epoch 26/30
115/115 [==============================] - 31s 272ms/step - loss:
0.8963 - accuracy: 0.7950 - val_loss: 0.9267 - val_accuracy: 0.7969
Epoch 27/30
```

```
115/115 [==============================] - 31s 270ms/step - loss:
0.8978 - accuracy: 0.8026 - val_loss: 0.9228 - val_accuracy: 0.7879
Epoch 28/30
115/115 [==============================] - 31s 270ms/step - loss:
0.8913 - accuracy: 0.8227 - val_loss: 0.9170 - val_accuracy: 0.7991
Epoch 29/30
115/115 [==============================] - 32s 275ms/step - loss:
0.8892 - accuracy: 0.8075 - val_loss: 0.9139 - val_accuracy: 0.8013
Epoch 30/30
115/115 [==============================] - 31s 271ms/step - loss:
0.8953 - accuracy: 0.8032 - val_loss: 0.9062 - val_accuracy: 0.8058
```

Model Result

```python
import matplotlib.pyplot as plt

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

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

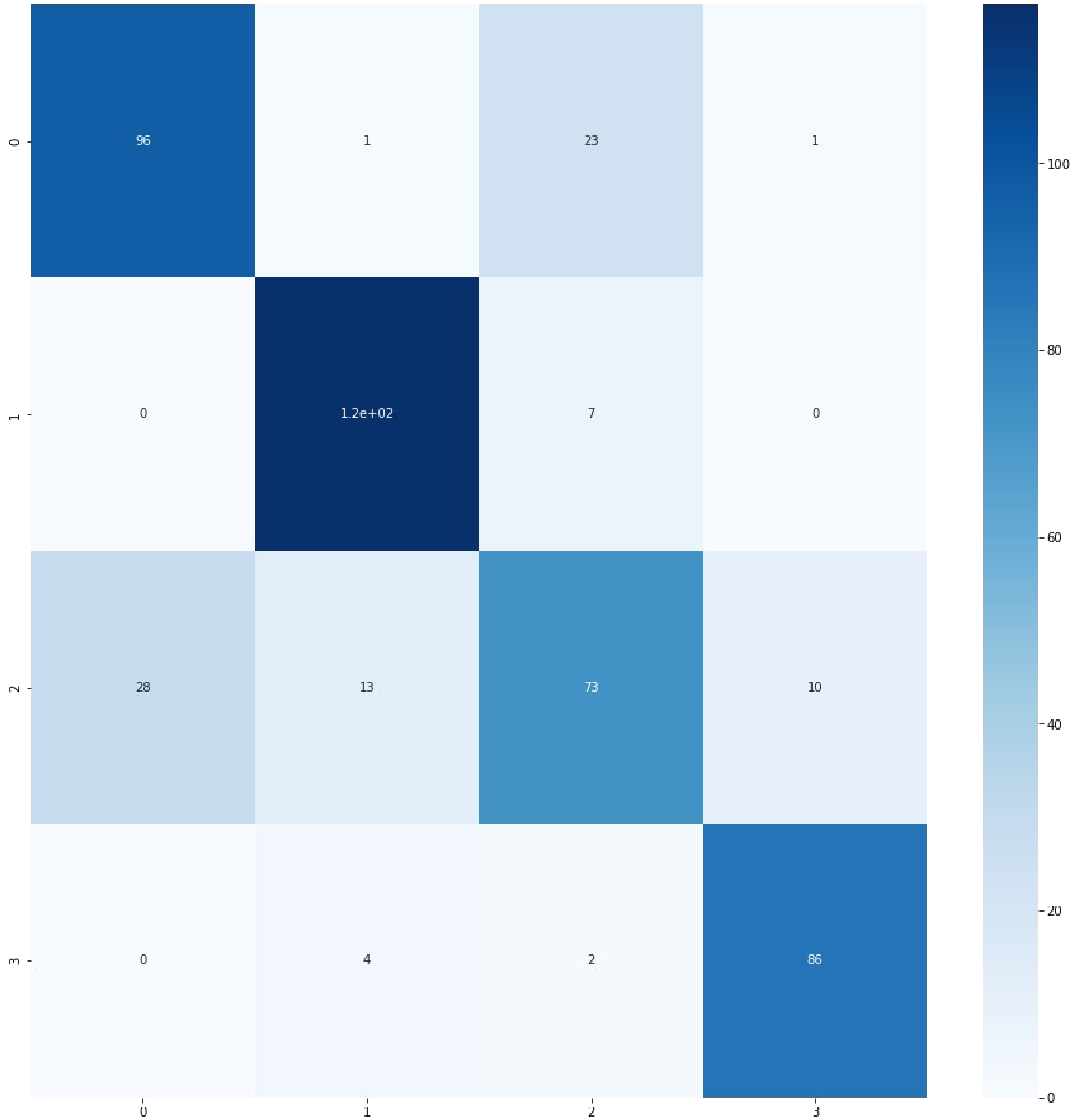





```
predicted_classes = np.argmax(model2.predict(valid_gen, steps =
valid_gen.n // valid_gen.batch_size + 1), axis = 1)
true_classes = valid_gen.classes
class_labels = list(valid_gen.class_indices.keys())
```

```python
confusionmatrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize = (16, 16))
sns.heatmap(confusionmatrix, cmap = 'Blues', annot = True, cbar =
True)

print(classification_report(true_classes, predicted_classes))
```

```
              precision    recall  f1-score   support

           0       0.77      0.79      0.78       121
           1       0.87      0.94      0.90       124
           2       0.70      0.59      0.64       124
           3       0.89      0.93      0.91        92

    accuracy                           0.81       461
   macro avg       0.81      0.82      0.81       461
weighted avg       0.80      0.81      0.80       461
```



3 - VGG19

```
model3 = tf.keras.Sequential()

base2= tf.keras.applications.VGG19(include_top=False,
                   input_shape=(224,224,3),
                   pooling='avg',classes=4,
                   weights='imagenet')
```

```
model3.add(base2)

model3.add(tf.keras.layers.Flatten())
model3.add(tf.keras.layers.BatchNormalization())
model3.add(tf.keras.layers.Dense(128, activation =
tfa.activations.gelu))
model3.add(tf.keras.layers.BatchNormalization())
model3.add(tf.keras.layers.Dense(64, activation =
tfa.activations.gelu))
model3.add(tf.keras.layers.Dropout(0.5))
model3.add(tf.keras.layers.Dense(32, activation =
tfa.activations.gelu))
model3.add(tf.keras.layers.Dense(4, 'softmax'))

for layer in base2.layers:
    layer.trainable = False

model3.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg19 (Functional)           (None, 512)               20024384
_________________________________________________________________
flatten_7 (Flatten)          (None, 512)               0
_________________________________________________________________
batch_normalization_14 (Batc (None, 512)               2048
_________________________________________________________________
dense_28 (Dense)             (None, 128)               65664
_________________________________________________________________
batch_normalization_15 (Batc (None, 128)               512
_________________________________________________________________
dense_29 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_30 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_31 (Dense)             (None, 4)                 132
=================================================================
Total params: 20,103,076
Trainable params: 77,412
Non-trainable params: 20,025,664
_________________________________________________________________
```

Training The Model

```
model3.compile(optimizer = optimizer,
              loss =
```

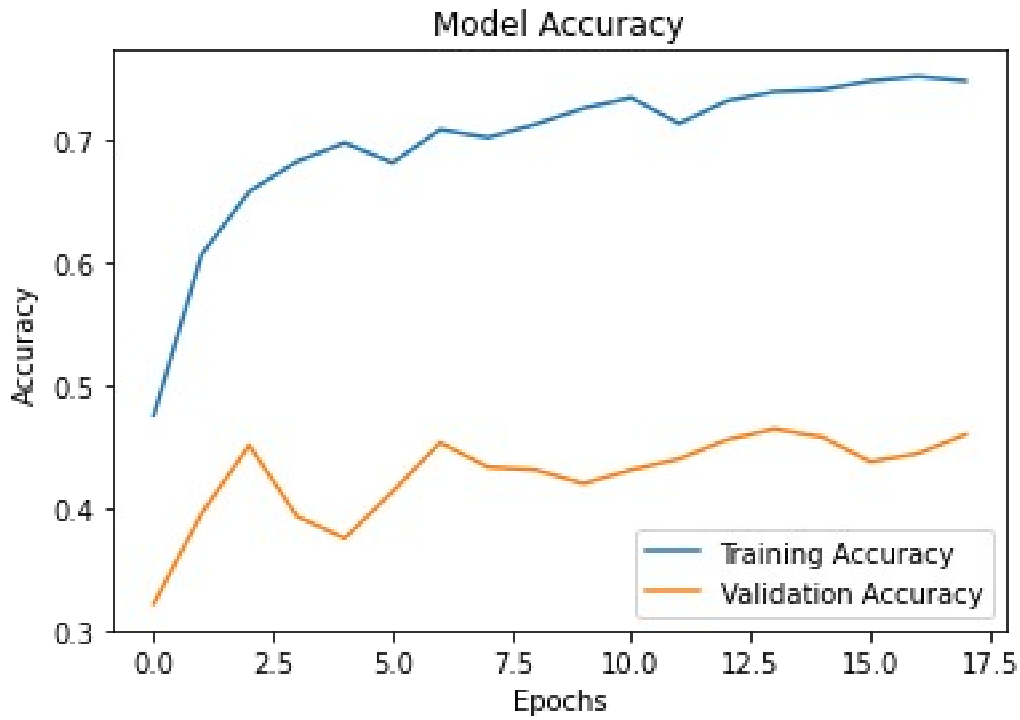
```
tf.keras.losses.CategoricalCrossentropy(label_smoothing = 0.2),
              metrics = ['accuracy'])

history = model3.fit(x = train_gen,
          steps_per_epoch = STEP_SIZE_TRAIN,
          validation_data = valid_gen,
          validation_steps = STEP_SIZE_VALID,
          epochs = EPOCHS,
          callbacks = early_stopping_callbacks)
```

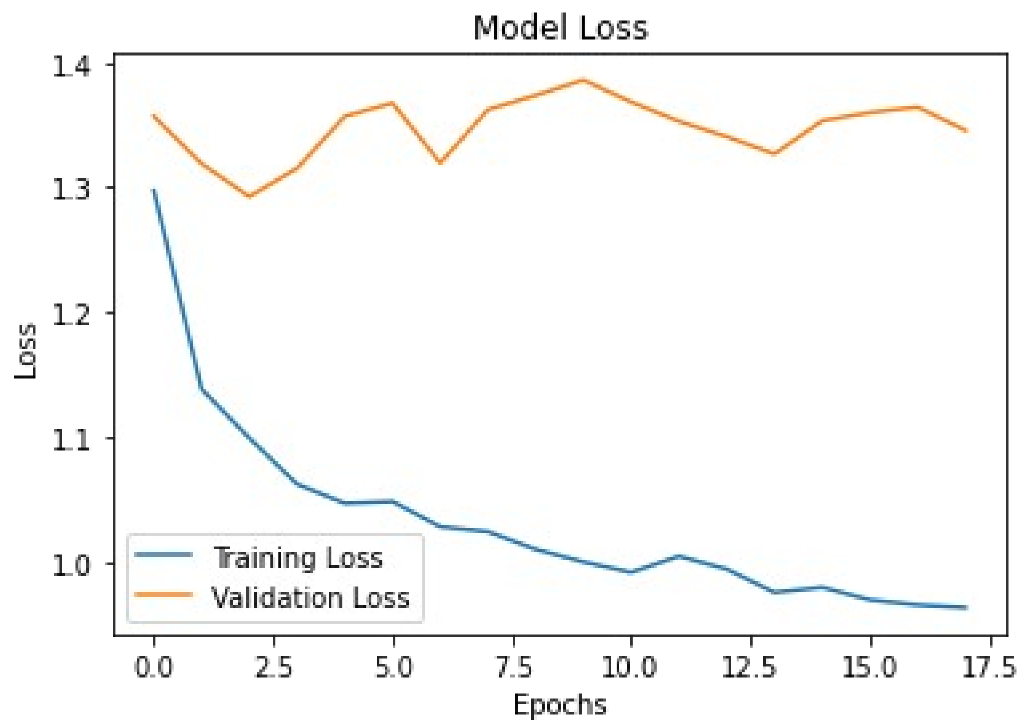
```
Epoch 1/30
115/115 [==============================] - 37s 298ms/step - loss:
1.2975 - accuracy: 0.4753 - val_loss: 1.3571 - val_accuracy: 0.3214
Epoch 2/30
115/115 [==============================] - 34s 294ms/step - loss:
1.1385 - accuracy: 0.6063 - val_loss: 1.3191 - val_accuracy: 0.3951
Epoch 3/30
115/115 [==============================] - 34s 292ms/step - loss:
1.0992 - accuracy: 0.6580 - val_loss: 1.2927 - val_accuracy: 0.4509
Epoch 4/30
115/115 [==============================] - 33s 289ms/step - loss:
1.0624 - accuracy: 0.6824 - val_loss: 1.3154 - val_accuracy: 0.3929
Epoch 5/30
115/115 [==============================] - 33s 288ms/step - loss:
1.0470 - accuracy: 0.6977 - val_loss: 1.3571 - val_accuracy: 0.3750
Epoch 6/30
115/115 [==============================] - 33s 291ms/step - loss:
1.0485 - accuracy: 0.6813 - val_loss: 1.3676 - val_accuracy: 0.4129
Epoch 7/30
115/115 [==============================] - 34s 292ms/step - loss:
1.0282 - accuracy: 0.7085 - val_loss: 1.3196 - val_accuracy: 0.4531
Epoch 8/30
115/115 [==============================] - 33s 287ms/step - loss:
1.0243 - accuracy: 0.7020 - val_loss: 1.3624 - val_accuracy: 0.4330
Epoch 9/30
115/115 [==============================] - 33s 291ms/step - loss:
1.0101 - accuracy: 0.7129 - val_loss: 1.3739 - val_accuracy: 0.4308
Epoch 10/30
115/115 [==============================] - 33s 289ms/step - loss:
1.0001 - accuracy: 0.7259 - val_loss: 1.3860 - val_accuracy: 0.4196
Epoch 11/30
115/115 [==============================] - 34s 292ms/step - loss:
0.9917 - accuracy: 0.7346 - val_loss: 1.3685 - val_accuracy: 0.4308
Epoch 12/30
115/115 [==============================] - 33s 287ms/step - loss:
1.0046 - accuracy: 0.7134 - val_loss: 1.3531 - val_accuracy: 0.4397
Epoch 13/30
115/115 [==============================] - 34s 291ms/step - loss:
0.9943 - accuracy: 0.7319 - val_loss: 1.3407 - val_accuracy: 0.4554
Epoch 14/30
```

```
115/115 [==============================] - 33s 290ms/step - loss:
0.9756 - accuracy: 0.7395 - val_loss: 1.3269 - val_accuracy: 0.4643
Epoch 15/30
115/115 [==============================] - 34s 293ms/step - loss:
0.9797 - accuracy: 0.7412 - val_loss: 1.3535 - val_accuracy: 0.4576
Epoch 16/30
115/115 [==============================] - 33s 287ms/step - loss:
0.9696 - accuracy: 0.7482 - val_loss: 1.3601 - val_accuracy: 0.4375
Epoch 17/30
115/115 [==============================] - 34s 292ms/step - loss:
0.9656 - accuracy: 0.7520 - val_loss: 1.3644 - val_accuracy: 0.4442
Epoch 18/30
115/115 [==============================] - 33s 289ms/step - loss:
0.9634 - accuracy: 0.7482 - val_loss: 1.3458 - val_accuracy: 0.4598
Restoring model weights from the end of the best epoch.
Epoch 00018: early stopping
```

Model Result

```python
import matplotlib.pyplot as plt

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

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


Model Accuracy
Accuracy
Epochs
Training Accuracy
Validation Accuracy
0.3
0.4
0.5
0.6
0.7
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5


Model Loss
Loss
Epochs
Training Loss
Validation Loss
1.0
1.1
1.2
1.3
1.4
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

4 - InceptionV3

```
model4 = tf.keras.Sequential()

base3=
tf.keras.applications.inception_v3.InceptionV3(include_top=False,
                   input_shape=(224,224,3),
                   pooling='avg',
                   classes=4,
                   weights='imagenet')

model4.add(base3)

model4.add(tf.keras.layers.Flatten())
model4.add(tf.keras.layers.BatchNormalization())
model4.add(tf.keras.layers.Dense(128, activation =
tfa.activations.gelu))
model4.add(tf.keras.layers.BatchNormalization())
model4.add(tf.keras.layers.Dense(64, activation =
tfa.activations.gelu))
model4.add(tf.keras.layers.Dropout(0.5))
model4.add(tf.keras.layers.Dense(32, activation =
tfa.activations.gelu))
model4.add(tf.keras.layers.Dense(4, 'softmax'))

for layer in base3.layers:
    layer.trainable = False

model4.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 2048)              21802784
_________________________________________________________________
flatten_9 (Flatten)          (None, 2048)              0
_________________________________________________________________
batch_normalization_206 (Bat (None, 2048)              8192
_________________________________________________________________
dense_36 (Dense)             (None, 128)               262272
_________________________________________________________________
batch_normalization_207 (Bat (None, 128)               512
_________________________________________________________________
dense_37 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_38 (Dense)             (None, 32)                2080
_________________________________________________________________
```

```
dense_39 (Dense)             (None, 4)                 132
=================================================================
Total params: 22,084,228
Trainable params: 277,092
Non-trainable params: 21,807,136
_________________________________________________________________
```

Training The Model

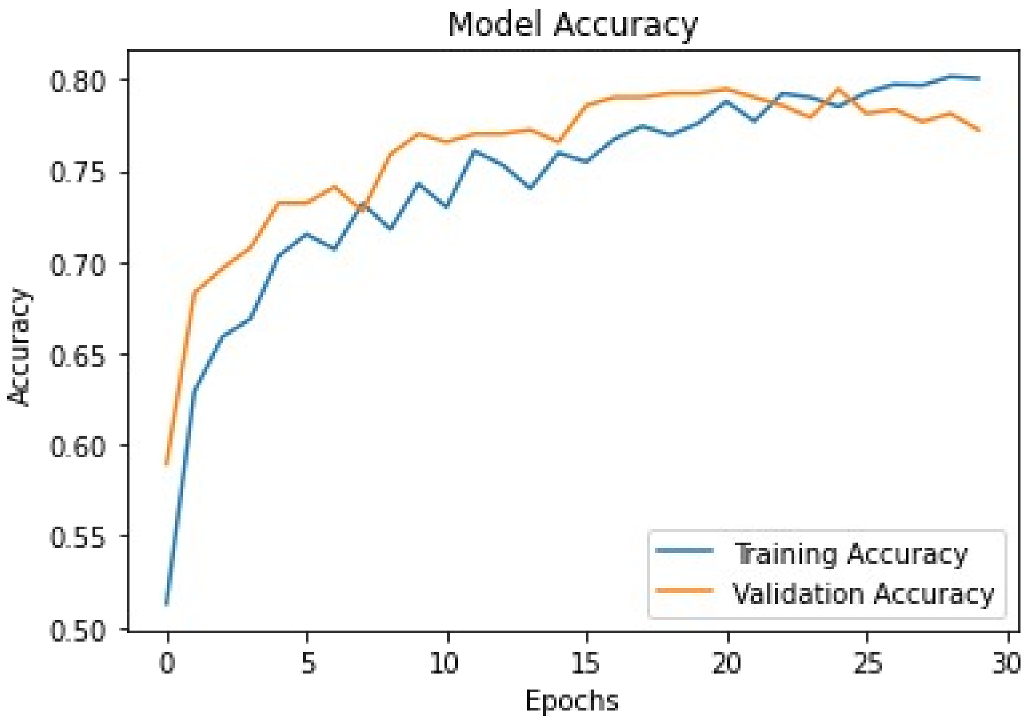
```
model4.compile(optimizer = optimizer,
              loss = 
tf.keras.losses.CategoricalCrossentropy(label_smoothing = 0.2),
              metrics = ['accuracy'])

history = model4.fit(x = train_gen,
          steps_per_epoch = STEP_SIZE_TRAIN,
          validation_data = valid_gen,
          validation_steps = STEP_SIZE_VALID,
          epochs = EPOCHS,
          callbacks = early_stopping_callbacks)

Epoch 1/30
115/115 [==============================] - 37s 276ms/step - loss: 
1.2980 - accuracy: 0.5128 - val_loss: 1.1558 - val_accuracy: 0.5893
Epoch 2/30
115/115 [==============================] - 31s 268ms/step - loss: 
1.1102 - accuracy: 0.6291 - val_loss: 1.0686 - val_accuracy: 0.6830
Epoch 3/30
115/115 [==============================] - 31s 273ms/step - loss: 
1.0781 - accuracy: 0.6591 - val_loss: 1.0270 - val_accuracy: 0.6964
Epoch 4/30
115/115 [==============================] - 31s 267ms/step - loss: 
1.0635 - accuracy: 0.6688 - val_loss: 1.0031 - val_accuracy: 0.7076
Epoch 5/30
115/115 [==============================] - 31s 269ms/step - loss: 
1.0283 - accuracy: 0.7031 - val_loss: 0.9739 - val_accuracy: 0.7321
Epoch 6/30
115/115 [==============================] - 31s 266ms/step - loss: 
1.0143 - accuracy: 0.7151 - val_loss: 0.9812 - val_accuracy: 0.7321
Epoch 7/30
115/115 [==============================] - 31s 268ms/step - loss: 
1.0105 - accuracy: 0.7069 - val_loss: 0.9713 - val_accuracy: 0.7411
Epoch 8/30
115/115 [==============================] - 30s 265ms/step - loss: 
0.9875 - accuracy: 0.7319 - val_loss: 0.9604 - val_accuracy: 0.7277
Epoch 9/30
115/115 [==============================] - 31s 268ms/step - loss: 
0.9854 - accuracy: 0.7178 - val_loss: 0.9477 - val_accuracy: 0.7589
Epoch 10/30
```

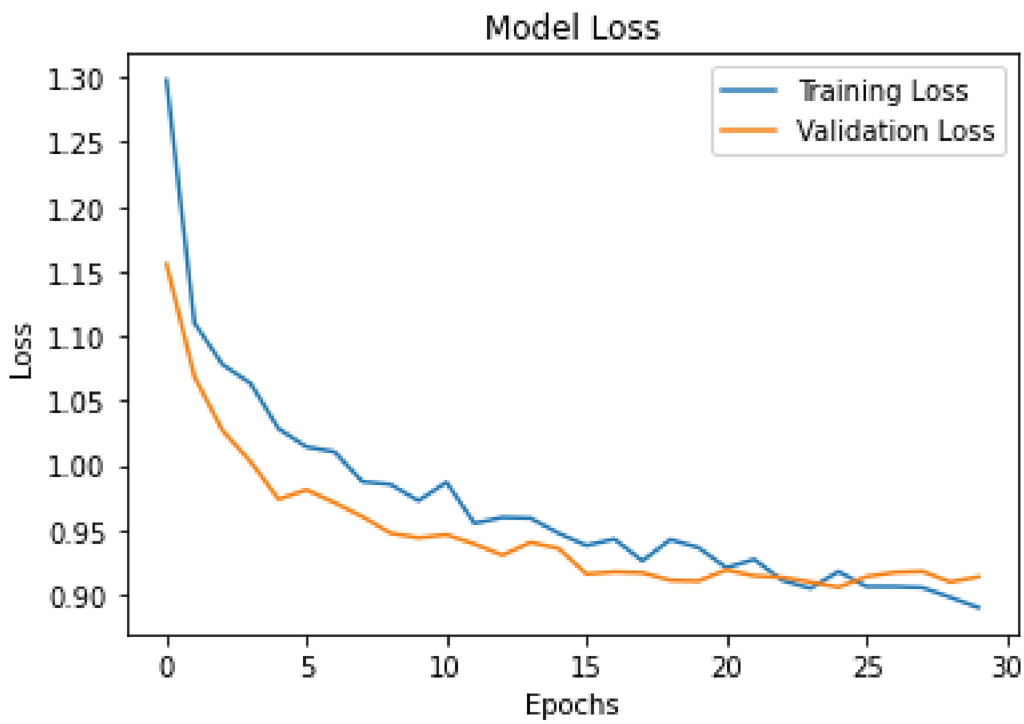
```
115/115 [==============================] - 30s 264ms/step - loss:
0.9728 - accuracy: 0.7428 - val_loss: 0.9441 - val_accuracy: 0.7701
Epoch 11/30
115/115 [==============================] - 31s 267ms/step - loss:
0.9871 - accuracy: 0.7299 - val_loss: 0.9465 - val_accuracy: 0.7656
Epoch 12/30
115/115 [==============================] - 31s 267ms/step - loss:
0.9553 - accuracy: 0.7607 - val_loss: 0.9393 - val_accuracy: 0.7701
Epoch 13/30
115/115 [==============================] - 30s 263ms/step - loss:
0.9599 - accuracy: 0.7531 - val_loss: 0.9306 - val_accuracy: 0.7701
Epoch 14/30
115/115 [==============================] - 30s 263ms/step - loss:
0.9593 - accuracy: 0.7402 - val_loss: 0.9406 - val_accuracy: 0.7723
Epoch 15/30
115/115 [==============================] - 30s 264ms/step - loss:
0.9477 - accuracy: 0.7597 - val_loss: 0.9360 - val_accuracy: 0.7656
Epoch 16/30
115/115 [==============================] - 31s 266ms/step - loss:
0.9381 - accuracy: 0.7548 - val_loss: 0.9161 - val_accuracy: 0.7857
Epoch 17/30
115/115 [==============================] - 30s 262ms/step - loss:
0.9431 - accuracy: 0.7673 - val_loss: 0.9177 - val_accuracy: 0.7902
Epoch 18/30
115/115 [==============================] - 31s 266ms/step - loss:
0.9262 - accuracy: 0.7743 - val_loss: 0.9170 - val_accuracy: 0.7902
Epoch 19/30
115/115 [==============================] - 30s 265ms/step - loss:
0.9427 - accuracy: 0.7694 - val_loss: 0.9112 - val_accuracy: 0.7924
Epoch 20/30
115/115 [==============================] - 31s 265ms/step - loss:
0.9366 - accuracy: 0.7760 - val_loss: 0.9107 - val_accuracy: 0.7924
Epoch 21/30
115/115 [==============================] - 31s 266ms/step - loss:
0.9209 - accuracy: 0.7879 - val_loss: 0.9193 - val_accuracy: 0.7946
Epoch 22/30
115/115 [==============================] - 31s 267ms/step - loss:
0.9274 - accuracy: 0.7771 - val_loss: 0.9146 - val_accuracy: 0.7902
Epoch 23/30
115/115 [==============================] - 30s 264ms/step - loss:
0.9111 - accuracy: 0.7923 - val_loss: 0.9135 - val_accuracy: 0.7857
Epoch 24/30
115/115 [==============================] - 31s 266ms/step - loss:
0.9050 - accuracy: 0.7901 - val_loss: 0.9098 - val_accuracy: 0.7790
Epoch 25/30
115/115 [==============================] - 30s 262ms/step - loss:
0.9179 - accuracy: 0.7852 - val_loss: 0.9059 - val_accuracy: 0.7946
Epoch 26/30
115/115 [==============================] - 30s 263ms/step - loss:
```

```
0.9064 - accuracy: 0.7928 - val_loss: 0.9140 - val_accuracy: 0.7812
Epoch 27/30
115/115 [==============================] - 30s 262ms/step - loss:
0.9064 - accuracy: 0.7972 - val_loss: 0.9173 - val_accuracy: 0.7835
Epoch 28/30
115/115 [==============================] - 30s 263ms/step - loss:
0.9056 - accuracy: 0.7966 - val_loss: 0.9181 - val_accuracy: 0.7768
Epoch 29/30
115/115 [==============================] - 30s 265ms/step - loss:
0.8980 - accuracy: 0.8015 - val_loss: 0.9099 - val_accuracy: 0.7812
Epoch 30/30
115/115 [==============================] - 30s 265ms/step - loss:
0.8901 - accuracy: 0.8004 - val_loss: 0.9140 - val_accuracy: 0.7723
```

Model Result

```python
import matplotlib.pyplot as plt

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

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

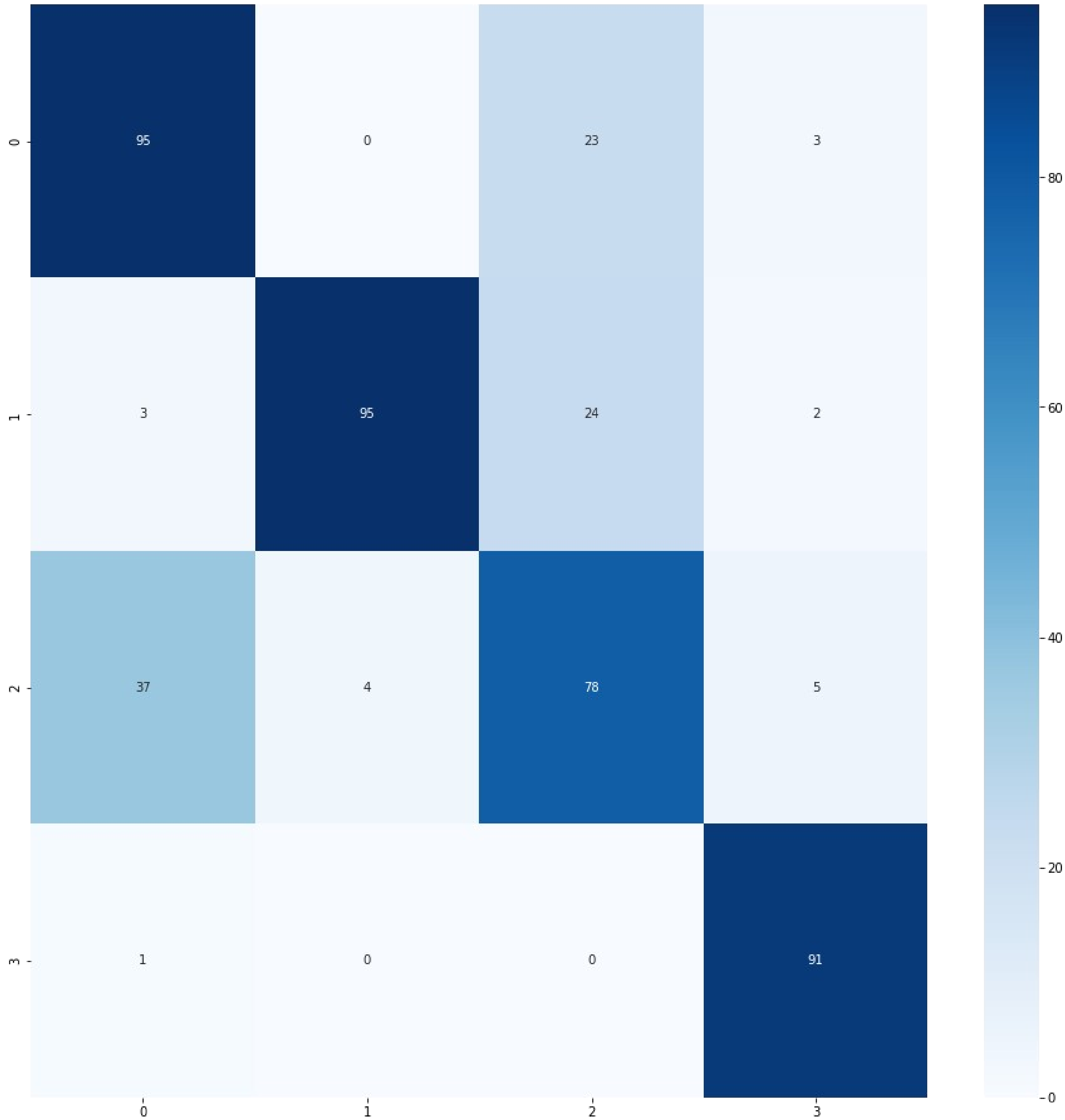





```
predicted_classes = np.argmax(model4.predict(valid_gen, steps =
valid_gen.n // valid_gen.batch_size + 1), axis = 1)
true_classes = valid_gen.classes
class_labels = list(valid_gen.class_indices.keys())
```

```
confusionmatrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize = (16, 16))
sns.heatmap(confusionmatrix, cmap = 'Blues', annot = True, cbar =
True)

print(classification_report(true_classes, predicted_classes))
```

```
              precision    recall  f1-score   support

           0       0.70      0.79      0.74       121
           1       0.96      0.77      0.85       124
           2       0.62      0.63      0.63       124
           3       0.90      0.99      0.94        92

    accuracy                           0.78       461
   macro avg       0.80      0.79      0.79       461
weighted avg       0.79      0.78      0.78       461
```




```
model2.save("resnet_model.h5")

model3.save("vgg19.h5")

model4.save("inception_model.h5")
```

5 - EfficientNet-B0

```
model5 = tf.keras.Sequential()

base5= tf.keras.applications.EfficientNetB0(include_top=False,
                   input_shape=(224,224,3),
                   pooling='avg',
                   classes=4,
                   weights='imagenet')

model5.add(base5)

model5.add(tf.keras.layers.Flatten())
model5.add(tf.keras.layers.BatchNormalization())
model5.add(tf.keras.layers.Dense(128, activation ="relu"))
model5.add(tf.keras.layers.BatchNormalization())
model5.add(tf.keras.layers.Dense(64, activation = "relu"))
model5.add(tf.keras.layers.Dropout(0.5))
model5.add(tf.keras.layers.Dense(32, activation = "relu"))
model5.add(tf.keras.layers.Dense(4, 'softmax'))

for layer in base5.layers:
    layer.trainable = False

model4.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inception_v3 (Functional)    (None, 2048)              21802784
_________________________________________________________________
flatten_9 (Flatten)          (None, 2048)              0
_________________________________________________________________
batch_normalization_206 (Bat (None, 2048)              8192
_________________________________________________________________
dense_36 (Dense)             (None, 128)               262272
_________________________________________________________________
batch_normalization_207 (Bat (None, 128)               512
_________________________________________________________________
dense_37 (Dense)             (None, 64)                8256
_________________________________________________________________
dropout_3 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_38 (Dense)             (None, 32)                2080
_________________________________________________________________
dense_39 (Dense)             (None, 4)                 132
=================================================================
Total params: 22,084,228
Trainable params: 277,092
```

```
Non-trainable params: 21,807,136
_________________________________________________________________
```

Training the model

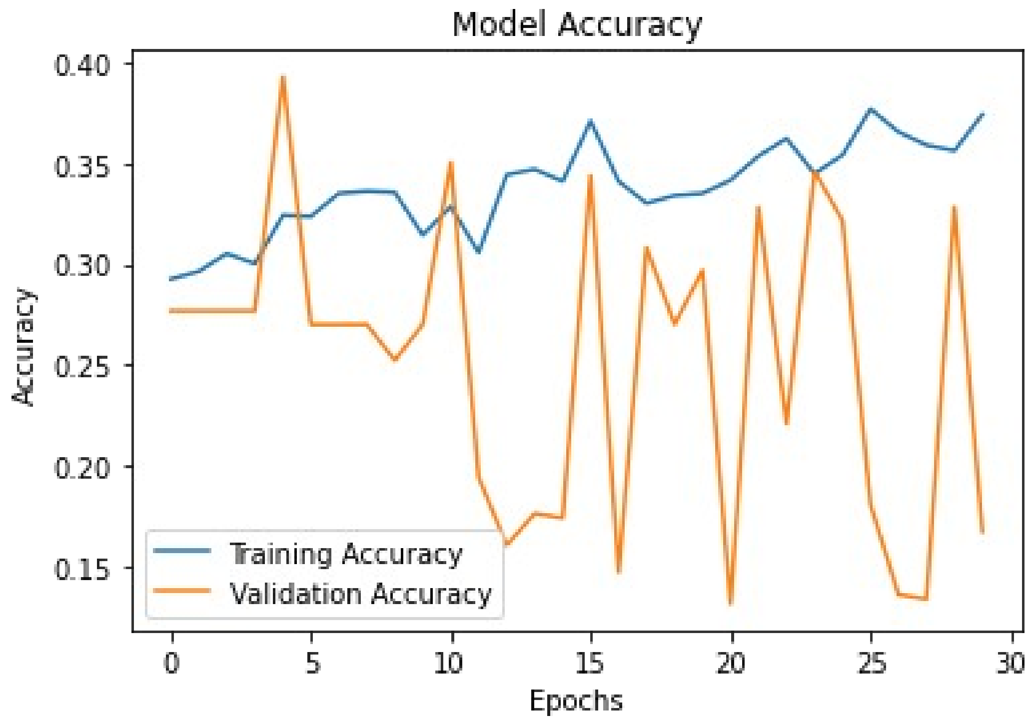
```
model5.compile(optimizer = optimizer,
              loss =
tf.keras.losses.CategoricalCrossentropy(label_smoothing = 0.2),
              metrics = ['accuracy'])

history = model5.fit(x = train_gen,
          steps_per_epoch = STEP_SIZE_TRAIN,
          validation_data = valid_gen,
          validation_steps = STEP_SIZE_VALID,
          epochs = EPOCHS,
          callbacks = early_stopping_callbacks)

Epoch 1/30
115/115 [==============================] - 40s 296ms/step - loss:
1.4786 - accuracy: 0.2926 - val_loss: 1.3873 - val_accuracy: 0.2768
Epoch 2/30
115/115 [==============================] - 30s 261ms/step - loss:
1.4124 - accuracy: 0.2964 - val_loss: 1.3918 - val_accuracy: 0.2768
Epoch 3/30
115/115 [==============================] - 30s 262ms/step - loss:
1.4032 - accuracy: 0.3051 - val_loss: 1.3879 - val_accuracy: 0.2768
Epoch 4/30
115/115 [==============================] - 30s 264ms/step - loss:
1.3893 - accuracy: 0.3002 - val_loss: 1.3778 - val_accuracy: 0.2768
Epoch 5/30
115/115 [==============================] - 31s 268ms/step - loss:
1.3664 - accuracy: 0.3241 - val_loss: 1.3698 - val_accuracy: 0.3929
Epoch 6/30
115/115 [==============================] - 31s 268ms/step - loss:
1.3646 - accuracy: 0.3235 - val_loss: 1.3742 - val_accuracy: 0.2701
Epoch 7/30
115/115 [==============================] - 30s 262ms/step - loss:
1.3615 - accuracy: 0.3350 - val_loss: 1.3932 - val_accuracy: 0.2701
Epoch 8/30
115/115 [==============================] - 30s 265ms/step - loss:
1.3557 - accuracy: 0.3361 - val_loss: 1.3811 - val_accuracy: 0.2701
Epoch 9/30
115/115 [==============================] - 30s 264ms/step - loss:
1.3616 - accuracy: 0.3355 - val_loss: 1.4124 - val_accuracy: 0.2522
Epoch 10/30
115/115 [==============================] - 30s 261ms/step - loss:
1.3611 - accuracy: 0.3143 - val_loss: 1.3773 - val_accuracy: 0.2701
Epoch 11/30
115/115 [==============================] - 30s 260ms/step - loss:
```

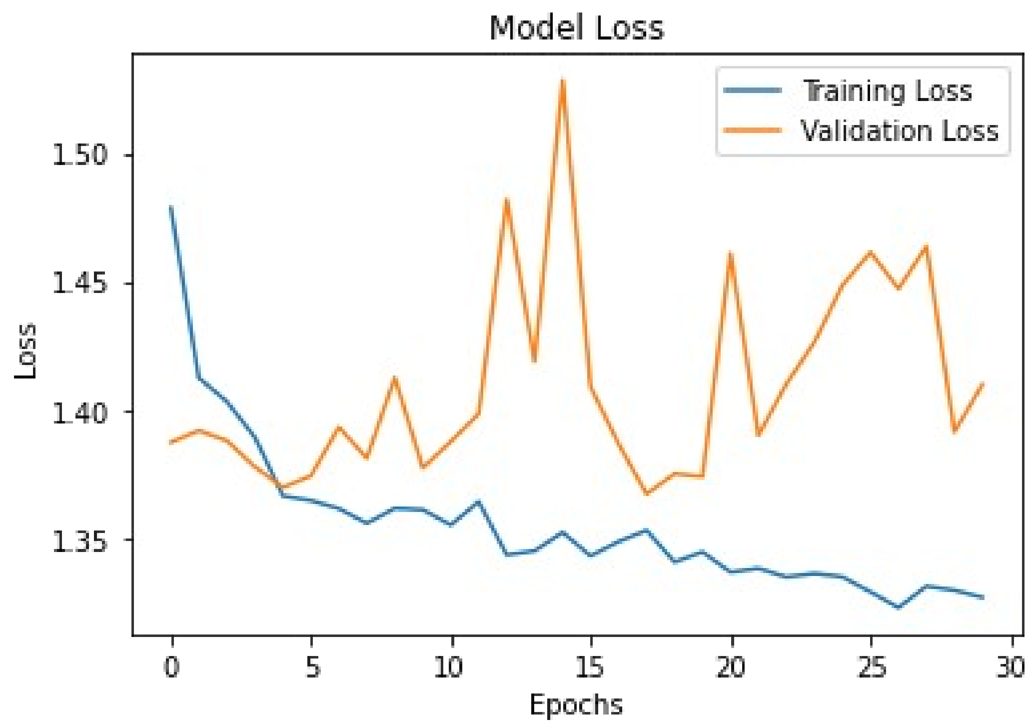
```
1.3551 - accuracy: 0.3284 - val_loss: 1.3875 - val_accuracy: 0.3504
Epoch 12/30
115/115 [==============================] - 30s 264ms/step - loss:
1.3642 - accuracy: 0.3056 - val_loss: 1.3984 - val_accuracy: 0.1942
Epoch 13/30
115/115 [==============================] - 30s 260ms/step - loss:
1.3434 - accuracy: 0.3442 - val_loss: 1.4819 - val_accuracy: 0.1607
Epoch 14/30
115/115 [==============================] - 30s 260ms/step - loss:
1.3450 - accuracy: 0.3469 - val_loss: 1.4188 - val_accuracy: 0.1763
Epoch 15/30
115/115 [==============================] - 30s 260ms/step - loss:
1.3522 - accuracy: 0.3409 - val_loss: 1.5284 - val_accuracy: 0.1741
Epoch 16/30
115/115 [==============================] - 30s 263ms/step - loss:
1.3430 - accuracy: 0.3709 - val_loss: 1.4088 - val_accuracy: 0.3438
Epoch 17/30
115/115 [==============================] - 30s 261ms/step - loss:
1.3486 - accuracy: 0.3409 - val_loss: 1.3867 - val_accuracy: 0.1473
Epoch 18/30
115/115 [==============================] - 30s 261ms/step - loss:
1.3530 - accuracy: 0.3301 - val_loss: 1.3672 - val_accuracy: 0.3080
Epoch 19/30
115/115 [==============================] - 30s 263ms/step - loss:
1.3407 - accuracy: 0.3339 - val_loss: 1.3751 - val_accuracy: 0.2701
Epoch 20/30
115/115 [==============================] - 31s 265ms/step - loss:
1.3446 - accuracy: 0.3350 - val_loss: 1.3740 - val_accuracy: 0.2969
Epoch 21/30
115/115 [==============================] - 30s 262ms/step - loss:
1.3367 - accuracy: 0.3415 - val_loss: 1.4608 - val_accuracy: 0.1317
Epoch 22/30
115/115 [==============================] - 31s 266ms/step - loss:
1.3382 - accuracy: 0.3535 - val_loss: 1.3900 - val_accuracy: 0.3281
Epoch 23/30
115/115 [==============================] - 31s 268ms/step - loss:
1.3348 - accuracy: 0.3622 - val_loss: 1.4101 - val_accuracy: 0.2210
Epoch 24/30
115/115 [==============================] - 31s 268ms/step - loss:
1.3362 - accuracy: 0.3448 - val_loss: 1.4264 - val_accuracy: 0.3460
Epoch 25/30
115/115 [==============================] - 30s 264ms/step - loss:
1.3349 - accuracy: 0.3540 - val_loss: 1.4486 - val_accuracy: 0.3214
Epoch 26/30
115/115 [==============================] - 31s 267ms/step - loss:
1.3290 - accuracy: 0.3768 - val_loss: 1.4613 - val_accuracy: 0.1808
Epoch 27/30
115/115 [==============================] - 30s 265ms/step - loss:
1.3229 - accuracy: 0.3654 - val_loss: 1.4471 - val_accuracy: 0.1362
```

```
Epoch 28/30
115/115 [==============================] - 31s 268ms/step - loss:
1.3313 - accuracy: 0.3589 - val_loss: 1.4636 - val_accuracy: 0.1339
Epoch 29/30
115/115 [==============================] - 31s 272ms/step - loss:
1.3296 - accuracy: 0.3562 - val_loss: 1.3914 - val_accuracy: 0.3281
Epoch 30/30
115/115 [==============================] - 31s 267ms/step - loss:
1.3269 - accuracy: 0.3741 - val_loss: 1.4098 - val_accuracy: 0.1674
```

Model Result

```python
import matplotlib.pyplot as plt

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

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

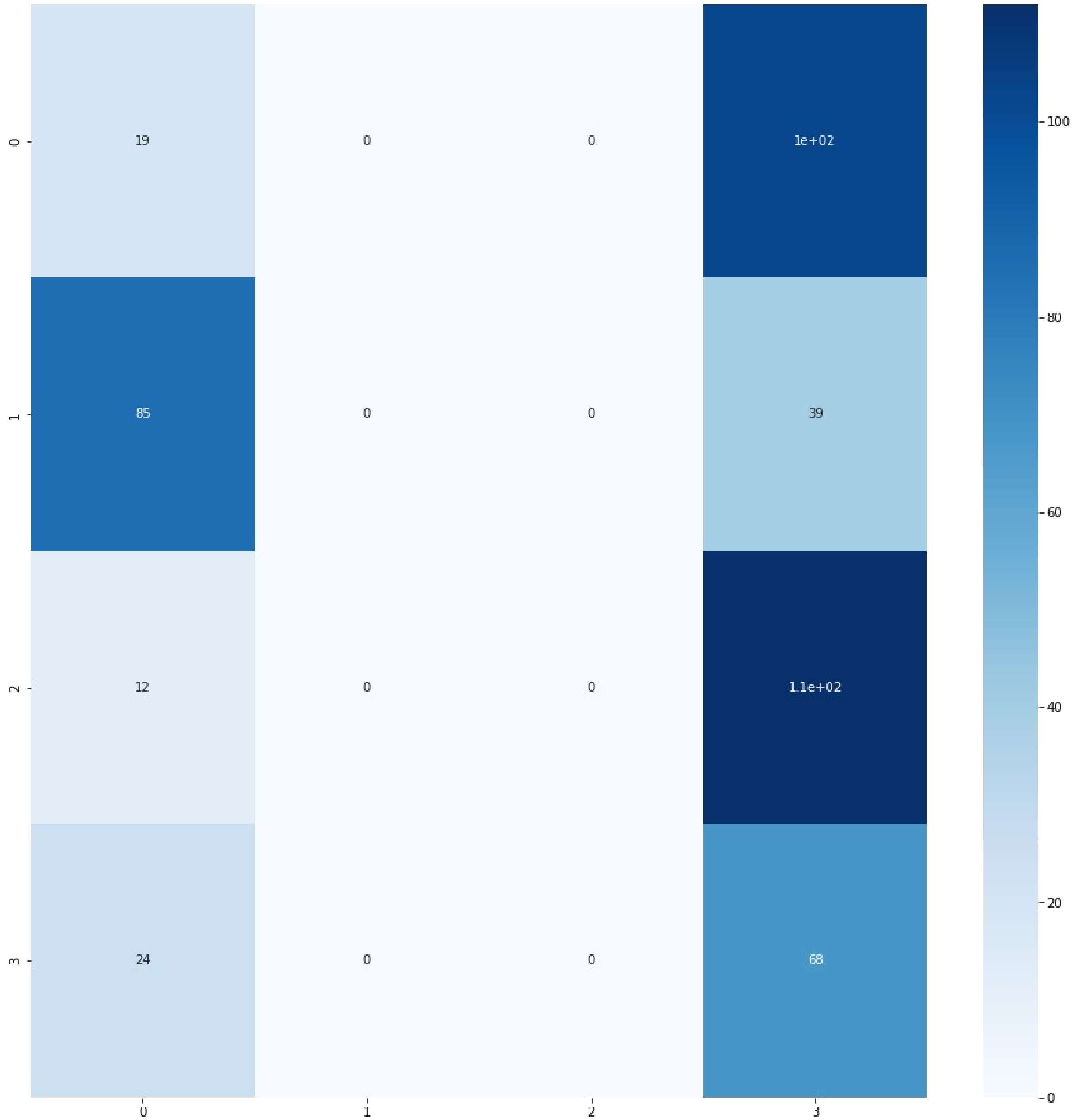





```
predicted_classes = np.argmax(model5.predict(valid_gen, steps =
valid_gen.n // valid_gen.batch_size + 1), axis = 1)
true_classes = valid_gen.classes
class_labels = list(valid_gen.class_indices.keys())
```

```python
confusionmatrix = confusion_matrix(true_classes, predicted_classes)
plt.figure(figsize = (16, 16))
sns.heatmap(confusionmatrix, cmap = 'Blues', annot = True, cbar =
True)

print(classification_report(true_classes, predicted_classes))
```

```
              precision    recall  f1-score   support

           0       0.14      0.16      0.15       121
           1       0.00      0.00      0.00       124
           2       0.00      0.00      0.00       124
           3       0.21      0.74      0.33        92

    accuracy                           0.19       461
   macro avg       0.09      0.22      0.12       461
weighted avg       0.08      0.19      0.10       461
```



19
0
0
1e+02
85
0
0
39
12
0
0
1.1e+02
24
0
0
68
0
1
2
3
0
20
40
60
80
100