Chest Cancer Classification with VGG16

What is the VGG16

The VGG16 model is a deep convolutional neural network (CNN) architecture that was introduced by researchers at the University of Oxford, specifically Karen Simonyan and Andrew Zisserman. The name "VGG" stands for the Visual Geometry Group, the research group where the model was developed.

The VGG16 architecture is characterized by its depth and simplicity. It consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. The convolutional layers are stacked one after another, with small 3x3 convolutional filters and max-pooling layers applied periodically to reduce spatial dimensions.

Key features of the VGG16 model:

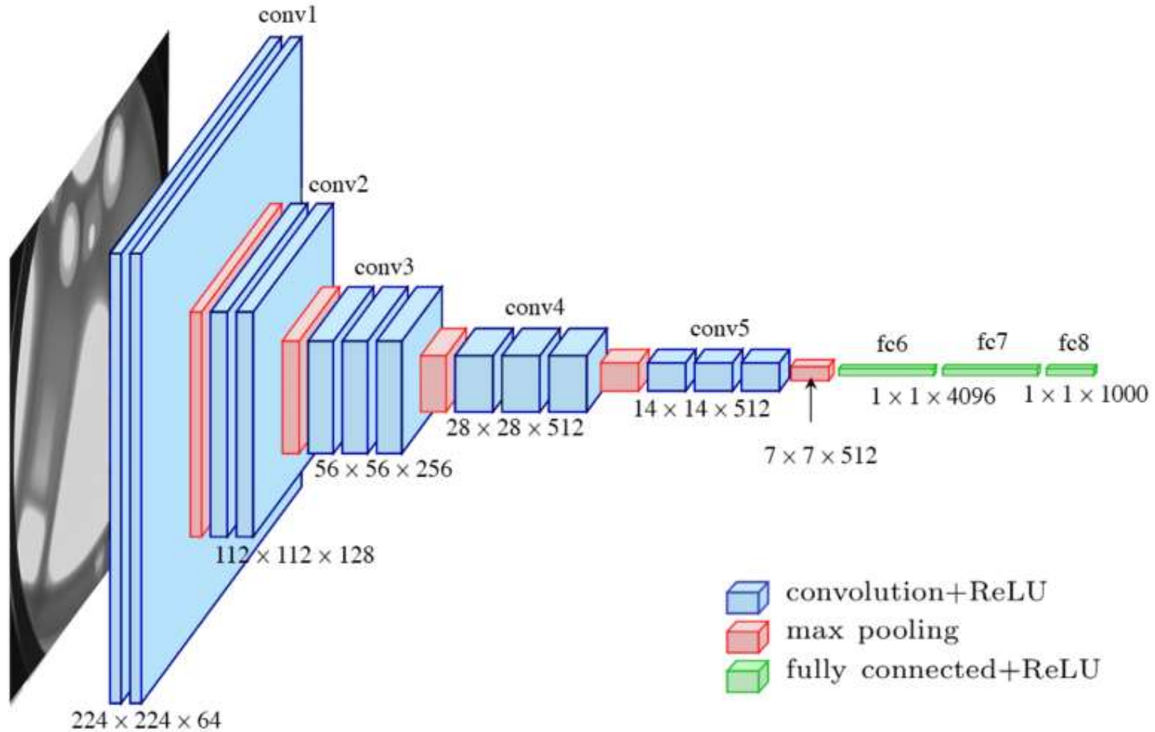
Depth: With its 16 layers, it was among the deeper convolutional neural networks when introduced, allowing it to learn intricate features from images.

Small Filters: Using 3x3 convolutional filters throughout the network allowed the model to learn more complex patterns and feature representations.

Simple Architecture: The VGG16 architecture is straightforward, comprising a series of stacked layers, making it easier to understand and implement.

The VGG16 model was pre-trained on the ImageNet dataset, which consists of millions of labeled images across thousands of categories. It achieved notable success in the ImageNet Large Scale Visual Recognition Challenge, showcasing its ability to classify images into a wide range of categories with high accuracy.

Researchers and practitioners often use the VGG16 model as a feature extractor or as a starting point for transfer learning in computer vision tasks. While newer architectures have since been developed, the VGG16 model remains a significant benchmark and serves as a foundation for understanding deeper neural network structures in image recognition and classification tasks.



Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sn
import skimage.io
import os
import tqdm
import glob
import tensorflow

from tqdm import tqdm
from sklearn.utils import shuffle
from sklearn import metrics
from sklearn.metrics import confusion_matrix, classification_report
from sklearn.model_selection import train_test_split

from skimage.io import imread, imshow
from skimage.transform import resize
from skimage.color import grey2rgb

import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.preprocessing import image_dataset_from_directory
```

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import InputLayer, BatchNormalization, Dropout, Flatten, Dense, Activation, MaxPool2D, Conv2D
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.utils import to_categorical
from keras import optimizers
from tensorflow.keras.optimizers import Adam

from keras.callbacks import Callback,ModelCheckpoint,ReduceLROnPlateau
from keras.models import Sequential,load_model
from keras.layers import Dense, Dropout
from keras.wrappers.scikit_learn import KerasClassifier
import keras.backend as K

#import tensorflow_addons as tfa
#from tensorflow.keras.metrics import Metric
#from tensorflow_addons.utils.types import AcceptableDTypes, FloatTensorLike
from typeguard import typechecked
from typing import Optional
```

In [2]:
```
AUTOTUNE = tf.data.experimental.AUTOTUNE
```

In [3]:
```
train_datagen = ImageDataGenerator(rescale = 1./255,
                                   validation_split = 0.2,

        rotation_range=5,
        width_shift_range=0.2,
        height_shift_range=0.2,
        shear_range=0.2,
        #zoom_range=0.2,
        horizontal_flip=True,
        vertical_flip=True,
        fill_mode='nearest')

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

test_datagen  = ImageDataGenerator(rescale = 1./255
                                   )
```

In [4]:
```
train_dataset  = train_datagen.flow_from_directory(directory = '../input/chest-ctscan-images/Data/train',
                                                   target_size = (224,224),
                                                   class_mode = 'categorical',
                                                   batch_size = 64)
```

```
Found 613 images belonging to 4 classes.
```

In [5]:
```
valid_dataset = valid_datagen.flow_from_directory(directory = '../input/chest-ctscan-images/Data/valid',
                                                  target_size = (224,224),
                                                  class_mode = 'categorical',
                                                  batch_size = 64)
```

```
Found 72 images belonging to 4 classes.
```

In [6]:
```
test_dataset = test_datagen.flow_from_directory(directory = '../input/chest-ctscan-images/Data/test',
                                                target_size = (224,224),
                                                class_mode = 'categorical',
                                                batch_size = 64)
```

```
Found 315 images belonging to 4 classes.
```

In [7]:
```
base_model = tf.keras.applications.VGG16(input_shape=(224,224,3),include_top=False,weights="imagenet")
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5
58892288/58889256 [==============================] - 2s 0us/step
```

In [8]:
```
# Freezing Layers
for layer in base_model.layers[:-8]:
    layer.trainable=False
```

In [9]:
```
# Building Model
model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(32,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dense(4,activation='softmax'))
```

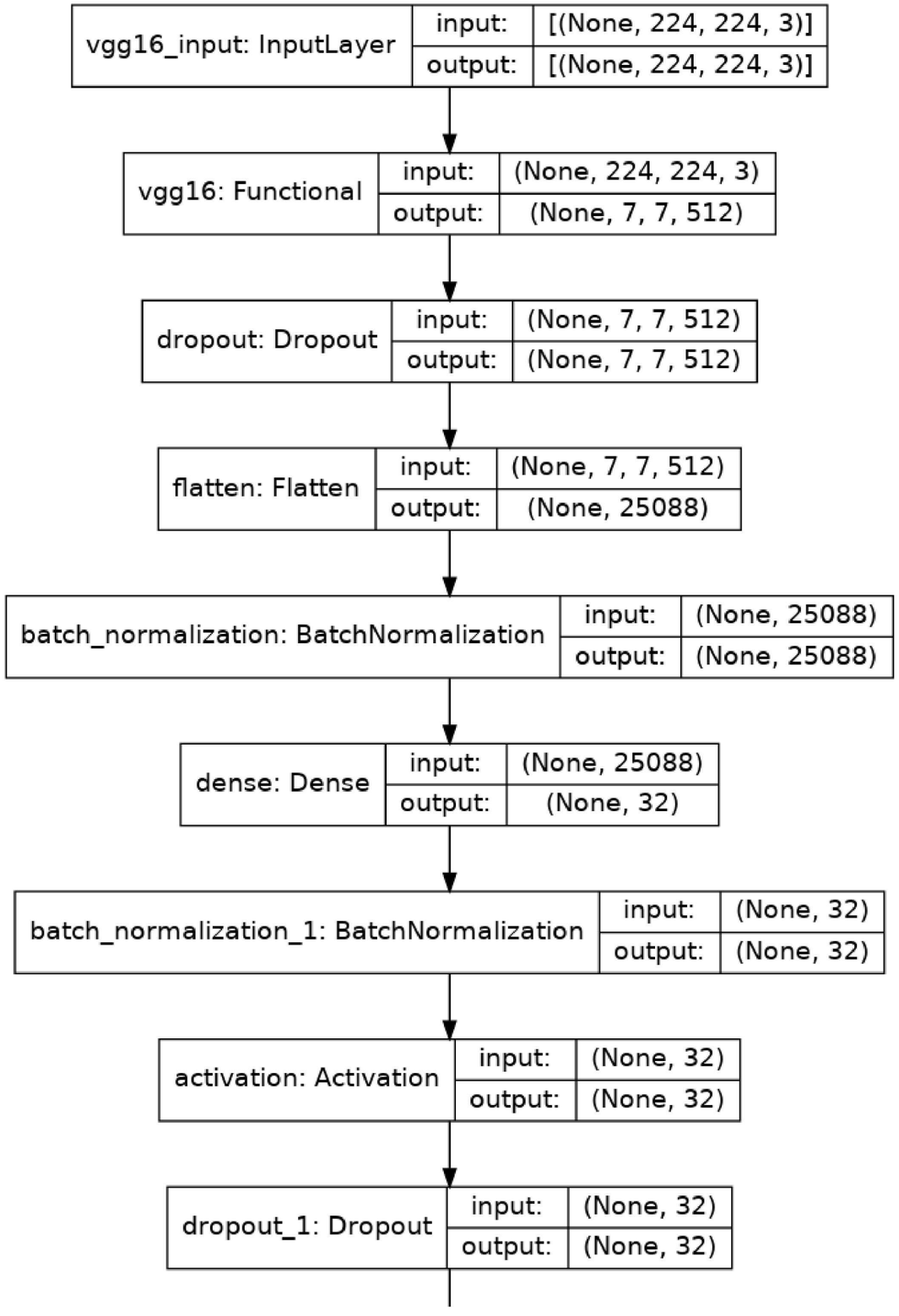
In [10]:
```
# Model Summary
model.summary()
```

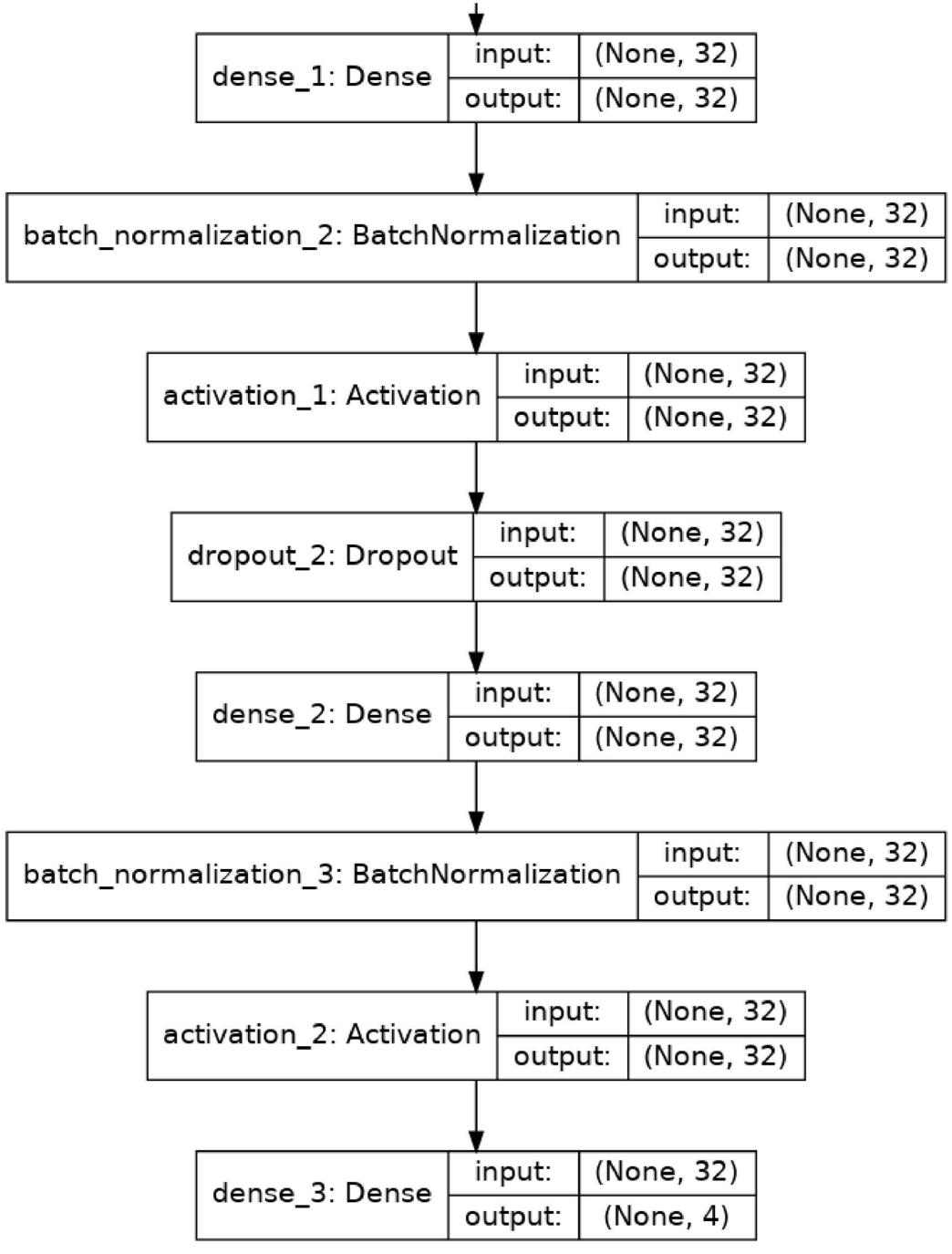
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 7, 7, 512)         14714688
_________________________________________________________________
dropout (Dropout)            (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
batch_normalization (BatchNo (None, 25088)             100352
_________________________________________________________________
dense (Dense)                (None, 32)                802848
_________________________________________________________________
batch_normalization_1 (Batch (None, 32)                128
_________________________________________________________________
activation (Activation)      (None, 32)                0
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_1 (Dense)              (None, 32)                1056
_________________________________________________________________
batch_normalization_2 (Batch (None, 32)                128
_________________________________________________________________
activation_1 (Activation)    (None, 32)                0
_________________________________________________________________
dropout_2 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_2 (Dense)              (None, 32)                1056
_________________________________________________________________
batch_normalization_3 (Batch (None, 32)                128
_________________________________________________________________
activation_2 (Activation)    (None, 32)                0
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 132
=================================================================
Total params: 15,620,516
Trainable params: 13,834,660
Non-trainable params: 1,785,856
_________________________________________________________________
```

In [11]:

```
from tensorflow.keras.utils import plot_model
from IPython.display import Image
plot_model(model, to_file='convnet.png', show_shapes=True,show_layer_names=True)
Image(filename='convnet.png')
```

Out[11]:


vgg16_input: InputLayer
input: [(None, 224, 224, 3)]
output: [(None, 224, 224, 3)]
vgg16: Functional
input: (None, 224, 224, 3)
output: (None, 7, 7, 512)
dropout: Dropout
input: (None, 7, 7, 512)
output: (None, 7, 7, 512)
flatten: Flatten
input: (None, 7, 7, 512)
output: (None, 25088)
batch_normalization: BatchNormalization
input: (None, 25088)
output: (None, 25088)
dense: Dense
input: (None, 25088)
output: (None, 32)
batch_normalization_1: BatchNormalization
input: (None, 32)
output: (None, 32)
activation: Activation
input: (None, 32)
output: (None, 32)
dropout_1: Dropout
input: (None, 32)
output: (None, 32)



In [12]:
```
def f1_score(y_true, y_pred): #taken from old keras source code
    true_positives = K.sum(K.round(K.clip(y_true * y_pred, 0, 1)))
    possible_positives = K.sum(K.round(K.clip(y_true, 0, 1)))
    predicted_positives = K.sum(K.round(K.clip(y_pred, 0, 1)))
    precision = true_positives / (predicted_positives + K.epsilon())
```

```
    recall = true_positives / (possible_positives + K.epsilon())
    f1_val = 2*(precision*recall)/(precision+recall+K.epsilon())
    return f1_val
```

In [13]:
```
METRICS = [
      tf.keras.metrics.BinaryAccuracy(name='accuracy'),
      tf.keras.metrics.Precision(name='precision'),
      tf.keras.metrics.Recall(name='recall'),
      tf.keras.metrics.AUC(name='auc'),
        f1_score,
]
```

In [14]:
```
lrd = ReduceLROnPlateau(monitor = 'val_loss',patience = 3,verbose = 1,factor = 0.50, min_lr = 1e-7)

mcp = ModelCheckpoint('model.h5')

es = EarlyStopping(verbose=1, patience=3)
```

In [15]:
```
model.compile(optimizer='Adam', loss='categorical_crossentropy',metrics=METRICS)
```

In [16]:
```
%time
history=model.fit(train_dataset,validation_data=valid_dataset,epochs = 20,verbose = 1,callbacks=[lrd,mcp,es])
```

```
CPU times: user 3 µs, sys: 1e+03 ns, total: 4 µs
Wall time: 7.39 µs
Epoch 1/20
10/10 [==============================] - 309s 31s/step - loss: 1.7041 - accuracy: 0.7101 - precision: 0.2287 - recall: 0.0684 - auc: 0.4636 - f1_score: 0.1010 - val_loss: 153.2695 - val_accuracy: 0.59
03 - val_precision: 0.1806 - val_recall: 0.1806 - val_auc: 0.4537 - val_f1_score: 0.1562
Epoch 2/20
10/10 [==============================] - 305s 30s/step - loss: 1.5212 - accuracy: 0.7219 - precision: 0.3100 - recall: 0.0926 - auc: 0.5569 - f1_score: 0.1433 - val_loss: 42.2709 - val_accuracy: 0.604
2 - val_precision: 0.2083 - val_recall: 0.2083 - val_auc: 0.4622 - val_f1_score: 0.2266
Epoch 3/20
10/10 [==============================] - 303s 30s/step - loss: 1.4527 - accuracy: 0.7423 - precision: 0.4320 - recall: 0.0981 - auc: 0.5723 - f1_score: 0.1579 - val_loss: 31.7001 - val_accuracy: 0.611
1 - val_precision: 0.2222 - val_recall: 0.2222 - val_auc: 0.4706 - val_f1_score: 0.2891
Epoch 4/20
10/10 [==============================] - 303s 30s/step - loss: 1.3599 - accuracy: 0.7470 - precision: 0.4734 - recall: 0.1062 - auc: 0.6285 - f1_score: 0.1744 - val_loss: 19.4024 - val_accuracy: 0.645
8 - val_precision: 0.2917 - val_recall: 0.2917 - val_auc: 0.5317 - val_f1_score: 0.3281
Epoch 5/20
10/10 [==============================] - 303s 30s/step - loss: 1.3010 - accuracy: 0.7600 - precision: 0.5866 - recall: 0.1344 - auc: 0.6597 - f1_score: 0.2181 - val_loss: 22.6077 - val_accuracy: 0.590
3 - val_precision: 0.1806 - val_recall: 0.1806 - val_auc: 0.4537 - val_f1_score: 0.1562
Epoch 6/20
10/10 [==============================] - 303s 30s/step - loss: 1.2935 - accuracy: 0.7602 - precision: 0.5918 - recall: 0.1304 - auc: 0.6491 - f1_score: 0.2150 - val_loss: 19.7089 - val_accuracy: 0.590
3 - val_precision: 0.1806 - val_recall: 0.1806 - val_auc: 0.4537 - val_f1_score: 0.1562
Epoch 7/20
10/10 [==============================] - 303s 30s/step - loss: 1.2905 - accuracy: 0.7649 - precision: 0.6300 - recall: 0.1450 - auc: 0.6524 - f1_score: 0.2313 - val_loss: 15.0300 - val_accuracy: 0.590
3 - val_precision: 0.1806 - val_recall: 0.1806 - val_auc: 0.4537 - val_f1_score: 0.1562
Epoch 8/20
10/10 [==============================] - 302s 32s/step - loss: 1.1919 - accuracy: 0.7687 - precision: 0.6600 - recall: 0.1556 - auc: 0.7251 - f1_score: 0.2528 - val_loss: 11.7119 - val_accuracy: 0.590
3 - val_precision: 0.1806 - val_recall: 0.1806 - val_auc: 0.4541 - val_f1_score: 0.1562
Epoch 9/20
10/10 [==============================] - 302s 32s/step - loss: 1.1857 - accuracy: 0.7712 - precision: 0.6748 - recall: 0.1631 - auc: 0.7373 - f1_score: 0.2685 - val_loss: 6.2872 - val_accuracy: 0.5903
- val_precision: 0.1806 - val_recall: 0.1806 - val_auc: 0.4662 - val_f1_score: 0.1562
Epoch 10/20
10/10 [==============================] - 303s 30s/step - loss: 1.1434 - accuracy: 0.7815 - precision: 0.7459 - recall: 0.1914 - auc: 0.7563 - f1_score: 0.3037 - val_loss: 5.2349 - val_accuracy: 0.5938
- val_precision: 0.1831 - val_recall: 0.1806 - val_auc: 0.5055 - val_f1_score: 0.1604
Epoch 11/20
10/10 [==============================] - 303s 30s/step - loss: 1.0856 - accuracy: 0.7860 - precision: 0.7945 - recall: 0.1948 - auc: 0.7854 - f1_score: 0.3130 - val_loss: 2.7747 - val_accuracy: 0.6250
- val_precision: 0.2097 - val_recall: 0.1806 - val_auc: 0.5588 - val_f1_score: 0.1092
Epoch 12/20
10/10 [==============================] - 303s 30s/step - loss: 1.1397 - accuracy: 0.7797 - precision: 0.7242 - recall: 0.1919 - auc: 0.7608 - f1_score: 0.3040 - val_loss: 1.2762 - val_accuracy: 0.7465
- val_precision: 0.4839 - val_recall: 0.2083 - val_auc: 0.6816 - val_f1_score: 0.2372
Epoch 13/20
10/10 [==============================] - 303s 30s/step - loss: 1.0498 - accuracy: 0.7951 - precision: 0.8036 - recall: 0.2390 - auc: 0.7954 - f1_score: 0.3698 - val_loss: 8.2673 - val_accuracy: 0.6181
- val_precision: 0.2361 - val_recall: 0.2361 - val_auc: 0.4698 - val_f1_score: 0.2422
Epoch 14/20
10/10 [==============================] - 306s 31s/step - loss: 1.0725 - accuracy: 0.7852 - precision: 0.7819 - recall: 0.1951 - auc: 0.7944 - f1_score: 0.3108 - val_loss: 27.8674 - val_accuracy: 0.604
2 - val_precision: 0.2083 - val_recall: 0.2083 - val_auc: 0.4527 - val_f1_score: 0.2812
Epoch 15/20
10/10 [==============================] - 302s 32s/step - loss: 1.0424 - accuracy: 0.7914 - precision: 0.8053 - recall: 0.2191 - auc: 0.8067 - f1_score: 0.3425 - val_loss: 65.4412 - val_accuracy: 0.590
3 - val_precision: 0.1806 - val_recall: 0.1806 - val_auc: 0.4537 - val_f1_score: 0.2109

Epoch 00015: ReduceLROnPlateau reducing learning rate to 0.0005000000237487257.
Epoch 00015: early stopping
```

In [17]:
```
model.evaluate(test_dataset, verbose=1)
```

Out[17]:
```
5/5 [==============================] - 78s 15s/step - loss: 89.7240 - accuracy: 0.5857 - precision: 0.1714 - recall: 0.1714 - auc: 0.4476 - f1_score: 0.1706
[89.72400665283203,
 0.5857142806053162,
 0.17142857611179352,
 0.17142857611179352,
 0.4476190209388733,
 0.17060375213623047]
```

In [18]:

```
#%% PLOTTING RESULTS (Train vs Validation FOLDER 1)

def Train_Val_Plot(acc,val_acc,loss,val_loss,auc,val_auc,precision,val_precision,f1,val_f1):

    fig, (ax1, ax2,ax3,ax4,ax5) = plt.subplots(1,5, figsize= (20,5))
    fig.suptitle(" MODEL'S METRICS VISUALIZATION ")

    ax1.plot(range(1, len(acc) + 1), acc)
    ax1.plot(range(1, len(val_acc) + 1), val_acc)
    ax1.set_title('History of Accuracy')
    ax1.set_xlabel('Epochs')
    ax1.set_ylabel('Accuracy')
    ax1.legend(['training', 'validation'])


    ax2.plot(range(1, len(loss) + 1), loss)
    ax2.plot(range(1, len(val_loss) + 1), val_loss)
    ax2.set_title('History of Loss')
    ax2.set_xlabel('Epochs')
    ax2.set_ylabel('Loss')
    ax2.legend(['training', 'validation'])

    ax3.plot(range(1, len(auc) + 1), auc)
    ax3.plot(range(1, len(val_auc) + 1), val_auc)
    ax3.set_title('History of AUC')
    ax3.set_xlabel('Epochs')
    ax3.set_ylabel('AUC')
    ax3.legend(['training', 'validation'])

    ax4.plot(range(1, len(precision) + 1), precision)
    ax4.plot(range(1, len(val_precision) + 1), val_precision)
    ax4.set_title('History of Precision')
    ax4.set_xlabel('Epochs')
    ax4.set_ylabel('Precision')
    ax4.legend(['training', 'validation'])

    ax5.plot(range(1, len(f1) + 1), f1)
    ax5.plot(range(1, len(val_f1) + 1), val_f1)
    ax5.set_title('History of F1-score')
    ax5.set_xlabel('Epochs')
    ax5.set_ylabel('F1 score')
    ax5.legend(['training', 'validation'])


    plt.show()


Train_Val_Plot(history.history['accuracy'],history.history['val_accuracy'],
               history.history['loss'],history.history['val_loss'],
               history.history['auc'],history.history['val_auc'],
               history.history['precision'],history.history['val_precision'],
               history.history['f1_score'],history.history['val_f1_score']
              )
```

MODEL'S METRICS VISUALIZATION