**Keras fit, fit\_generator, train\_on\_batch**

The Keras deep learning library provides three different methods to train deep learning models.

***.fit***

***.fit\_generator***

***.train\_on\_batch***

All these three functions used to accomplish the same work such as train the model, but they work in a very different way.

In this tutorial, we will learn about all these model training functions with example. when to use a particular function when training your deep learning models.

Let’s explore all these functions one-by-one.

**Keras.fit()**

The main **two** primary premises of ***Keras.fit*** is:

While training the model, our entire training data will fit into RAM

not allows performing real-time data augmentation on images

When we have a huge amount of training data, it required more resources to fit the entire training data into RAM. So, it isn’t a feasible solution to use **.fit()** function with large data.

The [Keras.fit()](https://keras.io/models/sequential/" \l "fit" \t "_blank) has various parameters:

fit(x=**None**, y=**None**, batch\_size=**None**, epochs=1, verbose=1, callbacks=**None**, validation\_split=0.0, validation\_data=**None**, shuffle=**True**, class\_weight=**None**, sample\_weight=**None**, initial\_epoch=0, steps\_per\_epoch=**None**, validation\_steps=**None**, validation\_freq=1, max\_queue\_size=10, workers=1, use\_multiprocessing=**False**)

**Keras.fit\_generator()**

It is perfectly acceptable to use Keras.fit() function when you are train model on a small and simplest dataset.  These datasets also do not require any data augmentation.

But, Real-world datasets are often too large to fit into memory. It also required to perform data augmentation to avoid overfitting. In these situations, we should use Keras’**fit\_generator()**function to train the model.

The ***[Keras.fit\_generator()](https://keras.io/models/sequential/" \l "fit_generator" \t "_blank)*** train the model on data generated batch-by-batch by a Python generator. It also allows you to do real-time data augmentation on images on CPU in parallel to training your model on GPU.

fit\_generator doesn’t accept the X and Y directly, need to pass through the generator.

fit\_generator(generator, steps\_per\_epoch=**None**, epochs=1, verbose=1, callbacks=**None**, validation\_data=**None**, validation\_steps=**None**, validation\_freq=1, class\_weight=**None**, max\_queue\_size=10, workers=1, use\_multiprocessing=**False**, shuffle=**True**, initial\_epoch=0)

generator: a tuple  (inputs, targets)

steps\_per\_epoch: Total number of steps (batches of samples)

**fit()**function has no such parameter.

**fit\_generator** use the generator as input. Keep in mind that the Generator is an iterator which iterates the loop infinitely and never exit. Since the function is intended to loop infinitely, Keras has no ability to determine when *one epoch starts* and a *new epoch begins*. Therefore, we compute the steps\_per\_epoch which is equal to **ceil(num\_samples / batch\_size).**

**Keras.train\_on\_batch**

Keras’ **[train\_on\_batch](https://keras.io/models/sequential/" \l "train_on_batch" \t "_blank)**  function accepts a single batch of data, perform backpropagation on it and then update the model parameters. The batch of data can be any size- doesn’t require to define explicitly.

train\_on\_batch(x, y, sample\_weight=**None**, class\_weight=**None**, reset\_metrics=**True**)

When you want to control over the training your deep learning models, you may wish to use **Keras.train\_on\_batch()**function. It requires writing custom code for training deep learning model which is quite complex.

If you are not an advanced deep learning practitioner, it is better to use **Keras.fit\_generator()** methods for model training.

KERAS DATA PIPELINES

APPROACH 1:🡪

1. We created **all\_images, train\_images, val\_images, ppl**

# all\_images will have path of all jpg with

#print(all\_images[0]) --> ./train/F0049/MID3/P00494\_face3.jpg

# In train\_images collecting path of all training data & # In val\_images collecting path of all validation data

# Path consist of Family name and person name as well

# ppl contains Family name and person name

# print(ppl[0]) --> F0049/MID3

1. Create Key value pairs for Family and its persons 🡪 train\_person\_to\_images\_map, val\_person\_to\_images\_map

# --> key as Family and person id

# --> value as image complete path of that person

print(train\_person\_to\_images\_map['F0002/MID1'])

'''OUTPUT: --> ['./train/F0002/MID1/P00018\_face1.jpg', './train/F0002/MID1/P00011\_face1.jpg',

'./train/F0002/MID1/P00015\_face2.jpg', './train/F0002/MID1/P00016\_face2.jpg',

'./train/F0002/MID1/P00017\_face3.jpg', './train/F0002/MID1/P00014\_face2.jpg',

'./train/F0002/MID1/P00010\_face4.jpg', './train/F0002/MID1/P00009\_face3.jpg',

'./train/F0002/MID1/P00012\_face2.jpg', './train/F0002/MID1/P00013\_face2.jpg']'''

print(val\_person\_to\_images\_map['F0900/MID1'])

'''OUTPUT: --> ['./train/F0900/MID1/P09508\_face1.jpg', './train/F0900/MID1/P09513\_face1.jpg',

'./train/F0900/MID1/P09506\_face1.jpg', './train/F0900/MID1/P09509\_face1.jpg',

'./train/F0900/MID1/P09505\_face1.jpg']'''

1. Now load relationships CSV file to 🡪 relationships dataframe

EXAMPLE

p1 p2

0 F0002/MID1 F0002/MID3

1 F0002/MID2 F0002/MID3

2 F0005/MID1 F0005/MID2

3 F0005/MID3 F0005/MID2

4 F0009/MID1 F0009/MID4

1. Create train & val from relationships dataframe
2. Create function generator for

# Function gen is to generate data for label 0

# example --> gen(train, train\_person\_to\_images\_map, batch\_size=16)

# train\_person\_to\_images\_map is dictionary with family and person as key and values are image path

#

def gen(list\_tuples, person\_to\_images\_map, batch\_size=16):

# All keys of dictionary stored in

ppl = list(person\_to\_images\_map.keys())

while True:

# list\_tuplples is train data from relationship csv file

# batch\_tuples contains sample from train data (from relationship.csv)

batch\_tuples = sample(list\_tuples, batch\_size // 2)

# If batch\_size is of size 16 then batch\_tuples will have 8 samples

labels = [1] \* len(batch\_tuples)

while len(batch\_tuples) < batch\_size:

# choice randomly picking data from ppl

p1 = choice(ppl)

p2 = choice(ppl)

# Randomply Creating data with labels 0

if p1 != p2 and (p1, p2) not in list\_tuples and (p2, p1) not in list\_tuples:

batch\_tuples.append((p1, p2))

labels.append(0)

# for each family person combination from batch\_tuples

# x[0] means considering first entry for ex --> ('F0488/MID1', 'F0488/MID4') --> consider 'F0488/MID1'

# Check in dictioanry train\_person\_to\_images\_map

for x in batch\_tuples:

if not len(person\_to\_images\_map[x[0]]):

print(x[0])

# Considering randomly images using choice function

X1 = [choice(person\_to\_images\_map[x[0]]) for x in batch\_tuples]

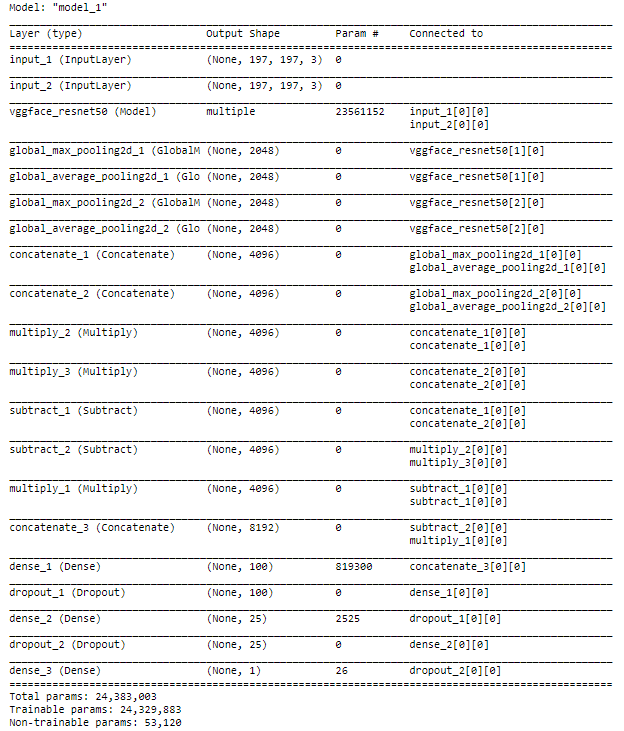
X1 = np.array([read\_img(x, 197) for x in X1])

X2 = [choice(person\_to\_images\_map[x[1]]) for x in batch\_tuples]

X2 = np.array([read\_img(x, 197) for x in X2])

yield [X1, X2], labels

1. Create baseline\_model1



1. Create callbacks

file\_path1 = "./model1.h5"

checkpoint1 = ModelCheckpoint(file\_path1, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max')

reduce\_on\_plateau1 = ReduceLROnPlateau(monitor="val\_acc", mode="max", factor=0.1, patience=20, verbose=1)

callbacks\_list1 = [checkpoint1, reduce\_on\_plateau1]

1. Use gen function in fit\_generator

model1.fit\_generator(gen(train, train\_person\_to\_images\_map, batch\_size=16), use\_multiprocessing=True,

validation\_data=gen(val, val\_person\_to\_images\_map, batch\_size=16), epochs=100, verbose=1,

workers = 4, callbacks=callbacks\_list1, steps\_per\_epoch=200, validation\_steps=100)

APPROACH 2:🡪

1. We created **all\_images, train\_images, val\_images, ppl**

# all\_images will have path of all jpg with

#print(all\_images[0]) --> ./train/F0049/MID3/P00494\_face3.jpg

# In train\_images collecting path of all training data & # In val\_images collecting path of all validation data

# Path consist of Family name and person name as well

# ppl contains Family name and person name

# print(ppl[0]) --> F0049/MID3

1. Create Key value pairs for Family and its persons 🡪 train\_person\_to\_images\_map, val\_person\_to\_images\_map

# --> key as Family and person id

# --> value as image complete path of that person

print(train\_person\_to\_images\_map['F0002/MID1'])

'''OUTPUT: --> ['./train/F0002/MID1/P00018\_face1.jpg', './train/F0002/MID1/P00011\_face1.jpg',

'./train/F0002/MID1/P00015\_face2.jpg', './train/F0002/MID1/P00016\_face2.jpg',

'./train/F0002/MID1/P00017\_face3.jpg', './train/F0002/MID1/P00014\_face2.jpg',

'./train/F0002/MID1/P00010\_face4.jpg', './train/F0002/MID1/P00009\_face3.jpg',

'./train/F0002/MID1/P00012\_face2.jpg', './train/F0002/MID1/P00013\_face2.jpg']'''

print(val\_person\_to\_images\_map['F0900/MID1'])

'''OUTPUT: --> ['./train/F0900/MID1/P09508\_face1.jpg', './train/F0900/MID1/P09513\_face1.jpg',

'./train/F0900/MID1/P09506\_face1.jpg', './train/F0900/MID1/P09509\_face1.jpg',

'./train/F0900/MID1/P09505\_face1.jpg']'''

1. Now load relationships CSV file to 🡪 relationships dataframe

EXAMPLE

p1 p2

0 F0002/MID1 F0002/MID3

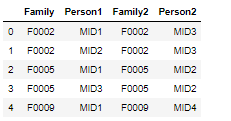
1 F0002/MID2 F0002/MID3

2 F0005/MID1 F0005/MID2

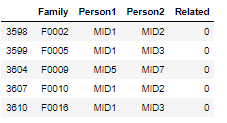
3 F0005/MID3 F0005/MID2

4 F0009/MID1 F0009/MID4

Change dataframe in below format



Then to following format



1. Create train & val from relationships dataframe
2. Now convert images to array using below function

# Now we will convert images to array

#val\_fammm = "F0009"

val\_fammm = "F09"

def img\_to\_array(dataframe, IMAGE\_SIZE):

# INITIALIZING LIST

X1\_train\_list=[]

X2\_train\_list=[]

y\_train=[]

X1\_val\_list=[]

X2\_val\_list=[]

y\_val=[]

for index, row in dataframe.iterrows():

#print(type(row[:]))

#print(row['Family']+'/'+row['Person1'])

#print(row['Family']+'/'+row['Person2'])

#print(row['Related'])

if val\_fammm not in row['Family'] and ((len(train\_person\_to\_images\_map[row['Family']+'/'+row['Person1']]) != 0) and (len(train\_person\_to\_images\_map[row['Family']+'/'+row['Person2']]) != 0) ):

#X1 = [choice(train\_person\_to\_images\_map[row['Family']+'/'+row['Person1']])]

X1\_train = [choice(train\_person\_to\_images\_map[row['Family']+'/'+row['Person1']])]

X1\_train = np.array([read\_img(x, IMAGE\_SIZE) for x in X1\_train])

X1\_train\_list.append(X1\_train[0])

X2\_train = [choice(train\_person\_to\_images\_map[row['Family']+'/'+row['Person2']])]

X2\_train = np.array([read\_img(x, IMAGE\_SIZE) for x in X2\_train])

X2\_train\_list.append(X2\_train[0])

y\_train.append(row['Related'])

elif ((len(val\_person\_to\_images\_map[row['Family']+'/'+row['Person1']]) != 0) and (len(val\_person\_to\_images\_map[row['Family']+'/'+row['Person2']]) != 0)):

X1\_val = [choice(val\_person\_to\_images\_map[row['Family']+'/'+row['Person1']])]

X1\_val = np.array([read\_img(x, IMAGE\_SIZE) for x in X1\_val])

X1\_val\_list.append(X1\_val[0])

X2\_val = [choice(val\_person\_to\_images\_map[row['Family']+'/'+row['Person2']])]

X2\_val = np.array([read\_img(x, IMAGE\_SIZE) for x in X2\_val])

X2\_val\_list.append(X2\_val[0])

y\_val.append(row['Related'])

#X1\_val = [choice(train\_person\_to\_images\_map[row['Family']+'/'+row['Person1']])]

#X1\_val = np.array([read\_img(x, 197) for x in X1\_val])

#X1\_val\_list.append(X1\_train[0])

#X2\_val = [choice(train\_person\_to\_images\_map[row['Family']+'/'+row['Person2']])]

#X2\_val = np.array([read\_img(x, 197) for x in X2\_val])

#X2\_val\_list.append(X2\_val[0])

#y\_val.append(row['Related'])

else:

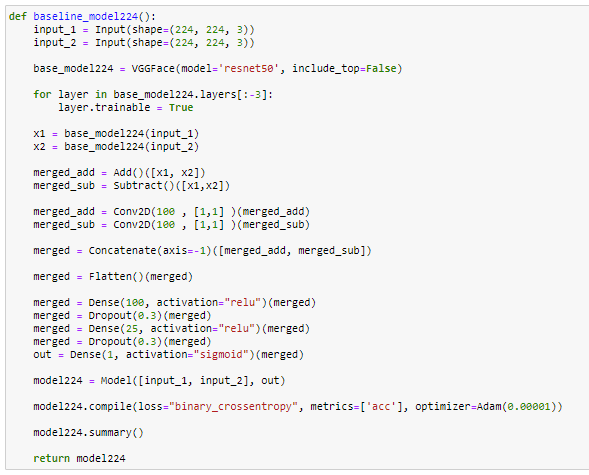
#print(row['Family']+'/'+row['Person1'])

#print(row['Family']+'/'+row['Person1'])

skip=1

return X1\_train\_list, X2\_train\_list, y\_train, X1\_val\_list, X2\_val\_list, y\_val

1. X1\_trn\_list, X2\_trn\_list, y\_trn, X1\_v\_list, X2\_v\_list, y\_v = img\_to\_array(df, 224)
2. Create baseline\_model1



1. Create callbacks

file\_path224 = "./model224.h5"

tb\_dir224 = "./TENSORBOARD/MODEL224/"

checkpoint224 = ModelCheckpoint(file\_path224, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max')

reduce\_on\_plateau224 = ReduceLROnPlateau(monitor="val\_acc", mode="max", factor=0.1, patience=20, verbose=1)

Tensorboard224 = TensorBoard(log\_dir=tb\_dir224, write\_graph=True, update\_freq=100, histogram\_freq=1, write\_images=True)

callbacks\_list224 = [checkpoint224, reduce\_on\_plateau224, Tensorboard224]

model224 = baseline\_model224()

1. Now using fit

model224.fit([X1\_train\_array, X2\_train\_array], y\_train\_array, epochs=100,batch\_size=None,verbose=1,

validation\_data=([X1\_val\_array, X2\_val\_array], y\_val\_array), callbacks=callbacks\_list224,

use\_multiprocessing=True, workers=4,steps\_per\_epoch=200, validation\_steps=100

)

APPROACH 3:🡪

Same as approach 2

Only difference is we used Datagenerator

class DataGenerator(keras.utils.Sequence):

'Generates data for Keras'

def \_\_init\_\_(self, list1, list2, labels, batch\_size=16, image\_dimensions = (224, 224, 3), shuffle=False, augment=False):

self.labels = labels # array of labels

self.list1 = list1 # array of image paths

self.list2 = list2 # array of image paths

self.dim = image\_dimensions # image dimensions

self.batch\_size = batch\_size # batch size

self.shuffle = shuffle # shuffle bool

self.augment = augment # augment data bool

self.n = 0

self.on\_epoch\_end()

def \_\_next\_\_(self):

# Get one batch of data

data = self.\_\_getitem\_\_(self.n)

# Batch index

self.n += 1

# If we have processed the entire dataset then

if self.n >= self.\_\_len\_\_():

self.on\_epoch\_end

self.n = 0

return data

def \_\_len\_\_(self):

'Denotes the number of batches per epoch'

return int(np.floor(len(self.list1) / self.batch\_size))

def on\_epoch\_end(self):

'Updates indexes after each epoch'

self.indexes = np.arange(len(self.list1))

if self.shuffle:

np.random.shuffle(self.indexes)

def \_\_getitem\_\_(self, index):

'Generate one batch of data'

# selects indices of data for next batch

indexes = self.indexes[index \* self.batch\_size : (index + 1) \* self.batch\_size]

# select data and load images

labels = np.array([self.labels[k] for k in indexes])

X1 = np.array([self.list1[k] for k in indexes])

X2 = np.array([self.list2[k] for k in indexes])

# preprocess and augment data

if self.augment == True:

images = self.augmentor(images)

return [X1,X2], labels

Create Generator

train\_datagen = DataGenerator(list1=X1\_trn\_list, list2=X2\_trn\_list, labels=y\_trn, batch\_size=1, image\_dimensions=(224,224,3),shuffle=False, augment=False)

val\_datagen = DataGenerator(list1=X1\_v\_list, list2=X2\_v\_list, labels=y\_v, batch\_size=1, image\_dimensions=(224,224,3),shuffle=False, augment=False)

We used fit in our case

model224.fit\_generator(train\_datagen,

validation\_data=val\_datagen, epochs=10, verbose=2,

callbacks=callbacks\_list224, validation\_steps=1)

=================================================================================================================

**APPROACH 4:🡪**

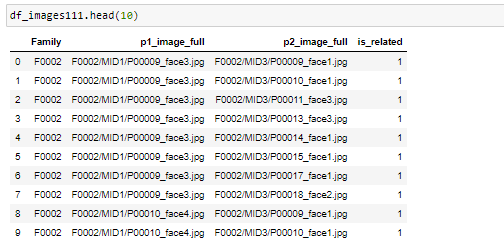
=================================================================================================================

In order to increase the accuracy next we focus on following point

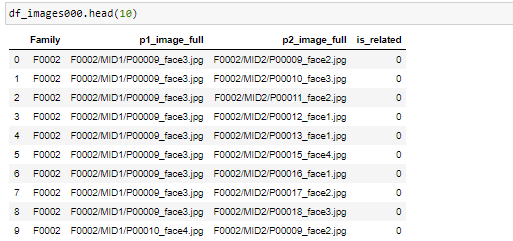
1. Since 0 output is less compare to output 1 so we consider more 0 output values
2. Create Data augmentation in case our test data consist of it.

This time we created dataframe in following format

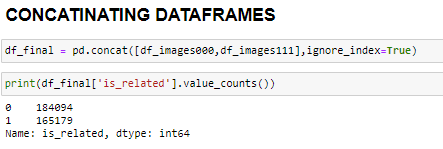
First create dataframe for output is\_related = 1



Next we create dataframe for output is\_related = 0



Then we concatenated the dataframe



CREATE TRAIN TEST CV

X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(data, related, test\_size=0.2)

image\_path\_train1 = X\_train['p1\_image\_full'].apply(lambda x: './train/' + x ).values[:-1]

image\_path\_train2 = X\_train['p2\_image\_full'].apply(lambda x: './train/' + x ).values[:-1]

labels = Y\_train.values[:-1]

image\_path\_test1 = X\_test['p1\_image\_full'].apply(lambda x: './train/' + x ).values[:-1]

image\_path\_test2 = X\_test['p2\_image\_full'].apply(lambda x: './train/' + x ).values[:-1]

labels\_test = Y\_test.values[:-1]

**CHANGE the DataGenerator with augmentation part**

class DataGenerator(keras.utils.Sequence):

'Generates data for Keras'

def \_\_init\_\_(self, images\_path1, images\_path2, labels, batch\_size=4, image\_dimensions = (224 ,224 ,3), shuffle=False, augment=True):

self.labels = labels # array of labels

self.images\_path1 = images\_path1 # array of image paths

self.images\_path2 = images\_path2 # array of image paths

self.dim = image\_dimensions # image dimensions

self.batch\_size = batch\_size # batch size

self.shuffle = shuffle # shuffle bool

self.augment = augment # augment data bool

self.n = 0

self.on\_epoch\_end()

def \_\_next\_\_(self):

# Get one batch of data

data = self.\_\_getitem\_\_(self.n)

# Batch index

self.n += 1

# If we have processed the entire dataset then

if self.n >= self.\_\_len\_\_():

self.on\_epoch\_end

self.n = 0

return data

def \_\_len\_\_(self):

'Denotes the number of batches per epoch'

return int(np.floor(len(self.images\_path1)/ self.batch\_size))

def on\_epoch\_end(self):

'Updates indexes after each epoch'

self.indexes = np.arange(len(self.images\_path1))

if self.shuffle:

np.random.shuffle(self.indexes)

def \_\_getitem\_\_(self, index):

'Generate one batch of data'

# selects indices of data for next batch

indexes = self.indexes[index \* self.batch\_size : (index + 1) \* self.batch\_size]

# select data and load images

labels = np.array([self.labels[k] for k in indexes])

#IMG\_SIZE=224

#image1 = image.load\_img((((self.images\_path1[k]) for k in indexes)), target\_size=(IMG\_SIZE, IMG\_SIZE))

#image1 = np.array(image).astype(np.float)

#image2 = image.load\_img((((self.images\_path2[k]) for k in indexes)), target\_size=(IMG\_SIZE, IMG\_SIZE))

image1 = [cv2.imread(self.images\_path1[k]) for k in indexes]

image2 = [cv2.imread(self.images\_path2[k]) for k in indexes]

# preprocess and augment data

if self.augment == True:

image1 = self.augmentor(image1)

image2 = self.augmentor(image2)

#image1 = image1.astype('float32')

#image2 = image2.astype('float32')

image1 = np.array([preprocess\_input(img.astype('float32')) for img in image1])

image2 = np.array([preprocess\_input(img.astype('float32')) for img in image2])

return [image1,image2], labels

def augmentor(self, images):

'Apply data augmentation'

sometimes = lambda aug: iaa.Sometimes(0.5, aug)

seq = iaa.Sequential(

[

# apply the following augmenters to most images

iaa.Fliplr(0.5), # horizontally flip 50% of all images

iaa.Flipud(0.2), # vertically flip 20% of all images

sometimes(iaa.Affine(

scale={"x": (0.9, 1.1), "y": (0.9, 1.1)},

# scale images to 80-120% of their size, individually per axis

translate\_percent={"x": (-0.1, 0.1), "y": (-0.1, 0.1)},

# translate by -20 to +20 percent (per axis)

rotate=(-10, 10), # rotate by -45 to +45 degrees

shear=(-5, 5), # shear by -16 to +16 degrees

order=[0, 1],

# use nearest neighbour or bilinear interpolation (fast)

cval=(0, 255), # if mode is constant, use a cval between 0 and 255

mode=ia.ALL

#use any scikit-image's warping modes (see 2nd image from the top for examples)

)),

],

random\_order=True

)

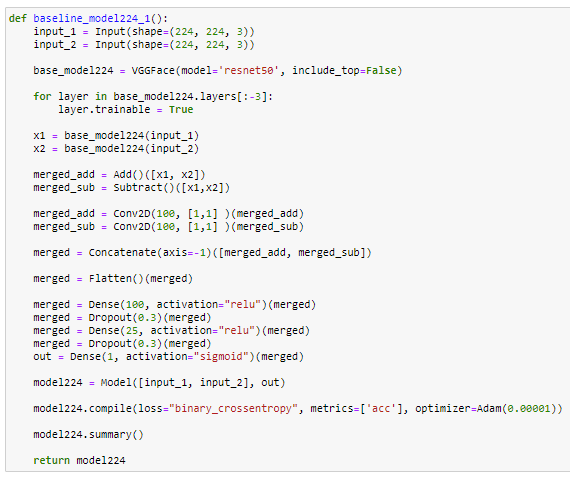
return seq.augment\_images(images)

**CREATE train\_datagen, val\_datagen**

train\_datagen = DataGenerator(image\_path\_train1, image\_path\_train2, labels, batch\_size=4, augment=True, shuffle=True)

val\_datagen = DataGenerator(image\_path\_test1, image\_path\_test2, labels\_test, batch\_size=4, augment=False, shuffle=False)

**CREATING MODEL**



**PREPARING CALLBACKS**

file\_path224 = "./model224\_01.h5"

tb\_dir224 = "./TENSORBOARD/MODEL224\_01/"

checkpoint224 = ModelCheckpoint(file\_path224, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max')

reduce\_on\_plateau224 = ReduceLROnPlateau(monitor="val\_acc", mode="max", factor=0.1, patience=20, verbose=1)

#Tensorboard224 = TensorBoard(log\_dir=tb\_dir224, write\_graph=True, update\_freq=100, histogram\_freq=1, write\_images=True)

Tensorboard224 = TensorBoard(log\_dir=tb\_dir224, write\_graph=True, update\_freq=100)

callbacks\_list224 = [checkpoint224, reduce\_on\_plateau224, Tensorboard224]

#callbacks\_list224 = [checkpoint224, reduce\_on\_plateau224]

model224\_01 = baseline\_model224\_1()

**USING FIT\_GENERATOR**

model224\_01.fit\_generator(train\_datagen,

validation\_data=val\_datagen, epochs=10, verbose=1,use\_multiprocessing=True,

callbacks=callbacks\_list224, validation\_steps=100, workers=32)

=================================================================================================================

**APPROACH 5:🡪**

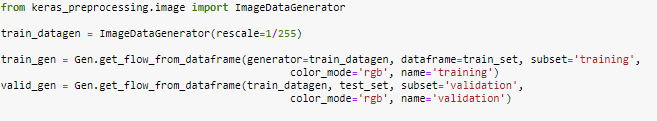
=================================================================================================================

After creating the dataframe same as we do in Approach 4

We change the way generator are used

Using flow\_from\_dataframe





APPROACH 6:🡪

In end we use Blend it all approach to get the good result