In [2]:

```
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, Der
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
from tensorflow.keras.applications.vgg16 import VGG16 # VGG16
from tensorflow.keras.applications.vgg19 import VGG19 # VGG19
from tensorflow.keras.applications.resnet50 import ResNet50 # ResNet50
from tensorflow.keras.applications.xception import Xception # Xception
from tensorflow.keras.applications.mobilenet import MobileNet # MobileNet
from tensorflow.keras.applications.nasnet import NASNetMobile # NASNetMobile
from tensorflow.keras.applications.densenet import DenseNet169 # DenseNet169
from tensorflow.keras.applications.densenet import DenseNet121 # DenseNet121
from tensorflow.keras.applications.mobilenet_v2 import MobileNetV2 # MobileNetV2
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.utils import to_categorical
from keras import optimizers

from keras.callbacks import Callback,ModelCheckpoint
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 [3]:

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
```

In [4]:

```
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 [5]:

```
train_dataset  = train_datagen.flow_from_directory(directory = 'Alzheimer_s Dataset/trair
                                                   target_size = (224,224),
                                                   class_mode = 'categorical',
                                                   subset = 'training',
                                                   batch_size = 32)
```

Found 4098 images belonging to 4 classes.

In [6]:

```
valid_dataset = valid_datagen.flow_from_directory(directory = 'Alzheimer_s Dataset/train'
                                                  target_size = (224,224),
                                                  class_mode = 'categorical',
                                                  subset = 'validation',
                                                  batch_size = 32)
```

Found 1023 images belonging to 4 classes.

In [7]:

```
test_dataset = test_datagen.flow_from_directory(directory ='Alzheimer_s Dataset/test',
                                                  target_size = (224,224),
                                                  class_mode = 'categorical',
                                                  batch_size = 32)
```

Found 1279 images belonging to 4 classes.

In [8]:

```
base_model = DenseNet121(input_shape=(224,224,3),
                   include_top=False,
                   weights="imagenet")
```

Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5 (https://storage.googleapis.com/tensorflow/keras-applications/densenet/densenet121_weights_tf_dim_ordering_tf_kernels_notop.h5)
29084464/29084464 [==============================] - 4s 0us/step

In [9]:

```
for layer in base_model.layers:
    layer.trainable=False
```

In [10]:

```
model=Sequential()
model.add(base_model)
model.add(Dropout(0.5))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(64,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(64,kernel_initializer='he_uniform'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(64,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 [11]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 densenet121 (Functional)    (None, 7, 7, 1024)        7037504

 dropout (Dropout)           (None, 7, 7, 1024)        0

 flatten (Flatten)           (None, 50176)             0

 batch_normalization (BatchN  (None, 50176)            200704
 ormalization)

 dense (Dense)               (None, 64)                3211328

 batch_normalization_1 (Batc  (None, 64)               256
 hNormalization)

 activation (Activation)     (None, 64)                0

 dropout_1 (Dropout)         (None, 64)                0

 dense_1 (Dense)             (None, 64)                4160

 batch_normalization_2 (Batc  (None, 64)               256
 hNormalization)

 activation_1 (Activation)   (None, 64)                0

 dropout_2 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 64)                4160

 batch_normalization_3 (Batc  (None, 64)               256
 hNormalization)

 activation_2 (Activation)   (None, 64)                0

 dropout_3 (Dropout)         (None, 64)                0

 dense_3 (Dense)             (None, 32)                2080

 batch_normalization_4 (Batc  (None, 32)               128
 hNormalization)

 activation_3 (Activation)   (None, 32)                0

 dropout_4 (Dropout)         (None, 32)                0

 dense_4 (Dense)             (None, 32)                1056

 batch_normalization_5 (Batc  (None, 32)               128
 hNormalization)

 activation_4 (Activation)   (None, 32)                0

 dense_5 (Dense)             (None, 4)                 132

=================================================================
Total params: 10,462,148
Trainable params: 3,323,780
```

```
Non-trainable params: 7,138,368
_________________________________________________________________
```

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]:

```
def exponential_decay(lr0, s):
    def exponential_decay_fn(epoch):
        return lr0 * 0.1 **(epoch / s)
    return exponential_decay_fn

exponential_decay_fn = exponential_decay(0.01, 5) # when i run it for 50 epochs

lr_scheduler = tf.keras.callbacks.LearningRateScheduler(exponential_decay_fn)
```

In [15]:

```
model.compile(optimizer='rmsprop', loss='categorical_crossentropy',metrics=METRICS)
```

In [67]:

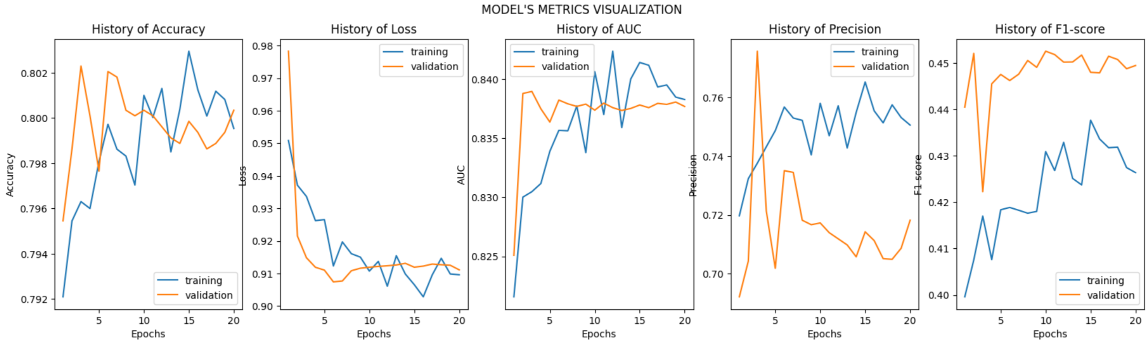
```
history=model.fit(train_dataset,
                        validation_data=valid_dataset,
                        epochs = 20,
                        verbose = 1,
                         callbacks=lr_scheduler)
```

```
Epoch 1/20
129/129 [==============================] - 240s 2s/step - loss: 0.9508 - a
ccuracy: 0.7921 - precision: 0.7197 - recall: 0.2757 - auc: 0.8216 - f1_sc
ore: 0.3996 - val_loss: 0.9783 - val_accuracy: 0.7955 - val_precision: 0.6
921 - val_recall: 0.3275 - val_auc: 0.8251 - val_f1_score: 0.4405 - lr: 0.
0100
Epoch 2/20
129/129 [==============================] - 280s 2s/step - loss: 0.9371 - a
ccuracy: 0.7954 - precision: 0.7324 - recall: 0.2865 - auc: 0.8300 - f1_sc
ore: 0.4075 - val_loss: 0.9215 - val_accuracy: 0.7986 - val_precision: 0.7
043 - val_recall: 0.3353 - val_auc: 0.8388 - val_f1_score: 0.4521 - lr: 0.
0063
Epoch 3/20
129/129 [==============================] - 263s 2s/step - loss: 0.9337 - a
ccuracy: 0.7963 - precision: 0.7376 - recall: 0.2875 - auc: 0.8305 - f1_sc
ore: 0.4170 - val_loss: 0.9148 - val_accuracy: 0.8023 - val_precision: 0.7
758 - val_recall: 0.2942 - val_auc: 0.8390 - val_f1_score: 0.4223 - lr: 0.
0040
Epoch 4/20
129/129 [==============================] - 247s 2s/step - loss: 0.9262 - a
ccuracy: 0.7960 - precision: 0.7432 - recall: 0.2811 - auc: 0.8312 - f1_sc
ore: 0.4076 - val_loss: 0.9119 - val_accuracy: 0.8001 - val_precision: 0.7
214 - val_recall: 0.3265 - val_auc: 0.8375 - val_f1_score: 0.4456 - lr: 0.
0025
Epoch 5/20
129/129 [==============================] - 249s 2s/step - loss: 0.9266 - a
ccuracy: 0.7981 - precision: 0.7487 - recall: 0.2894 - auc: 0.8339 - f1_sc
ore: 0.4184 - val_loss: 0.9110 - val_accuracy: 0.7977 - val_precision: 0.7
019 - val_recall: 0.3314 - val_auc: 0.8364 - val_f1_score: 0.4476 - lr: 0.
0016
Epoch 6/20
129/129 [==============================] - 250s 2s/step - loss: 0.9123 - a
ccuracy: 0.7997 - precision: 0.7568 - recall: 0.2931 - auc: 0.8357 - f1_sc
ore: 0.4189 - val_loss: 0.9074 - val_accuracy: 0.8021 - val_precision: 0.7
351 - val_recall: 0.3255 - val_auc: 0.8382 - val_f1_score: 0.4463 - lr: 0.
0010
Epoch 7/20
129/129 [==============================] - 251s 2s/step - loss: 0.9197 - a
ccuracy: 0.7986 - precision: 0.7530 - recall: 0.2894 - auc: 0.8356 - f1_sc
ore: 0.4183 - val_loss: 0.9077 - val_accuracy: 0.8018 - val_precision: 0.7
345 - val_recall: 0.3245 - val_auc: 0.8379 - val_f1_score: 0.4476 - lr: 6.
3096e-04
Epoch 8/20
129/129 [==============================] - 251s 2s/step - loss: 0.9161 - a
ccuracy: 0.7983 - precision: 0.7522 - recall: 0.2882 - auc: 0.8377 - f1_sc
ore: 0.4176 - val_loss: 0.9108 - val_accuracy: 0.8003 - val_precision: 0.7
182 - val_recall: 0.3314 - val_auc: 0.8377 - val_f1_score: 0.4506 - lr: 3.
9811e-04
Epoch 9/20
129/129 [==============================] - 248s 2s/step - loss: 0.9150 - a
ccuracy: 0.7970 - precision: 0.7405 - recall: 0.2897 - auc: 0.8338 - f1_sc
ore: 0.4180 - val_loss: 0.9116 - val_accuracy: 0.8001 - val_precision: 0.7
167 - val_recall: 0.3314 - val_auc: 0.8379 - val_f1_score: 0.4491 - lr: 2.
5119e-04
Epoch 10/20
129/129 [==============================] - 249s 2s/step - loss: 0.9107 - a
ccuracy: 0.8010 - precision: 0.7580 - recall: 0.2997 - auc: 0.8406 - f1_sc
ore: 0.4309 - val_loss: 0.9119 - val_accuracy: 0.8003 - val_precision: 0.7
173 - val_recall: 0.3324 - val_auc: 0.8374 - val_f1_score: 0.4526 - lr: 1.
5849e-04
Epoch 11/20
```

```
129/129 [==============================] - 251s 2s/step - loss: 0.9137 - a
ccuracy: 0.8000 - precision: 0.7470 - recall: 0.3026 - auc: 0.8370 - f1_sc
ore: 0.4268 - val_loss: 0.9122 - val_accuracy: 0.8001 - val_precision: 0.7
140 - val_recall: 0.3343 - val_auc: 0.8380 - val_f1_score: 0.4519 - lr: 1.
0000e-04
Epoch 12/20
129/129 [==============================] - 249s 2s/step - loss: 0.9061 - a
ccuracy: 0.8013 - precision: 0.7572 - recall: 0.3021 - auc: 0.8424 - f1_sc
ore: 0.4329 - val_loss: 0.9124 - val_accuracy: 0.7996 - val_precision: 0.7
119 - val_recall: 0.3333 - val_auc: 0.8376 - val_f1_score: 0.4502 - lr: 6.
3096e-05
Epoch 13/20
129/129 [==============================] - 248s 2s/step - loss: 0.9155 - a
ccuracy: 0.7985 - precision: 0.7428 - recall: 0.2967 - auc: 0.8359 - f1_sc
ore: 0.4251 - val_loss: 0.9126 - val_accuracy: 0.7991 - val_precision: 0.7
098 - val_recall: 0.3324 - val_auc: 0.8373 - val_f1_score: 0.4503 - lr: 3.
9811e-05
Epoch 14/20
129/129 [==============================] - 248s 2s/step - loss: 0.9098 - a
ccuracy: 0.8004 - precision: 0.7549 - recall: 0.2984 - auc: 0.8400 - f1_sc
ore: 0.4237 - val_loss: 0.9131 - val_accuracy: 0.7989 - val_precision: 0.7
058 - val_recall: 0.3353 - val_auc: 0.8375 - val_f1_score: 0.4518 - lr: 2.
5119e-05
Epoch 15/20
129/129 [==============================] - 248s 2s/step - loss: 0.9066 - a
ccuracy: 0.8030 - precision: 0.7653 - recall: 0.3055 - auc: 0.8414 - f1_sc
ore: 0.4377 - val_loss: 0.9119 - val_accuracy: 0.7999 - val_precision: 0.7
143 - val_recall: 0.3324 - val_auc: 0.8378 - val_f1_score: 0.4480 - lr: 1.
5849e-05
Epoch 16/20
129/129 [==============================] - 248s 2s/step - loss: 0.9028 - a
ccuracy: 0.8012 - precision: 0.7555 - recall: 0.3031 - auc: 0.8412 - f1_sc
ore: 0.4337 - val_loss: 0.9123 - val_accuracy: 0.7994 - val_precision: 0.7
113 - val_recall: 0.3324 - val_auc: 0.8376 - val_f1_score: 0.4479 - lr: 1.
0000e-05
Epoch 17/20
129/129 [==============================] - 248s 2s/step - loss: 0.9096 - a
ccuracy: 0.8001 - precision: 0.7514 - recall: 0.2994 - auc: 0.8393 - f1_sc
ore: 0.4318 - val_loss: 0.9129 - val_accuracy: 0.7986 - val_precision: 0.7
052 - val_recall: 0.3343 - val_auc: 0.8379 - val_f1_score: 0.4515 - lr: 6.
3096e-06
Epoch 18/20
129/129 [==============================] - 249s 2s/step - loss: 0.9147 - a
ccuracy: 0.8012 - precision: 0.7575 - recall: 0.3011 - auc: 0.8395 - f1_sc
ore: 0.4319 - val_loss: 0.9127 - val_accuracy: 0.7989 - val_precision: 0.7
049 - val_recall: 0.3363 - val_auc: 0.8379 - val_f1_score: 0.4508 - lr: 3.
9811e-06
Epoch 19/20
129/129 [==============================] - 250s 2s/step - loss: 0.9099 - a
ccuracy: 0.8008 - precision: 0.7532 - recall: 0.3023 - auc: 0.8385 - f1_sc
ore: 0.4275 - val_loss: 0.9125 - val_accuracy: 0.7994 - val_precision: 0.7
087 - val_recall: 0.3353 - val_auc: 0.8381 - val_f1_score: 0.4488 - lr: 2.
5119e-06
Epoch 20/20
129/129 [==============================] - 259s 2s/step - loss: 0.9096 - a
ccuracy: 0.7995 - precision: 0.7506 - recall: 0.2967 - auc: 0.8383 - f1_sc
ore: 0.4264 - val_loss: 0.9111 - val_accuracy: 0.8003 - val_precision: 0.7
182 - val_recall: 0.3314 - val_auc: 0.8377 - val_f1_score: 0.4495 - lr: 1.
5849e-06
```

In [68]:

```python
def Train_Val_Plot(acc,val_acc,loss,val_loss,auc,val_auc,precision,val_precision,f1,val_f

    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']
              )
```



In [70]:

```
scores = model.evaluate_generator(test_dataset)
```

```
C:\Users\aniru\AppData\Local\Temp\ipykernel_19156\39297891.py:1: UserWarni
ng: `Model.evaluate_generator` is deprecated and will be removed in a futu
re version. Please use `Model.evaluate`, which supports generators.
  scores = model.evaluate_generator(test_dataset)
```

In [71]:

```
print("Accuracy = ", scores[1])
print("Precision = ", scores[2])
print("Recall = ", scores[3])
print("AUC = ", scores[4])
print("F1_score = ", scores[5])
```

```
Accuracy =  0.7871384024620056
Precision =  0.6217948794364929
Recall =  0.3792025148868561
AUC =  0.8278332352638245
F1_score =  0.4697352945804596
```

In [69]:

```
model.save('model_01.h5')
```

Loading and predicting with the model

In [10]:

```
import tensorflow
from tensorflow.keras.preprocessing.image import load_img, img_to_array
import numpy as np
from sklearn.metrics import f1_score
```

In [15]:

```
def f1_score_metric(y_true, y_pred):
    # Implement the F1 score calculation here
    return f1_score(y_true, y_pred)

# Register the custom metric function
tf.keras.utils.get_custom_objects()['f1_score'] = f1_score_metric
```

In [16]:

```
prac_model = tensorflow.keras.models.load_model('model_01.h5')
```

In [44]:

```
input_image = load_img('moderateDem36.jpg', target_size=(224, 224))  # Adjust target_size
input_image = input_image.convert('RGB')
# Convert the image to an array
input_array = img_to_array(input_image)

# Rescale the pixel values to the range of 0-1
input_array /= 255.0

# Expand dimensions to match the expected input shape of the model
input_array = np.expand_dims(input_array, axis=0)
```

In [45]:

```
print('Preprocessed Input Shape:', input_array.shape)
```

```
Preprocessed Input Shape: (1, 224, 224, 3)
```

In [46]:

```
predictions = prac_model.predict(input_array)

# Interpret the prediction results
# Example: Assuming the model has 4 output classes
class_labels = ['Class 1', 'Class 2', 'Class 3', 'Class 4']
predicted_class = np.argmax(predictions, axis=1)
predicted_label = class_labels[predicted_class[0]]
print('Predictions : ', predictions)
print('Predicted Class:', predicted_class)
print('Predicted Label:', predicted_label)
```

```
1/1 [==============================] - 0s 98ms/step
Predictions :  [[0.24433692 0.0205209  0.3022289  0.4329133 ]]
Predicted Class: [3]
Predicted Label: Class 4
```

In [ ]:

```

```