```
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential, activations
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization, Activation, Dropou
from tensorflow.keras.optimizers import Adam,SGD
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import cv2 as cv2
import sklearn.metrics as sklearn
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
```

```
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_remo
```

```
!unzip gdrive/My\ Drive/Archive/archive.zip
```

```
Archive:  gdrive/My Drive/Archive/archive.zip
replace __MACOSX/._archive? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
train_path = '/content/archive/train'
test_path = '/content/archive/test'
```

```
class_names = ['Benign',"Malignant"]
class_labels = {}
for i, classes in enumerate(class_names, start=0):
  class_labels[classes] = i

print(class_labels)
```

```
{'Benign': 0, 'Malignant': 1}
```

```
train_data = []
train_labels = []

image_size = (150, 150)

for folder in os.listdir(train_path):
    print("In folder: {}".format(folder))
    for file in os.listdir(os.path.join(train_path, folder)):
        image_path = os.path.join(train_path, folder, file)
        image = cv2.imread(image_path, cv2.COLOR_BGR2RGB)
        image = cv2.resize(image, image_size)  # Resize the image using image_size
        train_data.append(image)
        train_labels.append(class_labels[folder])

train_data = np.array(train_data, dtype='float32')
train_labels = np.array(train_labels, dtype='int32')

train_data = train_data / 255.0
```

```
In folder: Benign
In folder: Malignant
```

```
test_data = []
test_labels = []

for folder in os.listdir(test_path):
  print("In folder: {}".format(folder))
  for file in os.listdir(os.path.join(test_path, folder)):
    image_path = os.path.join(test_path, folder, file)
    image = cv2.imread(image_path, cv2.COLOR_BGR2RGB)
    image = cv2.resize(image, image_size)
    test_data.append(image)
    test_labels.append(class_labels[folder])

test_data = np.array(test_data, dtype='float32')
test_labels = np.array(test_labels, dtype='int32')

test_data = test_data/255.0
```

```
In folder: Benign
In folder: Malignant
```

```
train_data.shape
```

```
(11879, 150, 150, 3)
```

```
model = tf.keras.Sequential([
        # 1st Convolutional Layer
        tf.keras.layers.Resizing(227,227, interpolation="bilinear", input_shape = train_data.shape[1:]),
        tf.keras.layers.Conv2D(filters=96, kernel_size=(11,11), strides=(4,4), activation='relu', input_shap
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPooling2D(pool_size=(3,3), strides=(2,2)),

        # 2nd Convolutional Layer
        tf.keras.layers.Conv2D(filters=256, kernel_size=(5,5), strides=(1,1), padding='same', activation='re
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPooling2D(pool_size=(3,3), strides=(2,2)),

        # 3rd Convolutional Layer
        tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same', activation='re
        tf.keras.layers.BatchNormalization(),

        # 4th Convolutional Layer
        tf.keras.layers.Conv2D(filters=384, kernel_size=(3,3), strides=(1,1), padding='same', activation='re
        tf.keras.layers.BatchNormalization(),

        # 5th Convolutional Layer
        tf.keras.layers.Conv2D(filters=256, kernel_size=(3,3), strides=(1,1), padding='same', activation='re
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.MaxPooling2D(pool_size=(3,3), strides=(2,2)),

        # Flattening
        tf.keras.layers.Flatten(),

        # 1st Fully Connected Layer
        tf.keras.layers.Dense(4096, activation='relu'),
        tf.keras.layers.Dropout(0.5),

        # 2nd Fully Connected Layer
        tf.keras.layers.Dense(4096, activation='relu'),
        tf.keras.layers.Dropout(0.5),

        # 3rd Fully Connected Layer
        tf.keras.layers.Dense(2, activation='sigmoid')
    ])
```

```
model.compile(
    loss='binary_crossentropy',
    optimizer=tf.optimizers.Adam(learning_rate=0.001),
    metrics=['accuracy'])
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resizing (Resizing)         (None, 227, 227, 3)       0
```

```
 conv2d (Conv2D)             (None, 55, 55, 96)        34944

 batch_normalization (Batch  (None, 55, 55, 96)        384
 Normalization)

 max_pooling2d (MaxPooling2  (None, 27, 27, 96)        0
 D)

 conv2d_1 (Conv2D)           (None, 27, 27, 256)       614656

 batch_normalization_1 (Bat  (None, 27, 27, 256)       1024
 chNormalization)

 max_pooling2d_1 (MaxPoolin  (None, 13, 13, 256)       0
 g2D)

 conv2d_2 (Conv2D)           (None, 13, 13, 384)       885120

 batch_normalization_2 (Bat  (None, 13, 13, 384)       1536
 chNormalization)

 conv2d_3 (Conv2D)           (None, 13, 13, 384)       1327488

 batch_normalization_3 (Bat  (None, 13, 13, 384)       1536
 chNormalization)

 conv2d_4 (Conv2D)           (None, 13, 13, 256)       884992

 batch_normalization_4 (Bat  (None, 13, 13, 256)       1024
 chNormalization)

 max_pooling2d_2 (MaxPoolin  (None, 6, 6, 256)         0
 g2D)

 flatten (Flatten)           (None, 9216)              0

 dense (Dense)               (None, 4096)              37752832

 dropout (Dropout)           (None, 4096)              0

 dense_1 (Dense)             (None, 4096)              16781312

 dropout_1 (Dropout)         (None, 4096)              0

 dense_2 (Dense)             (None, 2)                 8194

=================================================================
Total params: 58295042 (222.38 MB)
Trainable params: 58292290 (222.37 MB)
Non-trainable params: 2752 (10.75 KB)
_________________________________________________________________
```

```
call_back = EarlyStopping(
    monitor="val_loss",
    min_delta=0.00001,
    patience=5,
    verbose=1,
    mode="auto",
    baseline=None,
    restore_best_weights=False,
)
```

```
from keras.utils import to_categorical

train_labels_one_hot = to_categorical(train_labels)
test_labels_one_hot = to_categorical(test_labels)
```

```
past_data = model.fit(
    train_data,
    train_labels_one_hot,
    callbacks=call_back,
    batch_size=32,
    epochs=6,
    validation_split = 0.2
  )
```

```
Epoch 1/6
297/297 [==============================] - 31s 68ms/step - loss: 1.1218 - accuracy: 0.7737 - val_loss: 0.2455 - val_accu
Epoch 2/6
297/297 [==============================] - 17s 57ms/step - loss: 0.4296 - accuracy: 0.8177 - val_loss: 1.0965 - val_accu
Epoch 3/6
297/297 [==============================] - 17s 57ms/step - loss: 0.4377 - accuracy: 0.8227 - val_loss: 1.1630 - val_accu
```

```
Epoch 4/6
297/297 [==============================] - 17s 58ms/step - loss: 0.4229 - accuracy: 0.8272 - val_loss: 0.6650 - val_accu
Epoch 5/6
297/297 [==============================] - 17s 59ms/step - loss: 0.3984 - accuracy: 0.8335 - val_loss: 0.7609 - val_accu
Epoch 6/6
297/297 [==============================] - 17s 58ms/step - loss: 0.4140 - accuracy: 0.8330 - val_loss: 1.0508 - val_accu
Epoch 6: early stopping
```

```
test_labels_binary = np.argmax(
    test_labels_one_hot,
    axis=1
  )
loss, acc = model.evaluate(
    test_data,
    test_labels_one_hot,
    verbose=2
  )
print(f'Test accuracy: {acc:.4f}')
```

```
63/63 - 2s - loss: 0.7108 - accuracy: 0.5795 - 2s/epoch - 24ms/step
Test accuracy: 0.5795
```

```
y_pred = np.argmax(model.predict(test_data), axis=-1)
```

```
63/63 [==============================] - 1s 15ms/step
```

```
score=accuracy_score(test_labels,y_pred)
print(score)
```

```
0.5795
```

```
score=f1_score(test_labels,y_pred)
print(score)
```

```
0.2781115879828326
```

```
score=recall_score(test_labels,y_pred)
print(score)
```

```
0.162
```

```
score=precision_score(test_labels,y_pred)
print(score)
```

```
0.9818181818181818
```

```
result = confusion_matrix(test_labels, y_pred, normalize='pred')
print(result)
```

```
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential, activations
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization, Activation, Dropout, GlobalAvera
from tensorflow.keras.applications import ResNet50, VGG16, VGG19, DenseNet201, InceptionResNetV2, ResNet50, ResNet101, Incep
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import cv2 as cv2
import sklearn.metrics as sklearn
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
```

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

```
Mounted at /content/gdrive
```

```
!unzip gdrive/My\ Drive/Archive/archive.zip
```

```
train_path = '/content/archive/train'
test_path = '/content/archive/test'
```

```
class_names = ['Benign',"Malignant"]
class_labels = {}
for i, classes in enumerate(class_names, start=0):
  class_labels[classes] = i

print(class_labels)
```

```
{'Benign': 0, 'Malignant': 1}
```

```
train_datagen = ImageDataGenerator(
  rescale=1./255,
  rotation_range=20,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True,
  fill_mode='nearest'
  )
test_datagen = ImageDataGenerator(rescale=1./255)
```

```
train_dataset = train_datagen.flow_from_directory(
train_path,
target_size=(150, 150),
batch_size=32,
class_mode='binary'
)
```

```
Found 11879 images belonging to 2 classes.
```

```
test_dataset = test_datagen.flow_from_directory(
test_path,
target_size=(150, 150),
batch_size=32,
class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
```

```
def create_resnet50_model():
  base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
  base_model,
  GlobalAveragePooling2D(),
  Dense(1, activation='sigmoid')
  ])
  return model


def create_vgg16_model():
  base_model = VGG16(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
  base_model,
  GlobalAveragePooling2D(),
  Dense(1, activation='softmax')
  ])
  return model


def create_vgg19_model():
  base_model = VGG19(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
  base_model,
  GlobalAveragePooling2D(),
  Dense(1, activation='softmax')
  ])
  return model


models = [ create_resnet50_model(),create_vgg16_model(),
          create_vgg19_model()
          ]

for model in models:
  model.compile(
        loss='binary_crossentropy',
        optimizer=tf.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy'])
  model.summary()
```

```
Model: "sequential_7"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet50 (Functional)       (None, 5, 5, 2048)        23587712

 global_average_pooling2d_7  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_7 (Dense)             (None, 1)                 2049

=================================================================
Total params: 23589761 (89.99 MB)
Trainable params: 23536641 (89.79 MB)
Non-trainable params: 53120 (207.50 KB)
_________________________________________________________________
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg16 (Functional)          (None, 4, 4, 512)         14714688

 global_average_pooling2d_8  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_8 (Dense)             (None, 1)                 513

=================================================================
Total params: 14715201 (56.13 MB)
Trainable params: 14715201 (56.13 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Model: "sequential_9"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 vgg19 (Functional)          (None, 4, 4, 512)         20024384

 global_average_pooling2d_9  (None, 512)               0
  (GlobalAveragePooling2D)

 dense_9 (Dense)             (None, 1)                 513

=================================================================
Total params: 20024897 (76.39 MB)
Trainable params: 20024897 (76.39 MB)
```

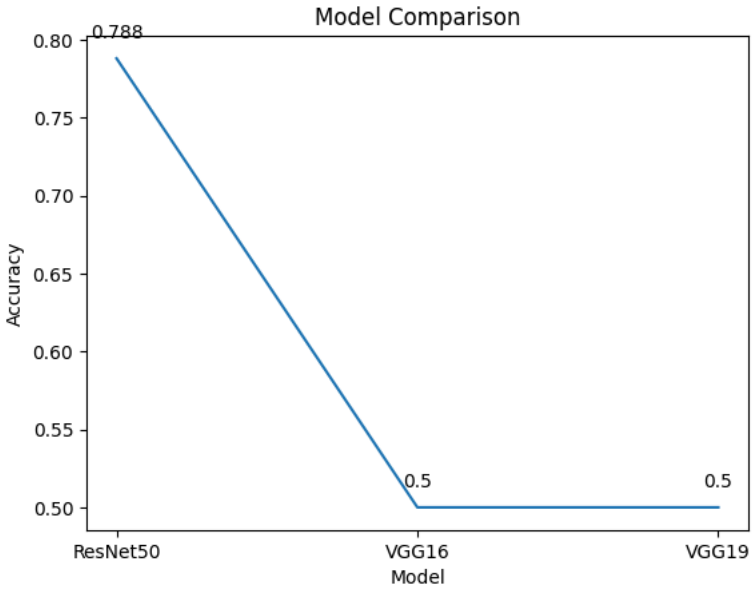
```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

```
histories = []
accuracies = []
recalls = []
precisions = []
f1_scores = []

for model in models:
    history = model.fit(
        train_dataset,
        epochs=6,
        validation_data=test_dataset
    )
    histories.append(history)

    loss, accuracy = model.evaluate(test_dataset)
    accuracies.append(accuracy)

    y_pred = model.predict(test_dataset)
    y_pred_binary = np.round(y_pred)

    precision = sklearn.precision_score(test_dataset.labels, y_pred_binary,zero_division=1)
    recall = sklearn.recall_score(test_dataset.labels, y_pred_binary,zero_division=1)
    f1 = sklearn.f1_score(test_dataset.labels, y_pred_binary)

    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

for i, model in enumerate(models):
    print(f"Model {i+1}:")
    print(f"  Precision: {precisions[i]}")
    print(f"  Recall: {recalls[i]}")
    print(f"  F1 Score: {f1_scores[i]}")
```

```
Epoch 1/6
372/372 [==============================] - 134s 282ms/step - loss: 0.4104 - accuracy: 0.8334 - val_loss: 1.0165 - val_ac
Epoch 2/6
372/372 [==============================] - 107s 287ms/step - loss: 0.3333 - accuracy: 0.8567 - val_loss: 1.4455 - val_ac
Epoch 3/6
372/372 [==============================] - 107s 286ms/step - loss: 0.3020 - accuracy: 0.8704 - val_loss: 2.4574 - val_ac
Epoch 4/6
372/372 [==============================] - 105s 281ms/step - loss: 0.2995 - accuracy: 0.8695 - val_loss: 0.5916 - val_ac
Epoch 5/6
372/372 [==============================] - 107s 286ms/step - loss: 0.2869 - accuracy: 0.8789 - val_loss: 0.3475 - val_ac
Epoch 6/6
372/372 [==============================] - 108s 289ms/step - loss: 0.2820 - accuracy: 0.8788 - val_loss: 0.4035 - val_ac
63/63 [==============================] - 4s 59ms/step - loss: 0.4035 - accuracy: 0.7880
63/63 [==============================] - 4s 55ms/step
Epoch 1/6
372/372 [==============================] - 112s 287ms/step - loss: 1.2731 - accuracy: 0.4706 - val_loss: 0.6939 - val_ac
Epoch 2/6
372/372 [==============================] - 108s 289ms/step - loss: 0.6915 - accuracy: 0.4706 - val_loss: 0.6945 - val_ac
Epoch 3/6
372/372 [==============================] - 107s 286ms/step - loss: 0.6915 - accuracy: 0.4706 - val_loss: 0.6949 - val_ac
Epoch 4/6
372/372 [==============================] - 107s 287ms/step - loss: 0.6914 - accuracy: 0.4706 - val_loss: 0.6949 - val_ac
Epoch 5/6
372/372 [==============================] - 105s 283ms/step - loss: 0.6915 - accuracy: 0.4706 - val_loss: 0.6949 - val_ac
Epoch 6/6
372/372 [==============================] - 107s 286ms/step - loss: 0.6914 - accuracy: 0.4706 - val_loss: 0.6948 - val_ac
63/63 [==============================] - 4s 66ms/step - loss: 0.6948 - accuracy: 0.5000
63/63 [==============================] - 4s 63ms/step
Epoch 1/6
372/372 [==============================] - 120s 307ms/step - loss: 0.9842 - accuracy: 0.4706 - val_loss: 0.6937 - val_ac
Epoch 2/6
372/372 [==============================] - 115s 309ms/step - loss: 0.6916 - accuracy: 0.4706 - val_loss: 0.6944 - val_ac
Epoch 3/6
372/372 [==============================] - 115s 308ms/step - loss: 0.6915 - accuracy: 0.4706 - val_loss: 0.6946 - val_ac
Epoch 4/6
372/372 [==============================] - 115s 308ms/step - loss: 0.6914 - accuracy: 0.4706 - val_loss: 0.6948 - val_ac
Epoch 5/6
372/372 [==============================] - 115s 310ms/step - loss: 0.6914 - accuracy: 0.4706 - val_loss: 0.6949 - val_ac
Epoch 6/6
372/372 [==============================] - 116s 312ms/step - loss: 0.6915 - accuracy: 0.4706 - val_loss: 0.6947 - val_ac
63/63 [==============================] - 5s 78ms/step - loss: 0.6947 - accuracy: 0.5000
63/63 [==============================] - 5s 78ms/step
Model 1:
  Precision: 0.504424778761062
  Recall: 0.342
  F1 Score: 0.40762812872467225
Model 2:
  Precision: 0.5
```

```
        Recall: 1.0
        F1 Score: 0.6666666666666666
      Model 3:
        Precision: 0.5
        Recall: 1.0
        F1 Score: 0.6666666666666666
```

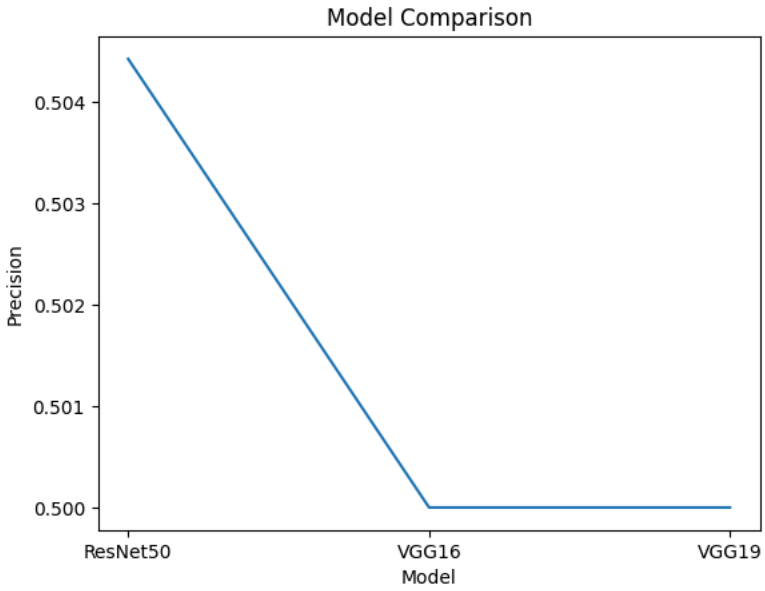
```
model_names = ['ResNet50', 'VGG16', 'VGG19']
plt.plot(model_names, accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Comparison')

for i, v in enumerate(accuracies):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```



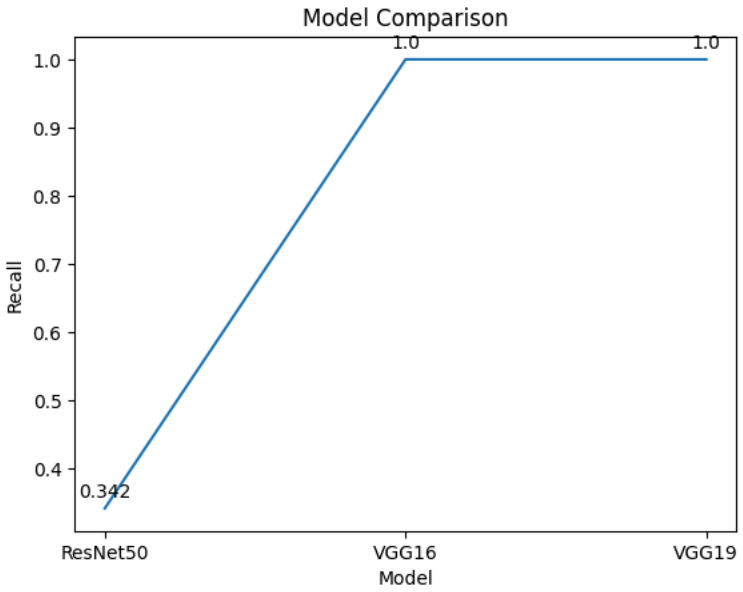

```
model_names = ['ResNet50', 'VGG16', 'VGG19']
plt.plot(model_names, precisions)
plt.xlabel('Model')
plt.ylabel('Precision')
plt.title('Model Comparison')

for i, v in enumerate(precisions):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```

0.5044

0.5 0.5




```
model_names = ['ResNet50', 'VGG16', 'VGG19']
plt.plot(model_names, recalls)
plt.xlabel('Model')
plt.ylabel('Recall')
plt.title('Model Comparison')

for i, v in enumerate(recalls):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```




```
model_names = ['ResNet50', 'VGG16', 'VGG19']
plt.plot(model_names, f1_scores)
plt.xlabel('Model')
plt.ylabel('F1 Score')
plt.title('Model Comparison')

for i, v in enumerate(f1_scores):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```

```
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential, activations
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization, Activation, Dropout, GlobalAverag
from tensorflow.keras.applications import ResNet50, VGG16, VGG19, DenseNet201, InceptionResNetV2, ResNet50, ResNet101, Incept
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import cv2 as cv2
import sklearn.metrics as sklearn
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import accuracy_score
```

```
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_remo
```

```
!unzip gdrive/My\ Drive/Archive/archive.zip
```

```
Archive:  gdrive/My Drive/Archive/archive.zip
replace __MACOSX/._archive? [y]es, [n]o, [A]ll, [N]one, [r]ename:
```

```
train_path = '/content/archive/train'
test_path = '/content/archive/test'
```

```
class_names = ['Benign',"Malignant"]
class_labels = {}
for i, classes in enumerate(class_names, start=0):
  class_labels[classes] = i

print(class_labels)
```

```
{'Benign': 0, 'Malignant': 1}
```

```
train_datagen = ImageDataGenerator(
  rescale=1./255,
  rotation_range=20,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True,
  fill_mode='nearest'
  )
test_datagen = ImageDataGenerator(rescale=1./255)
```

```
train_dataset = train_datagen.flow_from_directory(
train_path,
target_size=(150, 150),
batch_size=32,
class_mode='binary'
)
```

```
Found 11879 images belonging to 2 classes.
```

```
test_dataset = test_datagen.flow_from_directory(
test_path,
target_size=(150, 150),
batch_size=32,
class_mode='binary'
)
```

```
Found 2000 images belonging to 2 classes.
```

```
def create_Inception_Resnet_V2():
  base_model = InceptionResNetV2(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(1, activation='sigmoid')
  ])
  return model


def create_Resnet101():
  base_model = ResNet101(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(1, activation='sigmoid')
  ])
  return model


def create_InceptionV3():
  base_model = InceptionV3(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(1, activation='sigmoid')
  ])
  return model


models = [create_Inception_Resnet_V2(),
          create_Resnet101(), create_InceptionV3()
          ]

for model in models:
  model.compile(
        loss='binary_crossentropy',
        optimizer=tf.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy'])
  model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_resnet_v2 (Funct  (None, 3, 3, 1536)        54336736
 ional)

 global_average_pooling2d_3  (None, 1536)              0
  (GlobalAveragePooling2D)

 dense_3 (Dense)             (None, 1)                 1537

=================================================================
Total params: 54338273 (207.28 MB)
Trainable params: 54277729 (207.05 MB)
Non-trainable params: 60544 (236.50 KB)
_________________________________________________________________
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 resnet101 (Functional)      (None, 5, 5, 2048)        42658176

 global_average_pooling2d_4  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_4 (Dense)             (None, 1)                 2049

=================================================================
Total params: 42660225 (162.74 MB)
Trainable params: 42554881 (162.33 MB)
Non-trainable params: 105344 (411.50 KB)
_________________________________________________________________
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 inception_v3 (Functional)   (None, 3, 3, 2048)        21802784

 global_average_pooling2d_5  (None, 2048)              0
  (GlobalAveragePooling2D)

 dense_5 (Dense)             (None, 1)                 2049

=================================================================
Total params: 21804833 (83.18 MB)
```

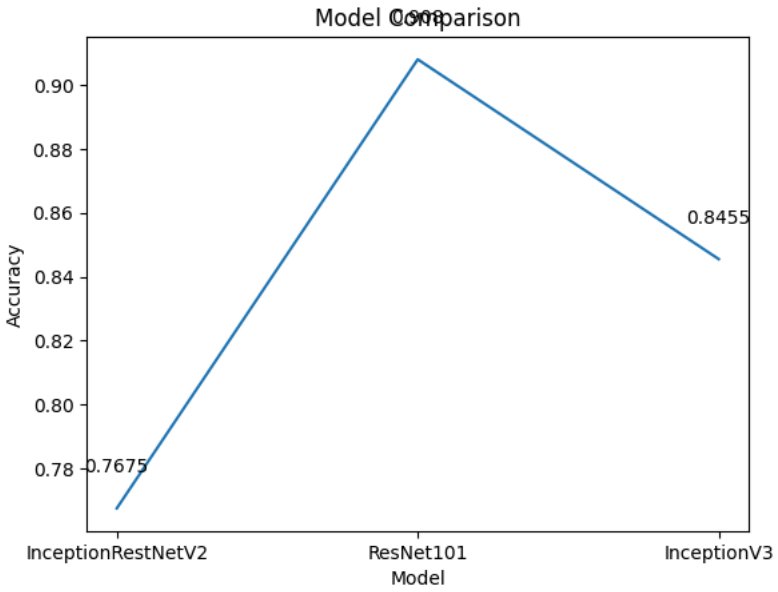
```
Trainable params: 21770401 (83.05 MB)
Non-trainable params: 34432 (134.50 KB)
_________________________________________________________________
```

```
histories = []
accuracies = []
recalls = []
precisions = []
f1_scores = []

for model in models:
    history = model.fit(
        train_dataset,
        epochs=5,
        validation_data=test_dataset
    )
    histories.append(history)

    loss, accuracy = model.evaluate(test_dataset)
    accuracies.append(accuracy)

    y_pred = model.predict(test_dataset)
    y_pred_binary = np.round(y_pred)

    precision = sklearn.precision_score(test_dataset.labels, y_pred_binary)
    recall = sklearn.recall_score(test_dataset.labels, y_pred_binary)
    f1 = sklearn.f1_score(test_dataset.labels, y_pred_binary)

    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

for i, model in enumerate(models):
    print(f"Model {i+1}:")
    print(f"  Precision: {precisions[i]}")
    print(f"  Recall: {recalls[i]}")
    print(f"  F1 Score: {f1_scores[i]}")
```

```
Epoch 1/5
372/372 [==============================] - 222s 329ms/step - loss: 0.3325 - accuracy: 0.8640 - val_loss: 0.3424 - val_ac
Epoch 2/5
372/372 [==============================] - 116s 310ms/step - loss: 0.2856 - accuracy: 0.8828 - val_loss: 0.3550 - val_ac
Epoch 3/5
372/372 [==============================] - 115s 309ms/step - loss: 0.2693 - accuracy: 0.8874 - val_loss: 0.2937 - val_ac
Epoch 4/5
372/372 [==============================] - 115s 309ms/step - loss: 0.2543 - accuracy: 0.8928 - val_loss: 0.3189 - val_ac
Epoch 5/5
372/372 [==============================] - 116s 312ms/step - loss: 0.2459 - accuracy: 0.8942 - val_loss: 0.4195 - val_ac
63/63 [==============================] - 5s 80ms/step - loss: 0.4195 - accuracy: 0.7675
63/63 [==============================] - 8s 70ms/step
Epoch 1/5
372/372 [==============================] - 204s 355ms/step - loss: 0.4468 - accuracy: 0.8120 - val_loss: 2.7761 - val_ac
Epoch 2/5
372/372 [==============================] - 130s 349ms/step - loss: 0.4126 - accuracy: 0.8161 - val_loss: 0.6834 - val_ac
Epoch 3/5
372/372 [==============================] - 131s 351ms/step - loss: 0.3376 - accuracy: 0.8534 - val_loss: 1.3581 - val_ac
Epoch 4/5
372/372 [==============================] - 132s 354ms/step - loss: 0.3270 - accuracy: 0.8590 - val_loss: 0.2529 - val_ac
Epoch 5/5
372/372 [==============================] - 132s 354ms/step - loss: 0.3130 - accuracy: 0.8648 - val_loss: 0.2322 - val_ac
63/63 [==============================] - 6s 101ms/step - loss: 0.2322 - accuracy: 0.9080
63/63 [==============================] - 8s 96ms/step
Epoch 1/5
372/372 [==============================] - 131s 258ms/step - loss: 0.4067 - accuracy: 0.8235 - val_loss: 0.7230 - val_ac
Epoch 2/5
372/372 [==============================] - 93s 250ms/step - loss: 0.3330 - accuracy: 0.8560 - val_loss: 0.4139 - val_acc
Epoch 3/5
372/372 [==============================] - 88s 237ms/step - loss: 0.3303 - accuracy: 0.8626 - val_loss: 0.5638 - val_acc
Epoch 4/5
372/372 [==============================] - 91s 245ms/step - loss: 0.3513 - accuracy: 0.8491 - val_loss: 33.8130 - val_ac
Epoch 5/5
372/372 [==============================] - 87s 235ms/step - loss: 0.3404 - accuracy: 0.8519 - val_loss: 0.3479 - val_acc
63/63 [==============================] - 3s 48ms/step - loss: 0.3479 - accuracy: 0.8455
63/63 [==============================] - 4s 37ms/step
Model 1:
  Precision: 0.48911222780569513
  Recall: 0.292
  F1 Score: 0.3656856606136506
Model 2:
  Precision: 0.4797979797979798
  Recall: 0.475
  F1 Score: 0.4773869346733668
Model 3:
  Precision: 0.49631190727081137
  Recall: 0.471
```

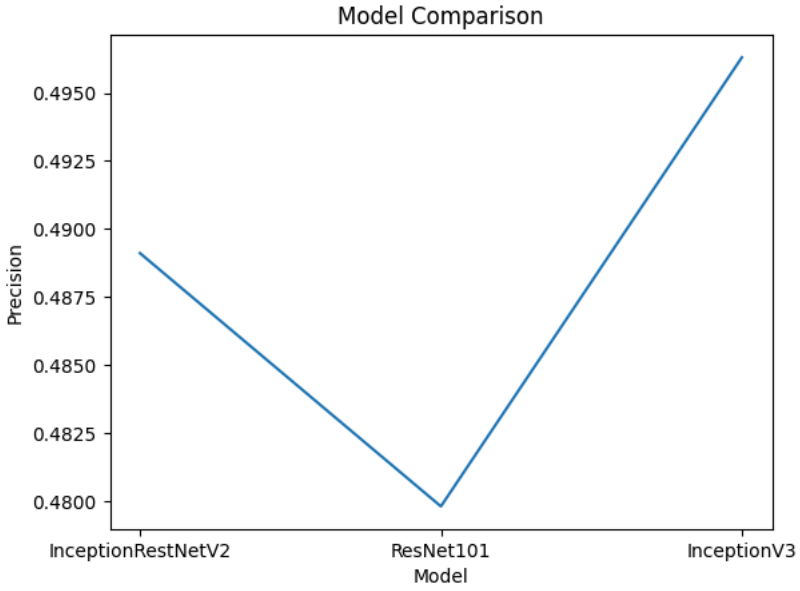
```
F1 Score: 0.4833247819394561
```

```
model_names = ['InceptionRestNetV2', 'ResNet101','InceptionV3']
plt.plot(model_names, accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Comparison')

for i, v in enumerate(accuracies):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```



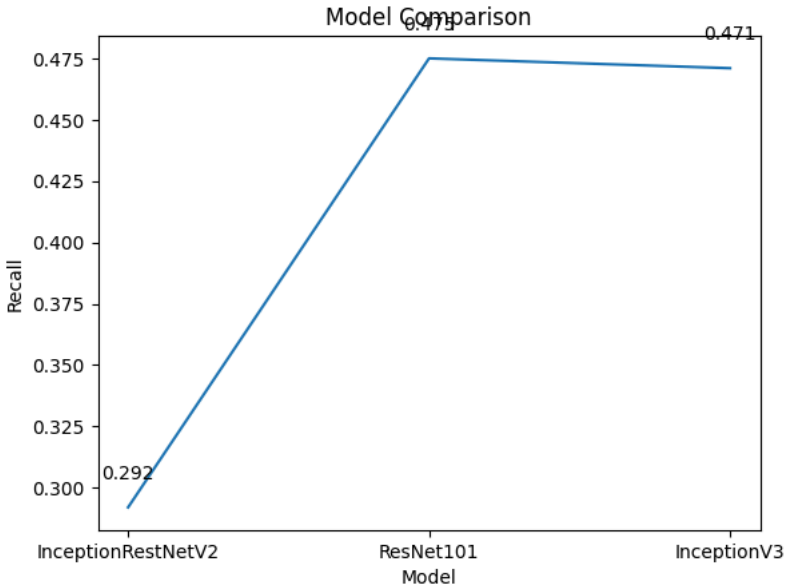

```
model_names = ['InceptionRestNetV2', 'ResNet101','InceptionV3']
plt.plot(model_names, precisions)
plt.xlabel('Model')
plt.ylabel('Precision')
plt.title('Model Comparison')

for i, v in enumerate(precision):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-38-5accfe4b0fd7> in <cell line: 7>()
      5 plt.title('Model Comparison')
      6
----> 7 for i, v in enumerate(precision):
      8   plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
      9 plt.show()

TypeError: 'numpy.float64' object is not iterable
```



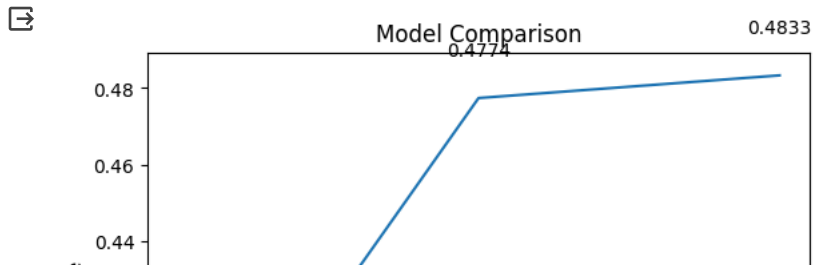
Next steps: Explain error

```
model_names = ['InceptionRestNetV2', 'ResNet101','InceptionV3']
plt.plot(model_names, recalls)
plt.xlabel('Model')
plt.ylabel('Recall')
plt.title('Model Comparison')

for i, v in enumerate(recalls):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```




```
model_names = ['InceptionRestNetV2', 'ResNet101','InceptionV3']
plt.plot(model_names, f1_scores)
plt.xlabel('Model')
plt.ylabel('F1 Score')
plt.title('Model Comparison')

for i, v in enumerate(f1_scores):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
```

```
plt.show()
```



```
import torch
import torch.nn as nn
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import os
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras import Sequential, activations
from keras.layers import Conv2D, MaxPooling2D, Dense, Flatten, Dropout, BatchNormalization, Activation, Dropout, GlobalAverag
from tensorflow.keras.applications import ResNet50, VGG16, VGG19, DenseNet201, InceptionResNetV2, ResNet50, ResNet101, Incept
from keras.callbacks import EarlyStopping
from sklearn.model_selection import train_test_split
import cv2 as cv2
import sklearn.metrics as sklearn
from sklearn.metrics import f1_score, recall_score, precision_score, accuracy_score


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

```
    Mounted at /content/gdrive
```

```
!unzip gdrive/My\ Drive/Archive/archive.zip
```

```
train_path = '/content/archive/train'
test_path = '/content/archive/test'
```

```
class_names = ['Benign',"Malignant"]
class_labels = {}
for i, classes in enumerate(class_names, start=0):
  class_labels[classes] = i

print(class_labels)
```

```
    {'Benign': 0, 'Malignant': 1}
```

```
train_datagen = ImageDataGenerator(
  rescale=1./255,
  rotation_range=20,
  width_shift_range=0.2,
  height_shift_range=0.2,
  shear_range=0.2,
  zoom_range=0.2,
  horizontal_flip=True,
  fill_mode='nearest'
  )
test_datagen = ImageDataGenerator(rescale=1./255)
```

```
train_dataset = train_datagen.flow_from_directory(
train_path,
target_size=(150, 150),
batch_size=32,
class_mode='binary'
)
```

```
    Found 11879 images belonging to 2 classes.
```

```
test_dataset = test_datagen.flow_from_directory(
test_path,
target_size=(150, 150),
batch_size=32,
class_mode='binary'
)
```

```
    Found 2000 images belonging to 2 classes.
```

```
def create_DenseNet121():
  base_model = DenseNet121(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(1, activation='sigmoid')
  ])
  return model
```

```
def create_EfficientNetB1():
  base_model = EfficientNetB1(weights='imagenet', include_top=False, input_shape=(150,150,3))
  model = Sequential([
        base_model,
        GlobalAveragePooling2D(),
        Dense(1, activation='sigmoid')
  ])
  return model


models = [
          create_DenseNet121(), create_EfficientNetB1()
          ]

for model in models:
  model.compile(
        loss='binary_crossentropy',
        optimizer=tf.optimizers.Adam(learning_rate=0.001),
        metrics=['accuracy'])
  model.summary()
```

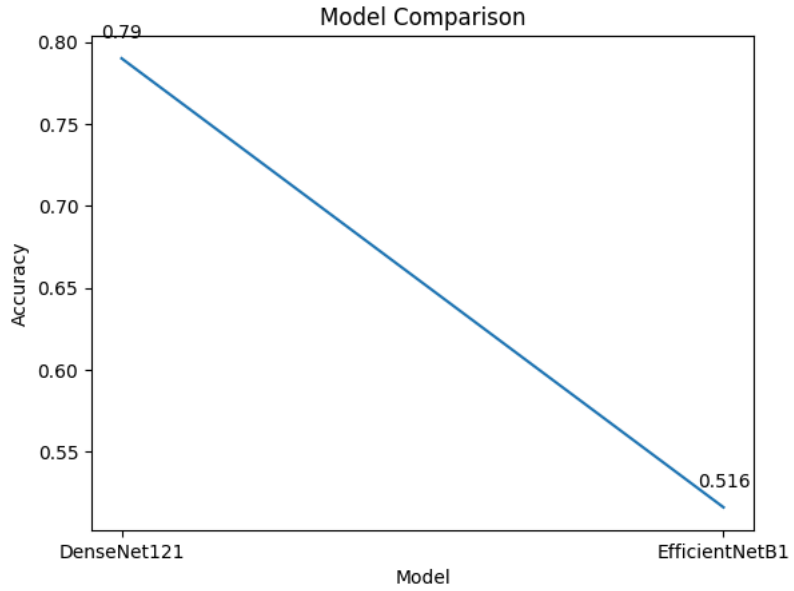
```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_o
29084464/29084464 [==============================] - 0s 0us/step
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb1_notop.h5
27018416/27018416 [==============================] - 0s 0us/step
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet121 (Functional)    (None, 4, 4, 1024)        7037504

 global_average_pooling2d (  (None, 1024)              0
 GlobalAveragePooling2D)

 dense (Dense)               (None, 1)                 1025

=================================================================
Total params: 7038529 (26.85 MB)
Trainable params: 6954881 (26.53 MB)
Non-trainable params: 83648 (326.75 KB)
_________________________________________________________________
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 efficientnetb1 (Functional  (None, 5, 5, 1280)        6575239
 )

 global_average_pooling2d_1  (None, 1280)              0
  (GlobalAveragePooling2D)

 dense_1 (Dense)             (None, 1)                 1281

=================================================================
Total params: 6576520 (25.09 MB)
Trainable params: 6514465 (24.85 MB)
Non-trainable params: 62055 (242.41 KB)
_________________________________________________________________
```

```
histories = []
accuracies = []
recalls = []
precisions = []
f1_scores = []

for model in models:
    history = model.fit(
        train_dataset,
        epochs=6,
        validation_data=test_dataset
    )
    histories.append(history)

    loss, accuracy = model.evaluate(test_dataset)
    accuracies.append(accuracy)

    y_pred = model.predict(test_dataset)
    y_pred_binary = np.round(y_pred)

    precision = sklearn.precision_score(test_dataset.labels, y_pred_binary)
    recall = sklearn.recall_score(test_dataset.labels, y_pred_binary)
    f1 = sklearn.f1_score(test_dataset.labels, y_pred_binary)
```

```
    precisions.append(precision)
    recalls.append(recall)
    f1_scores.append(f1)

for i, model in enumerate(models):
    print(f"Model {i+1}:")
    print(f"  Precision: {precisions[i]}")
    print(f"  Recall: {recalls[i]}")
    print(f"  F1 Score: {f1_scores[i]}")
```

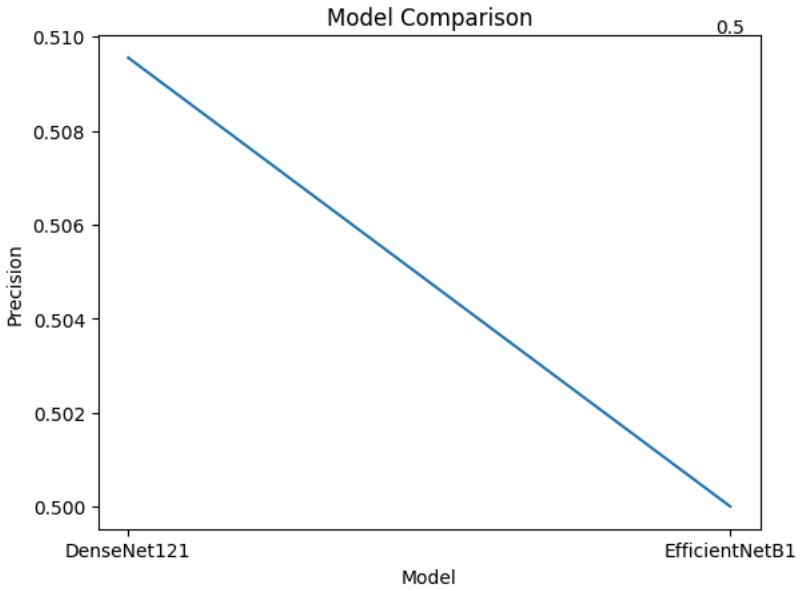
```
Epoch 1/6
372/372 [==============================] - 106s 284ms/step - loss: 0.2981 - accuracy: 0.8718 - val_loss: 1.2396 - val_ac
Epoch 2/6
372/372 [==============================] - 102s 275ms/step - loss: 0.3022 - accuracy: 0.8687 - val_loss: 0.3027 - val_ac
Epoch 3/6
372/372 [==============================] - 111s 298ms/step - loss: 0.2828 - accuracy: 0.8770 - val_loss: 0.4025 - val_ac
Epoch 4/6
372/372 [==============================] - 105s 282ms/step - loss: 0.2753 - accuracy: 0.8831 - val_loss: 0.3822 - val_ac
Epoch 5/6
372/372 [==============================] - 103s 277ms/step - loss: 0.2574 - accuracy: 0.8891 - val_loss: 0.5577 - val_ac
Epoch 6/6
372/372 [==============================] - 103s 276ms/step - loss: 0.2646 - accuracy: 0.8869 - val_loss: 0.4035 - val_ac
63/63 [==============================] - 3s 51ms/step - loss: 0.4035 - accuracy: 0.7900
63/63 [==============================] - 4s 62ms/step
Epoch 1/6
372/372 [==============================] - 101s 272ms/step - loss: 0.2629 - accuracy: 0.8880 - val_loss: 38.0783 - val_a
Epoch 2/6
372/372 [==============================] - 107s 286ms/step - loss: 0.2479 - accuracy: 0.8948 - val_loss: 1.3019 - val_ac
Epoch 3/6
372/372 [==============================] - 133s 358ms/step - loss: 0.2336 - accuracy: 0.9000 - val_loss: 0.6344 - val_ac
Epoch 4/6
372/372 [==============================] - 110s 294ms/step - loss: 0.2287 - accuracy: 0.9042 - val_loss: 24.0864 - val_a
Epoch 5/6
372/372 [==============================] - 106s 284ms/step - loss: 0.2193 - accuracy: 0.9082 - val_loss: 2.2976 - val_ac
Epoch 6/6
372/372 [==============================] - 102s 273ms/step - loss: 0.2156 - accuracy: 0.9077 - val_loss: 1.4530 - val_ac
63/63 [==============================] - 3s 45ms/step - loss: 1.4530 - accuracy: 0.5160
63/63 [==============================] - 3s 50ms/step
Model 1:
  Precision: 0.5095541401273885
  Recall: 0.32
  F1 Score: 0.3931203931203931
Model 2:
  Precision: 0.5
  Recall: 0.016
  F1 Score: 0.031007751937984496
```

```
model_names = ['DenseNet121','EfficientNetB1' ]
plt.plot(model_names, accuracies)
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.title('Model Comparison')

for i, v in enumerate(accuracies):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```

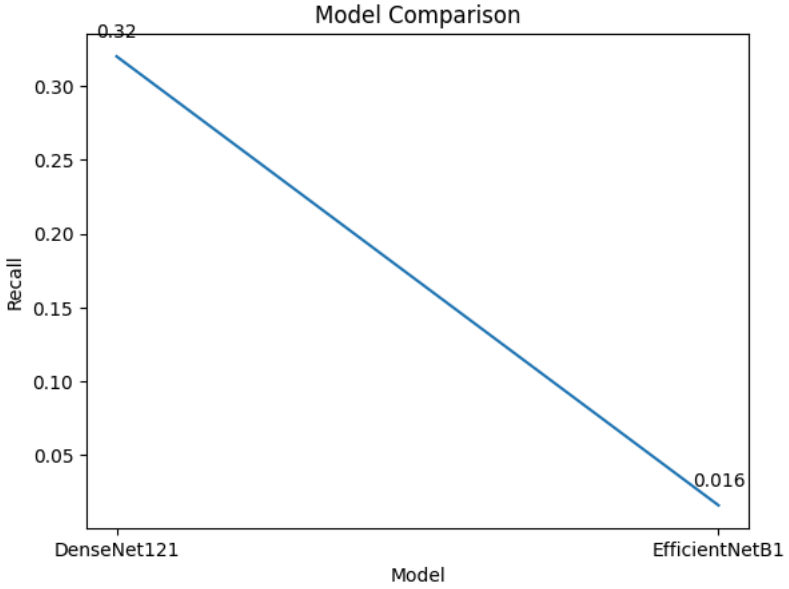


```
model_names = ['DenseNet121','EfficientNetB1' ]
plt.plot(model_names, precisions)
plt.xlabel('Model')
plt.ylabel('Precision')
plt.title('Model Comparison')

for i, v in enumerate(precisions):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```

```
0.5096
```




```
model_names = ['DenseNet121','EfficientNetB1' ]
plt.plot(model_names, recalls)
plt.xlabel('Model')
plt.ylabel('Recall')
plt.title('Model Comparison')

for i, v in enumerate(recalls):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```




```
model_names = ['DenseNet121','EfficientNetB1' ]
plt.plot(model_names, f1_scores)
plt.xlabel('Model')
plt.ylabel('F1 Score')
plt.title('Model Comparison')

for i, v in enumerate(f1_scores):
  plt.text(i, v + 0.01, str(round(v, 4)), ha='center', va='bottom')
plt.show()
```