Classification of PCOS/PCOD Using Transfer Learning and GAN Architectures to Generate Pseudo Ultrasound Images

MSc Research Project
Msc Data Analytics

Sweta Kumari
Student ID: x19240848

School of Computing
National College of Ireland

Supervisor: Dr.Majid Latifi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Sweta Kumari
Student ID:	x19240848
Programme:	Msc Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Dr.Majid Latifi
Submission Due Date:	16/08/2021
Project Title:	Classification of PCOS/PCOD Using Transfer Learning and GAN Architectures to Generate Pseudo Ultrasound Images
Word Count:	898
Page Count:	11

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	16th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Hardware and Software pre-requisite configuration:

The tools and software requirements used for this thesis research work can be installed on a laptop or a PC. The basic configuration list is given below:

Operating system	Windows 10
RAM	16 GB
Hard Disk	512 SSD
Processor	Core i7 8th gen

Pre-Requisites:

The basic toolset used in this research work for carrying out all the actions are listed below:

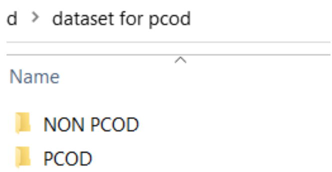
- Microsoft office tools
- Python 3.7
- Jupyter

Dataset Gathering:

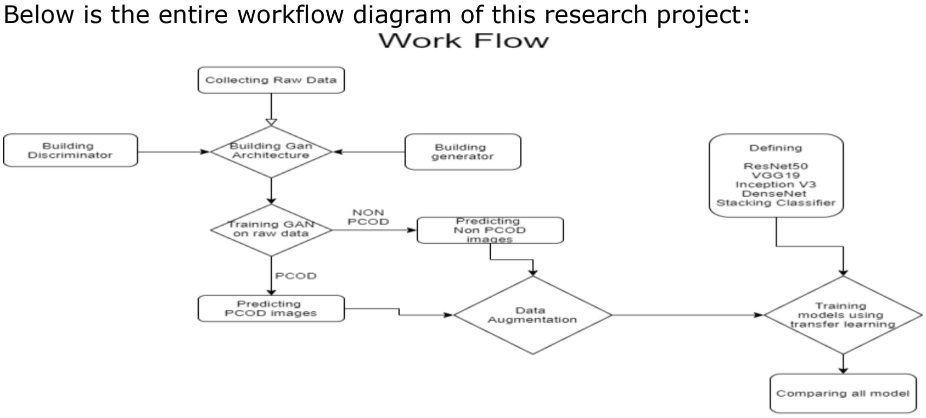
PCOS/PCOD Dataset:

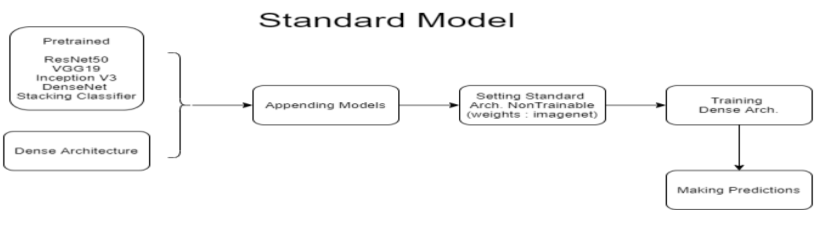
Initially it was fetched from Kaggle but images were very less approximately to 34 hence, other web sources were collected and then I myself created dataset for approximately of 94 images with proper labelling into PCOD and NON PCOD Category:

https://pubs.rsna.org/doi/10.1148/rg.326125503
https://radiopaedia.org/articles/polycystic-ovarian-syndrome-1
https://medpix.nlm.nih.gov/search?allen=true&allt=true&alli=true&query=pcos
https://www.intechopen.com/chapters/45102
https://www.degruyter.com/document/doi/10.1515/jpem-2014-0307/html
https://www.nebojsazecevic.com/en/themes/polycystic-ovary-syndrome-pcos/5246/clinical-picture-of-polycystic-ovary-syndrome
https://radiologykey.com/the-ovary-and-polycystic-ovary-syndrome/



Below is the entire workflow diagram of this research project:





As stated in research topic we are first covering GAN implementation:

Section 1:
GAN Model:

Importing the essential libraries to setup the environment

jupyter GanModel Last Checkpoint: Last Friday at 10:20 AM (autosaved)

File Edit View Insert Cell Kernel Widgets Help

Run Code

In [1]:
```python
import os
import cv2
import numpy as np
import matplotlib.pyplot as plt

from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import LeakyReLU
from keras.layers import Flatten
from keras.layers import Reshape
from keras.layers import Dense,Dropout
from keras.layers import Conv2DTranspose
from keras.optimizers import Adam
from keras.models import load_model
from tqdm.notebook import tqdm_notebook
from keras.utils.vis_utils import plot_model
```

In [2]:
```python
plt.style.use('fivethirtyeight')
```

Defining Functions/Models

In [3]:
```python
# Funtion to load real images
def load_real_images(path=None,image_shape=(128,128,3)):
    # list to store image
    images = []
    if path :
        # going through each image and loading them in memory
        image_list = os.listdir(path)
        for image in image_list:
            image_location = path + "/" + image
            # reading image from location
            image = cv2.imread(image_location)
            # resizing image
            image = cv2.resize(image,image_shape[0:2])
            images.append(image)

        images = np.array(images)
        # Clipping images in range [-1,1]
        images = (images-127.5)/127.5

        return images
    else :
        print("No path specified")
```

The above function is to load original images and going through each image and loading them in memory in array then reading each image from location after that resizing image and finally clipping images in range from [-1,1]

As GAN model consist of two components i.e generator and discriminator.

Generator Function :
It's basically a sequential classification model where we are performing down sampling with Conv2D as its convolutional neural network model.Its more like image to array then flatting the array and implement dense neural network with sigmoid activation function. Here we are using adam optimizer using binary_crossentropy loss function .

```python
def discriminator(input_shape = (128,128,3)):
    # Defining a Sequential classification model
    model = Sequential()
    # 1ST layer no downsampling
    model.add(Conv2D(64,(3,3),padding='same',input_shape=input_shape))
    model.add(LeakyReLU(alpha = 0.2))

    #2nd Layer Downsampling
    model.add(Conv2D(128,(3,3),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    # 3rd layer Downsampling
    model.add(Conv2D(256,(3,3),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    #4th Layer DownSampling
    model.add(Conv2D(256,(3,3),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    # Classifier(DNN)
    model.add(Flatten())
    model.add(Dropout(0.4))
    model.add(Dense(1,activation="sigmoid"))

    #Compiling model
    optimizer = Adam(lr=0.0002,beta_1= 0.5)
    model.compile(optimizer=optimizer,loss="binary_crossentropy",metrics=["accuracy"])

    return model
```

Below functions generating random images with random points and also generate real labels for class in output proceeding with generating latent points and reshaping to help the generator

In [4]:

```python
def generate_real_images(data,n_samples):
    # generate random index
    idx= np.random.randint(0,data.shape[0],n_samples)
    # get randomly selected image
    random_image = data[idx]
    # generete real class labels --> 1
    true_label = np.ones((n_samples,1))

    return random_image,true_label
```

In [5]:

```python
# Generating points in latent space for Generator model
def generate_latent_points(latent_dimension,n_samples):
    # Generate points in latent space
    input_points = np.random.randn(latent_dimension * n_samples)
    # reshaping points to pass as input to generator
    input_points = input_points.reshape(n_samples,latent_dimension)

    return input_points
```

In [6]:

```python
def generate_fake_images(generator_model,latent_dimension,n_samples):
    # Generating points in latent dimension
    X_input = generate_latent_points(latent_dimension,n_samples)
    # predict output image using generator
    pred_image = generator_model.predict(X_input)
    # Creating "fake" labels
    fake_labels = np.zeros((n_samples,1))

    return pred_image,fake_labels
```

Generator : This function generator the images where as discriminator distinguish between real and fake images.This function focus on image to array using Conv2D.Transposechannel which means the operation where kernel already learnt the Conv2D. This function is like upsampling means array to image route we can how upsampling done for 16X16, 32X32, 64X64 snd 128X128 as LeakyRelu activation function in discriminator based on tanh here we have same rule

In [8]:

```
# Creating Generator
def generator(latent_dimnesion):
    # Creating sequnetial model
    model = Sequential()
    # setting up foundation for low resolution 4x4 image
    starting_node = 512*4*4
    model.add(Dense(starting_node,input_dim = latent_dimension))
    model.add(LeakyReLU(alpha=0.2))
    model.add(Reshape((4,4,512)))

    # We will be upsampling out low dimension 4x4x256 image to 128x128x3
    # 1st Layer upsampling to 8x8
    #256,256,128,128,128
    model.add(Conv2DTranspose(256,(4,4),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    # 2nd Layer upsampling to 16x16
    model.add(Conv2DTranspose(256,(4,4),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    # 3rd layer Upsampling to 32x32
    model.add(Conv2DTranspose(128,(4,4),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    # 4th layer Upsampling to 64x64
    model.add(Conv2DTranspose(128,(4,4),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    # 5th layer upsampling to 128x128
    model.add(Conv2DTranspose(128,(4,4),strides=(2,2),padding="same"))
    model.add(LeakyReLU(alpha=0.2))

    # Output layer
    model.add(Conv2D(3,(3,3),activation="tanh",padding="same"))

    return model
```

Then we will merge both generator and discriminator keeping discriminator model false as a result it will get biased output hence discirmator will not be trained only Generator will be trained and complie with binary_crossentropy.

In [9]:

```
def generate_gan(generator_model,discriminator_model):
    # Disabling discriminator i.e. non-trainable weights
    discriminator_model.trainable = False

    # Merging Generator and Discriminator Models
    model = Sequential()
    # Add Generator
    model.add(generator_model)
    # Adding Discriminator
    model.add(discriminator_model)
    # Compiling model
    optimizer = Adam(lr=0.0002,beta_1 = 0.5)
    model.compile(optimizer = optimizer,loss="binary_crossentropy",)

    return model
```

Evaluation model is very important steps here this model will give summarized information of discriminator and saving generated images as fake.

```python
def evaluate_model(epoch,generator_model,discriminator_model,data,latent_dimension,n_samples=10):
    # Get real images
    X_real, y_real = generate_real_images(data, n_samples)
    # evaluating discriminator on real examples
    _, acc_real = discriminator_model.evaluate(X_real, y_real, verbose=0)
    # prepare fake samples
    x_fake, y_fake = generate_fake_images(generator_model, latent_dimension, n_samples)
    # evaluate discriminator on fake images
    _, acc_fake = discriminator_model.evaluate(x_fake, y_fake, verbose=0)
    # summarize discriminator performance
    print('>Accuracy real images: %.0f%%, fake images: %.0f%%' % (acc_real*100, acc_fake*100))
    # saving images
    saving_images(x_fake,epoch = epoch)
    # save the generator model tile file
    filename = 'generator_model_%03d.h5' % (epoch+1)
    generator_model.save(filename)
```

Training the model

```python
# Training generator and discriminator
def train(generator_model,discriminator_model,gan_model,data,latent_dimension,num_of_epoch=200,n_batch=1):
    # Defining number of batches per epoch
    batch_per_epoch = int(data.shape[0]/n_batch)
    half_batch = int(batch_per_epoch/2)
    # iteating through each epoch
    for epoch in tqdm_notebook(range(num_of_epoch),desc="Epoch"):
        # going through every batch in training set
        for batch in tqdm_notebook(range(batch_per_epoch),desc="Batch"):

            # Getting randomly selected real images form data
            X_real_image, y_real_labels = generate_real_images(data,half_batch)
            # Training Discriminator (Updating weights)
            discriminator_loss1,_ = discriminator_model.train_on_batch(X_real_image, y_real_labels)
            # Generating Fake Images
            X_fake_image, y_fake_labels = generate_fake_images(generator_model,latent_dimension,half_batch)
            # Training Discriminator (Updating weights)
            discriminator_loss2,_ = discriminator_model.train_on_batch(X_fake_image, y_fake_labels)

            # Preparing input points in latent space as input for generator
            X_gan_points = generate_latent_points(latent_dimension,n_batch)
            # Creating inverted labels for fake samples
            y_gan_labels = np.ones((n_batch))\
            # Now we will update generator via loss of discriminator
            gan_loss = gan_model.train_on_batch(X_gan_points,y_gan_labels)
            # Summarizing Losses
            print("> epoch==>%d, current/batch_size=> %d/%d, d1_loss=>%.3f,d2_loss=>%.3f,gan_loss=>%.3f"%
                  (epoch+1,batch+1,batch_per_epoch,discriminator_loss1,discriminator_loss2,gan_loss))
        # Evalauate model performance after every 10 epoch
        if (epoch+1) % 10 == 0:
            evaluate_model(epoch,generator_model,discriminator_model,data,latent_dimension)

```

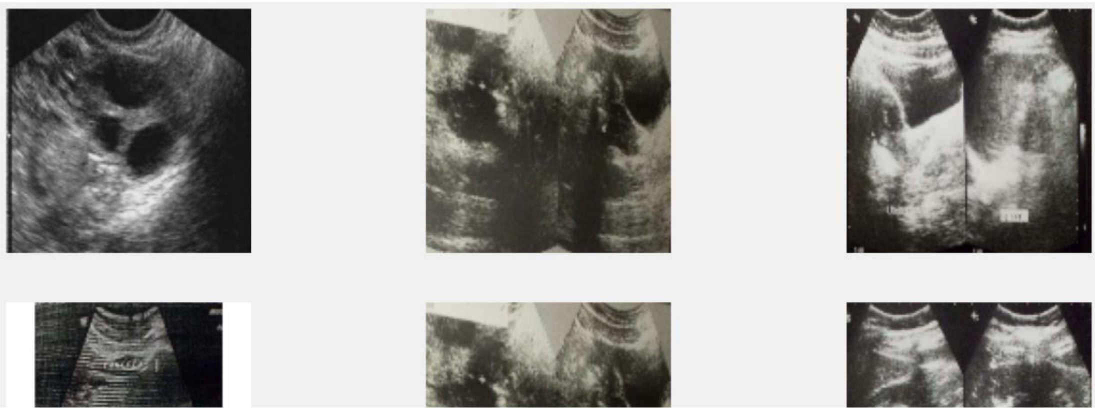
Visualising Data

In [14]:

```python
path = r"dataset\NON PCOD"    # Change path accroding to directory
image_shape = (128,128,3)
latent_dimension = 100
img_list = load_real_images(path)
```

In [16]:

```python
# Plotting Real images
saving_images(img_list[0:9],saving=False)
```



Loading models

18]:
```
# Define the discriminator
discriminator_model = discriminator()
# Define the Generator
generator_model = generator(latent_dimension)
# Defininf GAN model
gan_model = generate_gan(generator_model,discriminator_model)
```

Visualising Models

16]:
```
# plotting Discriminator model
plot_model(discriminator_model, to_file='discriminator_plot.png', show_shapes=True, show_layer_names=True)
```

16]:

conv2d_4_input: InputLayer	input:	[(None, 128, 128, 3)]
	output:	[(None, 128, 128, 3)]

Here we are loading both model and defining GAN model using function generate_gan(), 200 epoch executed as seen loss for discriminator constantly coming 0 we have stopped the model.

Training Model

In [19]:
```
# load data
data = load_real_images(path)
```

In [20]:
```
# train model
train(generator_model, discriminator_model, gan_model, data, latent_dimension)
```

```
> epoch==>200, current/batch_size=> 19/37, d1_loss=>0.000,d2_loss=>0.000,gan_loss=>7.532
> epoch==>200, current/batch_size=> 20/37, d1_loss=>0.000,d2_loss=>0.009,gan_loss=>5.585
> epoch==>200, current/batch_size=> 21/37, d1_loss=>0.000,d2_loss=>0.239,gan_loss=>15.111
> epoch==>200, current/batch_size=> 22/37, d1_loss=>0.000,d2_loss=>0.000,gan_loss=>26.931
> epoch==>200, current/batch_size=> 23/37, d1_loss=>0.090,d2_loss=>0.000,gan_loss=>22.629
> epoch==>200, current/batch_size=> 24/37, d1_loss=>0.004,d2_loss=>0.000,gan_loss=>21.990
> epoch==>200, current/batch_size=> 25/37, d1_loss=>0.002,d2_loss=>0.000,gan_loss=>21.789
> epoch==>200, current/batch_size=> 26/37, d1_loss=>0.038,d2_loss=>0.000,gan_loss=>19.051
> epoch==>200, current/batch_size=> 27/37, d1_loss=>0.003,d2_loss=>0.000,gan_loss=>19.227
> epoch==>200, current/batch_size=> 28/37, d1_loss=>0.001,d2_loss=>0.000,gan_loss=>14.546
> epoch==>200, current/batch_size=> 29/37, d1_loss=>0.001,d2_loss=>0.000,gan_loss=>13.313
> epoch==>200, current/batch_size=> 30/37, d1_loss=>0.002,d2_loss=>0.000,gan_loss=>13.403
> epoch==>200, current/batch_size=> 31/37, d1_loss=>0.001,d2_loss=>0.000,gan_loss=>14.287
> epoch==>200, current/batch_size=> 32/37, d1_loss=>0.004,d2_loss=>0.000,gan_loss=>13.737
> epoch==>200, current/batch_size=> 33/37, d1_loss=>0.002,d2_loss=>0.000,gan_loss=>7.946
> epoch==>200, current/batch_size=> 34/37, d1_loss=>0.000,d2_loss=>0.000,gan_loss=>8.960
> epoch==>200, current/batch_size=> 35/37, d1_loss=>0.001,d2_loss=>0.004,gan_loss=>6.760
> epoch==>200, current/batch_size=> 36/37, d1_loss=>0.000,d2_loss=>0.036,gan_loss=>3.944
> epoch==>200, current/batch_size=> 37/37, d1_loss=>0.002,d2_loss=>0.004,gan_loss=>6.807
>Accuracy real images: 100%, fake images: 100%
```

Section 2 Pre-Trained Models Implementation:

Importing libraries and ran each model for epoch 50 initially then also with 25 to compare the results.

Data augmentation is done by ImageGenerator() which split the data and push it into model.

jupyter finalmodel (autosaved)

File Edit View Insert Cell Kernel Widgets Help   Not Trusted

In [2]:
```python
import warnings
warnings.filterwarnings(action='ignore')
```

In [ ]:
```python

```

In [3]:
```python
import pickle
import numpy as np
from keras import backend as K

from keras.applications import ResNet50,VGG19
from keras.applications import InceptionV3,DenseNet121
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam,RMSprop
from keras.preprocessing import image
from keras.preprocessing.image import img_to_array

from keras.layers import Flatten
from sklearn.linear_model import LogisticRegression
from keras.layers import Dense, Dropout, InputLayer
from keras.layers import Input,BatchNormalization
from keras.models import Sequential,Model

import matplotlib.pyplot as plt
import keras_metrics
```

In [4]:
```python
plt.style.use('fivethirtyeight')
```

```python
data_gen = ImageDataGenerator(rotation_range = 40,
        shear_range = 0.2,
        zoom_range = 0.2,
        horizontal_flip = True,
        validation_split=0.2)
```

```python

train_data = data_gen.flow_from_directory(directory = r"dataset",
                                          target_size=inputShape[0:2],
                                          batch_size=4,
                                          class_mode='categorical',
                                          shuffle=True,
                                          subset='training')
val_data = data_gen.flow_from_directory(directory = r"dataset",
                                          target_size=inputShape[0:2],
                                          batch_size=1,
                                          class_mode='categorical',
                                          shuffle=True,
                                          subset='validation')
```

```
Found 236 images belonging to 2 classes.
Found 58 images belonging to 2 classes.
```

Resnet50

[13]:
```
model_res = ResNet50(include_top=False, weights="imagenet",input_shape =inputShape)
```

```
A local file was found, but it seems to be incomplete or outdated because the auto file hash does not match the original va
of 4d473c1dd8becc155b73f8504c6f6626 so we will re-download the data.
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/resnet/resnet50_weights_tf_dim_ordering_
ernels_notop.h5
94773248/94765736 [==============================] - 45s 0us/step
```

[14]:
```
model_resnet = Sequential()
model_resnet.add(model_res)
model_resnet.add(Flatten())
model_resnet.add(BatchNormalization())
model_resnet.add(Dense(256, activation='relu'))
model_resnet.add(Dropout(0.5))
model_resnet.add(BatchNormalization())
model_resnet.add(Dense(128, activation='relu'))
model_resnet.add(Dropout(0.5))
model_resnet.add(BatchNormalization())
model_resnet.add(Dense(2, activation='softmax')) # [0.9,0.1]

model_resnet.layers[0].trainable = False
```

[15]:
```
model_resnet.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
```

Taking this model as protype rest all pretrained model will have same logic.
Resnet here is taking the weights of imagenet database and setting trainable status to false and including dense layer with batch size f 4 and activation used here is softmax.

:
```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model_resnet.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy",keras_metrics.precision(), keras_met
# train the network
print("[INFO] training network...")
history_res = model_resnet.fit(train_data,epochs=EPOCHS,validation_data = val_data)
```

```
4 - f1_score: 0.8279 - val_loss: 2.1324 - val_accuracy: 0.5690 - val_precision: 0.3966 - val_recall: 0.0246 - val_f1_score:
0.0461
Epoch 20/25
59/59 [==============================] - 10s 169ms/step - loss: 0.2768 - accuracy: 0.9203 - precision: 0.9443 - recall: 0.947
4 - f1 score: 0.9456 - val loss: 1.9629 - val accuracy: 0.5345 - val precision: 0.4368 - val recall: 0.0816 - val f1 score:
```

VGG 19

21]:
```
model_19 = VGG19(include_top=False, weights="imagenet",input_shape =inputShape)
```

22]:
```
model_vgg = Sequential()
model_vgg.add(model_19)
model_vgg.add(Flatten())
model_vgg.add(BatchNormalization())
model_vgg.add(Dense(256, activation='relu'))
model_vgg.add(Dropout(0.5))
model_vgg.add(BatchNormalization())
model_vgg.add(Dense(128, activation='relu'))
model_vgg.add(Dropout(0.5))
model_vgg.add(BatchNormalization())
model_vgg.add(Dense(2, activation='softmax'))

model_vgg.layers[0].trainable = False
```

23]:
```
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model_vgg.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy",keras_metrics.precision(), keras_metri
# train the network
print("[INFO] training network...")
history_vgg= model_vgg.fit(train_data,epochs=EPOCHS,validation_data = val_data)
```

Inceptionv3

25]:
```python
model_V3 = InceptionV3(include_top=False, weights="imagenet",input_shape =inputShape)
```

26]:
```python
model_ins = Sequential()
model_ins.add(model_V3)
model_ins.add(Flatten())
model_ins.add(BatchNormalization())
model_ins.add(Dense(256, activation='relu'))
model_ins.add(Dropout(0.5))
model_ins.add(BatchNormalization())
model_ins.add(Dense(128, activation='relu'))
model_ins.add(Dropout(0.5))
model_ins.add(BatchNormalization())
model_ins.add(Dense(2, activation='softmax'))

model_ins.layers[0].trainable = False
```

27]:
```python
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model_ins.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy",keras_metrics.precision(), keras_metric
# train the network
print("[INFO] training network...")
history_ins= model_ins.fit(train_data,epochs=EPOCHS,validation_data = val_data)
```

DenseNet

In [29]:
```python
model_net = DenseNet121(include_top=False, weights="imagenet",input_shape =inputShape)
```

In [30]:
```python
model_dense = Sequential()
model_dense.add(model_net)
model_dense.add(Flatten())
model_dense.add(BatchNormalization())
model_dense.add(Dense(256, activation='relu'))
model_dense.add(Dropout(0.5))
model_dense.add(BatchNormalization())
model_dense.add(Dense(128, activation='relu'))
model_dense.add(Dropout(0.5))
model_dense.add(BatchNormalization())
model_dense.add(Dense(2, activation='softmax'))

model_dense.layers[0].trainable = False
```

In [31]:
```python
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
# distribution
model_dense.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy",keras_metrics.precision(), keras_metr
# train the network
print("[INFO] training network...")
history_dense= model_dense.fit(train_data,epochs=EPOCHS,validation_data = val_data)
```

```
[INFO] training network...
Epoch 1/25
```

Stacking Classsifiers

In [33]:
```python
from sklearn.metrics import accuracy_score,recall_score,precision_score
from sklearn.metrics import f1_score,confusion_matrix
```

In [34]:
```python
def stacking_predictions(models,data):
    # array to store values
    stackValues = None
    for model in models:
        # making predictions for each model
        y_pred = model.predict(data)
        # stack predictions into [rows, members, probabilities]
        if stackValues is None:
            stackValues = y_pred
        else:
            stackValues = np.dstack((stackValues,y_pred))
    # flatten predictions to [rows, members x probabilities]
    stackValues = stackValues.reshape((stackValues.shape[0], stackValues.shape[1]*stackValues.shape[2]))
    return stackValues
```

In [35]:
```python
def fit_models(models,data):
    # stacked data with ensemble
    stackedValues = stacking_predictions(models,data)
    log_reg = LogisticRegression()
    log_reg.fit(stackedValues,data.labels)
    return log_reg
```

In [36]:
```python
def stacked_prediction(members, model, inputX):
    # create dataset using ensemble
    stackedX = stacking_predictions(members, inputX)
    # make a prediction
```

[37]:
```
# stacking every model
stack_models = [model_resnet,model_vgg,model_ins,model_dense]
```

[38]:
```
model = fit_models(stack_models,train_data)
```

[39]:
```
stack_prediction_labels = stacked_prediction(stack_models, model, val_data)
acc = accuracy_score(val_data.labels, stack_prediction_labels)
recall = recall_score(val_data.labels, stack_prediction_labels,average="weighted")
precision = precision_score(val_data.labels, stack_prediction_labels,average="weighted")
fscore = f1_score(val_data.labels, stack_prediction_labels,average="weighted")
conf_matrix = confusion_matrix(val_data.labels, stack_prediction_labels)
print('Stacked Test Accuracy: %.3f' % acc)
print('Stacked Test Recall: %.3f' % recall)
print('Stacked Test Precision: %.3f' % precision)
print('Stacked Test F1Score : %.3f' % fscore)
```

Above we have implemented model stacking where output of each model will be taken as in to logistic regression and prediction will be made.
Coparing all plots in terms of accuracy and presenting the results, like wise same implemnted for Loss, Precicio

Comparison of all models

```

plt.figure(figsize=(15,5))
plt.title("Test Accuracy Comparison")
plt.plot(history_res.history["val_accuracy"],label="resnet")
plt.plot(history_vgg.history["val_accuracy"],label="VGG19")
plt.plot(history_ins.history["val_accuracy"],label="Inception")
plt.plot(history_dense.history["val_accuracy"],label="DenseNet")
plt.legend()
plt.show()
```

References:

https://keras.io/api/applications/inceptionv3/
https://keras.io/guides/transfer_learning/
https://machinelearningmastery.com/what-are-generative-adversarial-networks-gans/
https://towardsdatascience.com/gan-ways-to-improve-gan-performance-acf37f9f59b
https://towardsdatascience.com/a-practical-guide-to-stacking-using-scikit-learn-91e8d021863d
https://pubs.rsna.org/doi/10.1148/rg.326125503
https://radiopaedia.org/articles/polycystic-ovarian-syndrome-1
https://medpix.nlm.nih.gov/search?allen=true&allt=true&alli=true&query=pcos
https://www.intechopen.com/chapters/45102
https://www.degruyter.com/document/doi/10.1515/jpem-2014-0307/html
https://www.nebojsazecevic.com/en/themes/polycystic-ovary-syndrome-pcos/5246/clinical-picture-of-polycystic-ovary-syndrome
https://radiologykey.com/the-ovary-and-polycystic-ovary-syndrome/