1. Code for VGGNet based architecture:

import numpy as np

import pandas as pd

import os

import keras

os.environ["CUDA\_VISIBLE\_DEVICES"]="2"

from keras.models import Sequential

from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization, Activation

from keras.layers.normalization import BatchNormalization

#from keras import backend as K

from keras import optimizers

import tensorflow as tf

from keras.callbacks import ModelCheckpoint

from keras.callbacks import CSVLogger

from keras.optimizers import RMSprop

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau

import matplotlib.pyplot as plt

csv\_logger = CSVLogger('tr\_final.csv', append=True, separator=';')

co0=0

co1=0

co2=0

co3=0

co4=0

co5=0

c0=0

c1=0

c2=0

c3=0

c4=0

c5=0

p = 0

class TestCallback(keras.callbacks.Callback):

def \_\_init\_\_(self, test\_data):

self.test\_data = test\_data

def on\_epoch\_end(self, epoch, logs={}):

global co0,co1,co2,co3,co4,co5,c0,c1,c2,c3,c4,c5,p

x, y = self.test\_data

a = model.predict\_classes(x)

for z in range(len(x)):

q = open('est\_CNN\_final %d.txt'%p,'a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

if y[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif y[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif y[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif y[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

p = p+1

loss, acc = self.model.evaluate(x, y, verbose=1)

print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))

bar = open("test\_epochs\_CNN\_final.csv",'a')

bar.write(str(loss) + ", " + str(acc) + "\n")

foo = open("expectation\_CNN\_final.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + "," + str(co4) + "," + str(co5) + "\n")

boo = open("correct\_CNN\_final.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + str(co4) + "," + str(co5) + "\n")

#K.set\_image\_dim\_ordering('th')

# setting up a random seed for reproducibility

random\_seed = 611

np.random.seed(random\_seed)

dataset = []

label= np.empty((0))

r = []

q = ['t','i','r','l','s']

s = ['t','i','r']

with tf.device('/gpu:2'):

for k in range(9):

path = "/data/venki/myonet/trainingsets/sub%d"%(k+1)

os.chdir(path)

for j in range(5):

for i in range(20):

if k > 6:

if j <=2:

count = 0

with open(s[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

else:

break

else:

count = 0

with open(q[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

labels = np.asarray(pd.get\_dummies(label),dtype = np.int8)

print(len(dataset))

#print(dataset)

d = np.array(dataset)

print(d.shape)

segments = d.reshape(820,90,8)

numOfRows = segments.shape[1]

numOfColumns = segments.shape[2]

numChannels = 1

numFilters = 256 # number of filters in Conv2D layer

# kernal size of the Conv2D layer

kernalSize1 = 2

# max pooling window size

poolingWindowSz = 2

# number of filters in fully connected layers

numNueronsFCL1 = 256

numNueronsFCL2 = 256

# split ratio for test and validation

# number of epochs

Epochs = 200

# batchsize

batchSize = 4

# number of total clases

numClasses = labels.shape[1]

# dropout ratio for dropout layer

dropOutRatio = 0.1

# reshaping the data for network input

reshapedSegments = segments.reshape(segments.shape[0], numOfRows, numOfColumns,1)

# splitting in training and testing data

"""trainSplitRatio = 0.60

trainSplit = np.random.rand(len(reshapedSegments)) < trainSplitRatio

trainX = reshapedSegments[trainSplit]

testX = reshapedSegments[~trainSplit]

trainX = np.nan\_to\_num(trainX)

testX = np.nan\_to\_num(testX)

trainY = labels[trainSplit]

testY = labels[~trainSplit]

zprint(trainX.shape)"""

trainX,testX = reshapedSegments[:640,:],reshapedSegments[640:,:]

trainY,testY = labels[:640,:],labels[640:,:]

print(trainX.shape)

def cnnModel():

model = Sequential()

# adding the first convolutionial layer with 32 filters and 5 by 5 kernal size, using the rectifier as the activation function

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu', input\_shape = (90,8,1)))

model.add(BatchNormalization())

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation = "relu",use\_bias = True))

model.add(Dense(1024, activation = "relu",use\_bias = True))

model.add(Dense(2048, activation = "relu",use\_bias = True))

model.add(Dense(4096, activation = "relu",use\_bias = True))

#model.add(Dropout(0.5))

#model.add(Dropout(0.5))

model.add(Dense(5, activation = "softmax"))

#optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

model.compile(optimizer = 'adam', loss = "categorical\_crossentropy", metrics=["accuracy"])

return model

# Compiling the model to generate a model

# model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

model = cnnModel()

for layer in model.layers:

print(layer.name)

print(len(testX))

print(model.summary())

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',

patience=3,

verbose=1,

factor=0.5,

min\_lr=0.00001)

datagen = ImageDataGenerator(

featurewise\_center=False, # set input mean to 0 over the dataset

samplewise\_center=False, # set each sample mean to 0

featurewise\_std\_normalization=False, # divide inputs by std of the dataset

samplewise\_std\_normalization=False, # divide each input by its std

zca\_whitening=False, # apply ZCA whitening

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.15, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=True, # randomly flip images

vertical\_flip=True) # randomly flip images

datagen.fit(trainX)

history = model.fit\_generator(datagen.flow(trainX,trainY, batch\_size=4),

epochs = 200, validation\_data = (testX,testY),

verbose = 2, steps\_per\_epoch=trainX.shape[0] // 4

, callbacks=[learning\_rate\_reduction])

#model.fit(trainX,trainY,epochs=200,batch\_size=4,verbose=2)

score = model.evaluate(testX,testY,verbose=1)

print(score)

a = model.predict\_classes(testX)

print(a)

print('Baseline Error: %.2f%%' %(100-score[1]\*100))

model.save('model\_EMG\_cnnnet.h5')

np.save('groundTruth\_cnnnet.npy',testY)

np.save('testData\_cnnnet.npy',testX)

for z in range(len(a)):

q = open('est\_cnnnet.txt','a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

elif testY[z][4] == 1:

co4 = co4+1

if testY[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif testY[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif testY[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif testY[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

elif testY[z][4] == 1 and a[z] == 4:

c4 = c4+1

q.write("line %d has"%z + "3\n")

foo = open("expectation\_cnnnet.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + ", " + str(co4) + ", " + str(co5) + "\n")

boo = open("correct\_caps\_cnnnet.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + ", " + str(co4) + ", " + str(co5) + "\n")

2. Code for GoogLe Net architecture:

import numpy as np

import pandas as pd

import os

import keras

os.environ["CUDA\_VISIBLE\_DEVICES"]="2"

from keras.models import Sequential,Model

from keras.layers import Dense, Conv2D, MaxPool2D, Flatten, Dropout, BatchNormalization, Activation, Input

from keras.layers.normalization import BatchNormalization

#from keras import backend as K

from keras import optimizers

import tensorflow as tf

from keras.callbacks import ModelCheckpoint

from keras.callbacks import CSVLogger

from keras.optimizers import RMSprop

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau

import matplotlib.pyplot as plt

csv\_logger = CSVLogger('tr\_final.csv', append=True, separator=';')

co0=0

co1=0

co2=0

co3=0

co4=0

co5=0

c0=0

c1=0

c2=0

c3=0

c4=0

c5=0

p = 0

class TestCallback(keras.callbacks.Callback):

def \_\_init\_\_(self, test\_data):

self.test\_data = test\_data

def on\_epoch\_end(self, epoch, logs={}):

global co0,co1,co2,co3,co4,co5,c0,c1,c2,c3,c4,c5,p

x, y = self.test\_data

a = model.predict\_classes(x)

for z in range(len(x)):

q = open('est\_CNN\_final %d.txt'%p,'a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

if y[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif y[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif y[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif y[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

p = p+1

loss, acc = self.model.evaluate(x, y, verbose=1)

print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))

bar = open("test\_epochs\_CNN\_final.csv",'a')

bar.write(str(loss) + ", " + str(acc) + "\n")

foo = open("expectation\_CNN\_final.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + "," + str(co4) + "," + str(co5) + "\n")

boo = open("correct\_CNN\_final.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + str(co4) + "," + str(co5) + "\n")

#K.set\_image\_dim\_ordering('th')

# setting up a random seed for reproducibility

random\_seed = 611

np.random.seed(random\_seed)

dataset = []

label= np.empty((0))

r = []

q = ['t','i','r','l','s']

s = ['t','i','r']

with tf.device('/gpu:2'):

for k in range(9):

path = "/data/venki/myonet/trainingsets/sub%d"%(k+1)

os.chdir(path)

for j in range(5):

for i in range(20):

if k > 6:

if j <=2:

count = 0

with open(s[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

else:

break

else:

count = 0

with open(q[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

labels = np.asarray(pd.get\_dummies(label),dtype = np.int8)

print(len(dataset))

#print(dataset)

d = np.array(dataset)

print(d.shape)

segments = d.reshape(820,90,8)

numOfRows = segments.shape[1]

numOfColumns = segments.shape[2]

numChannels = 1

numFilters = 256 # number of filters in Conv2D layer

# kernal size of the Conv2D layer

kernalSize1 = 2

# max pooling window size

poolingWindowSz = 2

# number of filters in fully connected layers

numNueronsFCL1 = 256

numNueronsFCL2 = 256

# split ratio for test and validation

# number of epochs

Epochs = 200

# batchsize

batchSize = 4

# number of total clases

numClasses = labels.shape[1]

# dropout ratio for dropout layer

dropOutRatio = 0.1

# reshaping the data for network input

reshapedSegments = segments.reshape(segments.shape[0], numOfRows, numOfColumns,1)

# splitting in training and testing data

"""trainSplitRatio = 0.60

trainSplit = np.random.rand(len(reshapedSegments)) < trainSplitRatio

trainX = reshapedSegments[trainSplit]

testX = reshapedSegments[~trainSplit]

trainX = np.nan\_to\_num(trainX)

testX = np.nan\_to\_num(testX)

trainY = labels[trainSplit]

testY = labels[~trainSplit]

zprint(trainX.shape)"""

trainX,testX = reshapedSegments[:640,:],reshapedSegments[640:,:]

trainY,testY = labels[:640,:],labels[640:,:]

print(trainX.shape)

"""def cnnModel():

model = Sequential()

# adding the first convolutionial layer with 32 filters and 5 by 5 kernal size, using the rectifier as the activation function

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu', input\_shape = (90,8,1)))

model.add(BatchNormalization())

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation = "relu",use\_bias = True))

model.add(Dense(1024, activation = "relu",use\_bias = True))

model.add(Dense(2048, activation = "relu",use\_bias = True))

model.add(Dense(4096, activation = "relu",use\_bias = True))

#model.add(Dropout(0.5))

#model.add(Dropout(0.5))

#model.add(Dense(5, activation = "softmax"))

return model

# Compiling the model to generate a model

# model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])"""

input\_img = Input(shape = (90, 8, 1))

tower\_1 = Conv2D(64, (1,1), padding='same', activation='relu')(input\_img)

tower\_1 = Conv2D(64, (3,3), padding='same', activation='relu')(tower\_1)

tower\_2 = Conv2D(64, (1,1), padding='same', activation='relu')(input\_img)

tower\_2 = Conv2D(64, (5,5), padding='same', activation='relu')(tower\_2)

tower\_3 = MaxPool2D((3,3), strides=(1,1), padding='same')(input\_img)

tower\_3 = Conv2D(64, (1,1), padding='same', activation='relu')(tower\_3)

output = keras.layers.concatenate([tower\_1, tower\_2, tower\_3], axis = 3)

output = Flatten()(output)

out = Dense(5, activation='softmax')(output)

model = Model(inputs = input\_img, outputs = out)

#optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

model.compile(optimizer = 'adam', loss = "categorical\_crossentropy", metrics=["accuracy"])

for layer in model.layers:

print(layer.name)

print(len(testX))

print(model.summary())

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',

patience=3,

verbose=1,

factor=0.5,

min\_lr=0.00001)

datagen = ImageDataGenerator(

featurewise\_center=False, # set input mean to 0 over the dataset

samplewise\_center=False, # set each sample mean to 0

featurewise\_std\_normalization=False, # divide inputs by std of the dataset

samplewise\_std\_normalization=False, # divide each input by its std

zca\_whitening=False, # apply ZCA whitening

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.15, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=True, # randomly flip images

vertical\_flip=True) # randomly flip images

datagen.fit(trainX)

history = model.fit\_generator(datagen.flow(trainX,trainY, batch\_size=4),

epochs = 200, validation\_data = (testX,testY),

verbose = 2, steps\_per\_epoch=trainX.shape[0] // 4

, callbacks=[learning\_rate\_reduction])

#model.fit(trainX,trainY,epochs=200,batch\_size=4,verbose=2)

score = model.evaluate(testX,testY,verbose=1)

print(score)

y\_pred = model.predict(testX)

a = np.argmax(y\_pred,axis=1)

print(a)

print('Baseline Error: %.2f%%' %(100-score[1]\*100))

model.save('model\_EMG\_cnnnet.h5')

np.save('groundTruth\_cnnnet.npy',testY)

np.save('testData\_cnnnet.npy',testX)

for z in range(len(a)):

q = open('est\_cnnnet.txt','a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

elif testY[z][4] == 1:

co4 = co4+1

if testY[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif testY[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif testY[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif testY[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

elif testY[z][4] == 1 and a[z] == 4:

c4 = c4+1

q.write("line %d has"%z + "3\n")

foo = open("expectation\_cnnnet.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + ", " + str(co4) + ", " + str(co5) + "\n")

boo = open("correct\_caps\_cnnnet.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + ", " + str(co4) + ", " + str(co5) + "\n")

3. Code for DenseNet

import numpy as np

import pandas as pd

import os

import keras

os.environ["CUDA\_VISIBLE\_DEVICES"]="2"

from keras.models import Model

from keras.layers.core import Dense, Dropout, Activation

from keras.layers.convolutional import Convolution2D

from keras.layers.pooling import AveragePooling2D

from keras.layers.pooling import GlobalAveragePooling2D

from keras.layers import Input

from keras.layers.merge import Concatenate

from keras.layers.normalization import BatchNormalization

from keras.regularizers import l2

import keras.backend as K

from keras.utils.layer\_utils import convert\_all\_kernels\_in\_model, convert\_dense\_weights\_data\_format

from keras.utils.data\_utils import get\_file

from keras.engine.topology import get\_source\_inputs

import tensorflow as tf

from keras.callbacks import ModelCheckpoint

from keras.callbacks import CSVLogger

from keras.optimizers import RMSprop

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau

import matplotlib.pyplot as plt

img\_dim = (90,8,1)

densenet\_depth = 40

densenet\_growth\_rate = 12

csv\_logger = CSVLogger('tr\_final.csv', append=True, separator=';')

co0=0

co1=0

co2=0

co3=0

co4=0

co5=0

c0=0

c1=0

c2=0

c3=0

c4=0

c5=0

p = 0

class TestCallback(keras.callbacks.Callback):

def \_\_init\_\_(self, test\_data):

self.test\_data = test\_data

def on\_epoch\_end(self, epoch, logs={}):

global co0,co1,co2,co3,co4,co5,c0,c1,c2,c3,c4,c5,p

x, y = self.test\_data

a = model.predict\_classes(x)

for z in range(len(x)):

q = open('est\_CNN\_final %d.txt'%p,'a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

if y[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif y[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif y[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif y[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

p = p+1

loss, acc = self.model.evaluate(x, y, verbose=1)

print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))

bar = open("test\_epochs\_CNN\_final.csv",'a')

bar.write(str(loss) + ", " + str(acc) + "\n")

foo = open("expectation\_CNN\_final.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + "," + str(co4) + "," + str(co5) + "\n")

boo = open("correct\_CNN\_final.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + str(co4) + "," + str(co5) + "\n")

#K.set\_image\_dim\_ordering('th')

# setting up a random seed for reproducibility

random\_seed = 611

np.random.seed(random\_seed)

dataset = []

label= np.empty((0))

r = []

q = ['t','i','r','l','s']

s = ['t','i','r']

with tf.device('/gpu:2'):

for k in range(9):

path = "/data/venki/myonet/trainingsets/sub%d"%(k+1)

os.chdir(path)

for j in range(5):

for i in range(20):

if k > 6:

if j <=2:

count = 0

with open(s[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

else:

break

else:

count = 0

with open(q[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

labels = np.asarray(pd.get\_dummies(label),dtype = np.int8)

print(len(dataset))

#print(dataset)

d = np.array(dataset)

print(d.shape)

segments = d.reshape(820,90,8)

numOfRows = segments.shape[1]

numOfColumns = segments.shape[2]

numChannels = 1

numFilters = 256 # number of filters in Conv2D layer

# kernal size of the Conv2D layer

kernalSize1 = 2

# max pooling window size

poolingWindowSz = 2

# number of filters in fully connected layers

numNueronsFCL1 = 256

numNueronsFCL2 = 256

# split ratio for test and validation

# number of epochs

Epochs = 200

# batchsize

batchSize = 4

# number of total clases

numClasses = labels.shape[1]

# dropout ratio for dropout layer

dropOutRatio = 0.1

# reshaping the data for network input

reshapedSegments = segments.reshape(segments.shape[0], numOfRows, numOfColumns,1)

# splitting in training and testing data

"""trainSplitRatio = 0.60

trainSplit = np.random.rand(len(reshapedSegments)) < trainSplitRatio

trainX = reshapedSegments[trainSplit]

testX = reshapedSegments[~trainSplit]

trainX = np.nan\_to\_num(trainX)

testX = np.nan\_to\_num(testX)

trainY = labels[trainSplit]

testY = labels[~trainSplit]

zprint(trainX.shape)"""

trainX,testX = reshapedSegments[:640,:],reshapedSegments[640:,:]

trainY,testY = labels[:640,:],labels[640:,:]

print(trainX.shape)

"""def cnnModel():

model = Sequential()

# adding the first convolutionial layer with 32 filters and 5 by 5 kernal size, using the rectifier as the activation function

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu', input\_shape = (90,8,1)))

model.add(BatchNormalization())

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation = "relu",use\_bias = True))

model.add(Dense(1024, activation = "relu",use\_bias = True))

model.add(Dense(2048, activation = "relu",use\_bias = True))

model.add(Dense(4096, activation = "relu",use\_bias = True))

#model.add(Dropout(0.5))

#model.add(Dropout(0.5))

#model.add(Dense(5, activation = "softmax"))

return model

# Compiling the model to generate a model

# model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])"""

"""tower\_1 = Conv2D(64, (1,1), padding='same', activation='relu')(input\_img)

tower\_1 = Conv2D(64, (3,3), padding='same', activation='relu')(tower\_1)

tower\_2 = Conv2D(64, (1,1), padding='same', activation='relu')(input\_img)

tower\_2 = Conv2D(64, (5,5), padding='same', activation='relu')(tower\_2)

tower\_3 = MaxPool2D((3,3), strides=(1,1), padding='same')(input\_img)

tower\_3 = Conv2D(64, (1,1), padding='same', activation='relu')(tower\_3)

output = keras.layers.concatenate([tower\_1, tower\_2, tower\_3], axis = 3)

output = Flatten()(output)

out = Dense(5, activation='softmax')(output)

model = Model(inputs = input\_img, outputs = out)"""

def conv\_block(input, nb\_filter, dropout\_rate=None, weight\_decay=1E-4):

''' Apply BatchNorm, Relu 3x3, Conv2D, optional dropout

Args:

input: Input keras tensor

nb\_filter: number of filters

dropout\_rate: dropout rate

weight\_decay: weight decay factor

Returns: keras tensor with batch\_norm, relu and convolution2d added

'''

x = Activation('relu')(input)

x = Convolution2D(nb\_filter, (3, 3), kernel\_initializer="he\_uniform", padding="same", use\_bias=False,

kernel\_regularizer=l2(weight\_decay))(x)

if dropout\_rate is not None:

x = Dropout(dropout\_rate)(x)

return x

def transition\_block(input, nb\_filter, dropout\_rate=None, weight\_decay=1E-4):

''' Apply BatchNorm, Relu 1x1, Conv2D, optional dropout and Maxpooling2D

Args:

input: keras tensor

nb\_filter: number of filters

dropout\_rate: dropout rate

weight\_decay: weight decay factor

Returns: keras tensor, after applying batch\_norm, relu-conv, dropout, maxpool

'''

concat\_axis = 1 if K.image\_dim\_ordering() == "th" else -1

x = Convolution2D(nb\_filter, (1, 1), kernel\_initializer="he\_uniform", padding="same", use\_bias=False,

kernel\_regularizer=l2(weight\_decay))(input)

if dropout\_rate is not None:

x = Dropout(dropout\_rate)(x)

x = AveragePooling2D((2, 2), strides=(2, 2))(x)

x = BatchNormalization(axis=concat\_axis, gamma\_regularizer=l2(weight\_decay),

beta\_regularizer=l2(weight\_decay))(x)

return x

def dense\_block(x, nb\_layers, nb\_filter, growth\_rate, dropout\_rate=None, weight\_decay=1E-4):

''' Build a dense\_block where the output of each conv\_block is fed to subsequent ones

Args:

x: keras tensor

nb\_layers: the number of layers of conv\_block to append to the model.

nb\_filter: number of filters

growth\_rate: growth rate

dropout\_rate: dropout rate

weight\_decay: weight decay factor

Returns: keras tensor with nb\_layers of conv\_block appended

'''

concat\_axis = 1 if K.image\_dim\_ordering() == "th" else -1

feature\_list = [x]

for i in range(nb\_layers):

x = conv\_block(x, growth\_rate, dropout\_rate, weight\_decay)

feature\_list.append(x)

x = Concatenate(axis=concat\_axis)(feature\_list)

nb\_filter += growth\_rate

return x, nb\_filter

def createDenseNet(nb\_classes, img\_dim, depth=40, nb\_dense\_block=3, growth\_rate=12, nb\_filter=16, dropout\_rate=None,

weight\_decay=1E-4, verbose=True):

''' Build the create\_dense\_net model

Args:

nb\_classes: number of classes

img\_dim: tuple of shape (channels, rows, columns) or (rows, columns, channels)

depth: number or layers

nb\_dense\_block: number of dense blocks to add to end

growth\_rate: number of filters to add

nb\_filter: number of filters

dropout\_rate: dropout rate

weight\_decay: weight decay

Returns: keras tensor with nb\_layers of conv\_block appended

'''

model\_input = Input(shape=img\_dim)

concat\_axis = 1 if K.image\_dim\_ordering() == "th" else -1

assert (depth - 4) % 3 == 0, "Depth must be 3 N + 4"

# layers in each dense block

nb\_layers = int((depth - 4) / 3)

# Initial convolution

x = Convolution2D(nb\_filter, (3, 3), kernel\_initializer="he\_uniform", padding="same", name="initial\_conv2D", use\_bias=False,kernel\_regularizer=l2(weight\_decay))(model\_input)

x = BatchNormalization(axis=concat\_axis, gamma\_regularizer=l2(weight\_decay),

beta\_regularizer=l2(weight\_decay))(x)

# Add dense blocks

for block\_idx in range(nb\_dense\_block - 1):

x, nb\_filter = dense\_block(x, nb\_layers, nb\_filter, growth\_rate, dropout\_rate=dropout\_rate,

weight\_decay=weight\_decay)

# add transition\_block

x = transition\_block(x, nb\_filter, dropout\_rate=dropout\_rate, weight\_decay=weight\_decay)

# The last dense\_block does not have a transition\_block

x, nb\_filter = dense\_block(x, nb\_layers, nb\_filter, growth\_rate, dropout\_rate=dropout\_rate,

weight\_decay=weight\_decay)

x = Activation('relu')(x)

x = GlobalAveragePooling2D()(x)

x = Dense(nb\_classes, activation='softmax', kernel\_regularizer=l2(weight\_decay), bias\_regularizer=l2(weight\_decay))(x)

densenet = Model(inputs=model\_input, outputs=x)

if verbose:

print("DenseNet-%d-%d created." % (depth, growth\_rate))

return densenet

model = createDenseNet(nb\_classes=5,img\_dim=img\_dim,depth=densenet\_depth,

growth\_rate = densenet\_growth\_rate)

#optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

model.compile(optimizer = 'adam', loss = "categorical\_crossentropy", metrics=["accuracy"])

for layer in model.layers:

print(layer.name)

print(len(testX))

print(model.summary())

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',

patience=3,

verbose=1,

factor=0.5,

min\_lr=0.00001)

datagen = ImageDataGenerator(

featurewise\_center=False, # set input mean to 0 over the dataset

samplewise\_center=False, # set each sample mean to 0

featurewise\_std\_normalization=False, # divide inputs by std of the dataset

samplewise\_std\_normalization=False, # divide each input by its std

zca\_whitening=False, # apply ZCA whitening

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.15, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=True, # randomly flip images

vertical\_flip=True) # randomly flip images

datagen.fit(trainX)

history = model.fit\_generator(datagen.flow(trainX,trainY, batch\_size=4),

epochs = 200, validation\_data = (testX,testY),

verbose = 2, steps\_per\_epoch=trainX.shape[0] // 4

, callbacks=[learning\_rate\_reduction])

#model.fit(trainX,trainY,epochs=200,batch\_size=4,verbose=2)

score = model.evaluate(testX,testY,verbose=1)

print(score)

y\_pred = model.predict(testX)

a = np.argmax(y\_pred,axis=1)

print(a)

print('Baseline Error: %.2f%%' %(100-score[1]\*100))

model.save('model\_EMG\_cnnnet.h5')

np.save('groundTruth\_cnnnet.npy',testY)

np.save('testData\_cnnnet.npy',testX)

for z in range(len(a)):

q = open('est\_cnnnet.txt','a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

elif testY[z][4] == 1:

co4 = co4+1

if testY[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif testY[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif testY[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif testY[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

elif testY[z][4] == 1 and a[z] == 4:

c4 = c4+1

q.write("line %d has"%z + "3\n")

foo = open("expectation\_cnnnet.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + ", " + str(co4) + ", " + str(co5) + "\n")

boo = open("correct\_caps\_cnnnet.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + ", " + str(co4) + ", " + str(co5) + "\n")

4. Code for Inception net v4:

import numpy as np

import pandas as pd

import os

import keras

os.environ["CUDA\_VISIBLE\_DEVICES"]="2"

from keras.layers import Input

from keras.layers.merge import concatenate

from keras.layers import Dense, Dropout, Flatten, Activation, Conv2D

from keras.layers.convolutional import MaxPooling2D, AveragePooling2D

from keras.layers.normalization import BatchNormalization

from keras.models import Model

import tensorflow as tf

from keras.callbacks import ModelCheckpoint

from keras.callbacks import CSVLogger

from keras.optimizers import RMSprop

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import ReduceLROnPlateau

import matplotlib.pyplot as plt

img\_dim = (90,8,1)

densenet\_depth = 40

densenet\_growth\_rate = 12

csv\_logger = CSVLogger('tr\_final.csv', append=True, separator=';')

co0=0

co1=0

co2=0

co3=0

co4=0

co5=0

c0=0

c1=0

c2=0

c3=0

c4=0

c5=0

p = 0

class TestCallback(keras.callbacks.Callback):

def \_\_init\_\_(self, test\_data):

self.test\_data = test\_data

def on\_epoch\_end(self, epoch, logs={}):

global co0,co1,co2,co3,co4,co5,c0,c1,c2,c3,c4,c5,p

x, y = self.test\_data

a = model.predict\_classes(x)

for z in range(len(x)):

q = open('est\_CNN\_final %d.txt'%p,'a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

if y[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif y[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif y[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif y[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

p = p+1

loss, acc = self.model.evaluate(x, y, verbose=1)

print('\nTesting loss: {}, acc: {}\n'.format(loss, acc))

bar = open("test\_epochs\_CNN\_final.csv",'a')

bar.write(str(loss) + ", " + str(acc) + "\n")

foo = open("expectation\_CNN\_final.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + "," + str(co4) + "," + str(co5) + "\n")

boo = open("correct\_CNN\_final.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + str(co4) + "," + str(co5) + "\n")

#K.set\_image\_dim\_ordering('th')

# setting up a random seed for reproducibility

random\_seed = 611

np.random.seed(random\_seed)

dataset = []

label= np.empty((0))

r = []

q = ['t','i','r','l','s']

s = ['t','i','r']

with tf.device('/gpu:2'):

for k in range(9):

path = "/data/venki/myonet/trainingsets/sub%d"%(k+1)

os.chdir(path)

for j in range(5):

for i in range(20):

if k > 6:

if j <=2:

count = 0

with open(s[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

else:

break

else:

count = 0

with open(q[j]+'%d.txt'%(i+1),'r') as f:

lines = (f.read().splitlines())

print(f.name)

lines = [w.replace("[", '') for w in lines]

lines = [w.replace(']', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

lines = [w.replace(' ', '') for w in lines]

print(len(lines))

for l in lines:

li = l.split(",")

li = [float(ele) for ele in li]

count = count + 1

if count <= 90:

dataset.append(li)

label = np.append(label, j)

print(label)

labels = np.asarray(pd.get\_dummies(label),dtype = np.int8)

print(len(dataset))

#print(dataset)

d = np.array(dataset)

print(d.shape)

segments = d.reshape(820,90,8)

numOfRows = segments.shape[1]

numOfColumns = segments.shape[2]

numChannels = 1

numFilters = 256 # number of filters in Conv2D layer

# kernal size of the Conv2D layer

kernalSize1 = 2

# max pooling window size

poolingWindowSz = 2

# number of filters in fully connected layers

numNueronsFCL1 = 256

numNueronsFCL2 = 256

# split ratio for test and validation

# number of epochs

Epochs = 200

# batchsize

batchSize = 4

# number of total clases

numClasses = labels.shape[1]

# dropout ratio for dropout layer

dropOutRatio = 0.1

# reshaping the data for network input

reshapedSegments = segments.reshape(segments.shape[0], numOfRows, numOfColumns,1)

# splitting in training and testing data

"""trainSplitRatio = 0.60

trainSplit = np.random.rand(len(reshapedSegments)) < trainSplitRatio

trainX = reshapedSegments[trainSplit]

testX = reshapedSegments[~trainSplit]

trainX = np.nan\_to\_num(trainX)

testX = np.nan\_to\_num(testX)

trainY = labels[trainSplit]

testY = labels[~trainSplit]

zprint(trainX.shape)"""

trainX,testX = reshapedSegments[:640,:],reshapedSegments[640:,:]

trainY,testY = labels[:640,:],labels[640:,:]

print(trainX.shape)

"""def cnnModel():

model = Sequential()

# adding the first convolutionial layer with 32 filters and 5 by 5 kernal size, using the rectifier as the activation function

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu', input\_shape = (90,8,1)))

model.add(BatchNormalization())

model.add(Conv2D(filters = 8, kernel\_size = (7,7),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 16, kernel\_size = (5,5),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(2,2)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 32, kernel\_size = (3,3),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(Conv2D(filters = 64, kernel\_size = (2,2),padding = 'Same',

activation ='relu'))

model.add(BatchNormalization())

model.add(MaxPool2D(pool\_size=(1,1)))

#model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation = "relu",use\_bias = True))

model.add(Dense(1024, activation = "relu",use\_bias = True))

model.add(Dense(2048, activation = "relu",use\_bias = True))

model.add(Dense(4096, activation = "relu",use\_bias = True))

#model.add(Dropout(0.5))

#model.add(Dropout(0.5))

#model.add(Dense(5, activation = "softmax"))

return model

# Compiling the model to generate a model

# model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])"""

"""tower\_1 = Conv2D(64, (1,1), padding='same', activation='relu')(input\_img)

tower\_1 = Conv2D(64, (3,3), padding='same', activation='relu')(tower\_1)

tower\_2 = Conv2D(64, (1,1), padding='same', activation='relu')(input\_img)

tower\_2 = Conv2D(64, (5,5), padding='same', activation='relu')(tower\_2)

tower\_3 = MaxPool2D((3,3), strides=(1,1), padding='same')(input\_img)

tower\_3 = Conv2D(64, (1,1), padding='same', activation='relu')(tower\_3)

output = keras.layers.concatenate([tower\_1, tower\_2, tower\_3], axis = 3)

output = Flatten()(output)

out = Dense(5, activation='softmax')(output)

model = Model(inputs = input\_img, outputs = out)"""

"""def conv\_block(input, nb\_filter, dropout\_rate=None, weight\_decay=1E-4):

''' Apply BatchNorm, Relu 3x3, Conv2D, optional dropout

Args:

input: Input keras tensor

nb\_filter: number of filters

dropout\_rate: dropout rate

weight\_decay: weight decay factor

Returns: keras tensor with batch\_norm, relu and convolution2d added

'''

x = Activation('relu')(input)

x = Convolution2D(nb\_filter, (3, 3), kernel\_initializer="he\_uniform", padding="same", use\_bias=False,

kernel\_regularizer=l2(weight\_decay))(x)

if dropout\_rate is not None:

x = Dropout(dropout\_rate)(x)

return x

def transition\_block(input, nb\_filter, dropout\_rate=None, weight\_decay=1E-4):

''' Apply BatchNorm, Relu 1x1, Conv2D, optional dropout and Maxpooling2D

Args:

input: keras tensor

nb\_filter: number of filters

dropout\_rate: dropout rate

weight\_decay: weight decay factor

Returns: keras tensor, after applying batch\_norm, relu-conv, dropout, maxpool

'''

concat\_axis = 1 if K.image\_dim\_ordering() == "th" else -1

x = Convolution2D(nb\_filter, (1, 1), kernel\_initializer="he\_uniform", padding="same", use\_bias=False,

kernel\_regularizer=l2(weight\_decay))(input)

if dropout\_rate is not None:

x = Dropout(dropout\_rate)(x)

x = AveragePooling2D((2, 2), strides=(2, 2))(x)

x = BatchNormalization(axis=concat\_axis, gamma\_regularizer=l2(weight\_decay),

beta\_regularizer=l2(weight\_decay))(x)

return x

def dense\_block(x, nb\_layers, nb\_filter, growth\_rate, dropout\_rate=None, weight\_decay=1E-4):

''' Build a dense\_block where the output of each conv\_block is fed to subsequent ones

Args:

x: keras tensor

nb\_layers: the number of layers of conv\_block to append to the model.

nb\_filter: number of filters

growth\_rate: growth rate

dropout\_rate: dropout rate

weight\_decay: weight decay factor

Returns: keras tensor with nb\_layers of conv\_block appended

'''

concat\_axis = 1 if K.image\_dim\_ordering() == "th" else -1

feature\_list = [x]

for i in range(nb\_layers):

x = conv\_block(x, growth\_rate, dropout\_rate, weight\_decay)

feature\_list.append(x)

x = Concatenate(axis=concat\_axis)(feature\_list)

nb\_filter += growth\_rate

return x, nb\_filter

def createDenseNet(nb\_classes, img\_dim, depth=40, nb\_dense\_block=3, growth\_rate=12, nb\_filter=16, dropout\_rate=None,

weight\_decay=1E-4, verbose=True):

''' Build the create\_dense\_net model

Args:

nb\_classes: number of classes

img\_dim: tuple of shape (channels, rows, columns) or (rows, columns, channels)

depth: number or layers

nb\_dense\_block: number of dense blocks to add to end

growth\_rate: number of filters to add

nb\_filter: number of filters

dropout\_rate: dropout rate

weight\_decay: weight decay

Returns: keras tensor with nb\_layers of conv\_block appended

'''

model\_input = Input(shape=img\_dim)

concat\_axis = 1 if K.image\_dim\_ordering() == "th" else -1

assert (depth - 4) % 3 == 0, "Depth must be 3 N + 4"

# layers in each dense block

nb\_layers = int((depth - 4) / 3)

# Initial convolution

x = Convolution2D(nb\_filter, (3, 3), kernel\_initializer="he\_uniform", padding="same", name="initial\_conv2D", use\_bias=False,kernel\_regularizer=l2(weight\_decay))(model\_input)

x = BatchNormalization(axis=concat\_axis, gamma\_regularizer=l2(weight\_decay),

beta\_regularizer=l2(weight\_decay))(x)

# Add dense blocks

for block\_idx in range(nb\_dense\_block - 1):

x, nb\_filter = dense\_block(x, nb\_layers, nb\_filter, growth\_rate, dropout\_rate=dropout\_rate,

weight\_decay=weight\_decay)

# add transition\_block

x = transition\_block(x, nb\_filter, dropout\_rate=dropout\_rate, weight\_decay=weight\_decay)

# The last dense\_block does not have a transition\_block

x, nb\_filter = dense\_block(x, nb\_layers, nb\_filter, growth\_rate, dropout\_rate=dropout\_rate,

weight\_decay=weight\_decay)

x = Activation('relu')(x)

x = GlobalAveragePooling2D()(x)

x = Dense(nb\_classes, activation='softmax', kernel\_regularizer=l2(weight\_decay), bias\_regularizer=l2(weight\_decay))(x)

densenet = Model(inputs=model\_input, outputs=x)

if verbose:

print("DenseNet-%d-%d created." % (depth, growth\_rate))

return densenet

model = createDenseNet(nb\_classes=5,img\_dim=img\_dim,depth=densenet\_depth,

growth\_rate = densenet\_growth\_rate)"""

def conv\_block(x, nb\_filter, nb\_row, nb\_col, padding = "same", strides = (1, 1), use\_bias = False):

'''Defining a Convolution block that will be used throughout the network.'''

x = Conv2D(nb\_filter, (nb\_row, nb\_col), strides = strides, padding = padding, use\_bias = use\_bias)(x)

x = BatchNormalization(axis = -1, momentum = 0.9997, scale = False)(x)

x = Activation("relu")(x)

return x

def stem(input):

'''The stem of the pure Inception-v4 and Inception-ResNet-v2 networks. This is input part of those networks.'''

# Input shape is 299 \* 299 \* 3 (Tensorflow dimension ordering)

x = conv\_block(input, 32, 3, 3, strides = (2, 2), padding = "same") # 149 \* 149 \* 32

x = conv\_block(x, 32, 3, 3, padding = "same") # 147 \* 147 \* 32

x = conv\_block(x, 64, 3, 3) # 147 \* 147 \* 64

x1 = MaxPooling2D((3, 3), strides = (2, 2), padding = "same")(x)

x2 = conv\_block(x, 96, 3, 3, strides = (2, 2), padding = "same")

x = concatenate([x1, x2], axis = -1) # 73 \* 73 \* 160

x1 = conv\_block(x, 64, 1, 1)

x1 = conv\_block(x1, 96, 3, 3, padding = "same")

x2 = conv\_block(x, 64, 1, 1)

x2 = conv\_block(x2, 64, 1, 7)

x2 = conv\_block(x2, 64, 7, 1)

x2 = conv\_block(x2, 96, 3, 3, padding = "same")

x = concatenate([x1, x2], axis = -1) # 71 \* 71 \* 192

x1 = conv\_block(x, 192, 3, 3, strides = (2, 2), padding = "same")

x2 = MaxPooling2D((3, 3), strides = (2, 2), padding = "same")(x)

x = concatenate([x1, x2], axis = -1) # 35 \* 35 \* 384

return x

def inception\_A(input):

'''Architecture of Inception\_A block which is a 35 \* 35 grid module.'''

a1 = AveragePooling2D((3, 3), strides = (1, 1), padding = "same")(input)

a1 = conv\_block(a1, 96, 1, 1)

a2 = conv\_block(input, 96, 1, 1)

a3 = conv\_block(input, 64, 1, 1)

a3 = conv\_block(a3, 96, 3, 3)

a4 = conv\_block(input, 64, 1, 1)

a4 = conv\_block(a4, 96, 3, 3)

a4 = conv\_block(a4, 96, 3, 3)

merged = concatenate([a1, a2, a3, a4], axis = -1)

return merged

def inception\_B(input):

'''Architecture of Inception\_B block which is a 17 \* 17 grid module.'''

b1 = AveragePooling2D((3, 3), strides = (1, 1), padding = "same")(input)

b1 = conv\_block(b1, 128, 1, 1)

b2 = conv\_block(input, 384, 1, 1)

b3 = conv\_block(input, 192, 1, 1)

b3 = conv\_block(b3, 224, 1, 7)

b3 = conv\_block(b3, 256, 7, 1)

b4 = conv\_block(input, 192, 1, 1)

b4 = conv\_block(b4, 192, 7, 1)

b4 = conv\_block(b4, 224, 1, 7)

b4 = conv\_block(b4, 224, 7, 1)

b4 = conv\_block(b4, 256, 1, 7)

merged = concatenate([b1, b2, b3, b4], axis = -1)

return merged

def inception\_C(input):

'''Architecture of Inception\_C block which is a 8 \* 8 grid module.'''

c1 = AveragePooling2D((3, 3), strides = (1, 1), padding = "same")(input)

c1 = conv\_block(c1, 256, 1, 1)

c2 = conv\_block(input, 256, 1, 1)

c3 = conv\_block(input, 384, 1, 1)

c31 = conv\_block(c2, 256, 1, 3)

c32 = conv\_block(c2, 256, 3, 1)

c3 = concatenate([c31, c32], axis = -1)

c4 = conv\_block(input, 384, 1, 1)

c4 = conv\_block(c3, 448, 3, 1)

c4 = conv\_block(c3, 512, 1, 3)

c41 = conv\_block(c3, 256, 1, 3)

c42 = conv\_block(c3, 256, 3, 1)

c4 = concatenate([c41, c42], axis = -1)

merged = concatenate([c1, c2, c3, c4], axis = -1)

return merged

def reduction\_A(input, k = 192, l = 224, m = 256, n = 384):

'''Architecture of a 35 \* 35 to 17 \* 17 Reduction\_A block.'''

ra1 = MaxPooling2D((3, 3), strides = (2, 2), padding = "same")(input)

ra2 = conv\_block(input, n, 3, 3, strides = (2, 2), padding = "same")

ra3 = conv\_block(input, k, 1, 1)

ra3 = conv\_block(ra3, l, 3, 3)

ra3 = conv\_block(ra3, m, 3, 3, strides = (2, 2), padding = "same")

merged = concatenate([ra1, ra2, ra3], axis = -1)

return merged

def reduction\_B(input):

'''Architecture of a 17 \* 17 to 8 \* 8 Reduction\_B block.'''

rb1 = MaxPooling2D((3, 3), strides = (2, 2), padding = "same")(input)

rb2 = conv\_block(input, 192, 1, 1)

rb2 = conv\_block(rb2, 192, 3, 3, strides = (2, 2), padding = "same")

rb3 = conv\_block(input, 256, 1, 1)

rb3 = conv\_block(rb3, 256, 1, 7)

rb3 = conv\_block(rb3, 320, 7, 1)

rb3 = conv\_block(rb3, 320, 3, 3, strides = (2, 2), padding = "same")

merged = concatenate([rb1, rb2, rb3], axis = -1)

return merged

def inception\_v4(nb\_classes = 5, load\_weights = True):

'''Creates the Inception\_v4 network.'''

init = Input((90, 8, 1)) # Channels last, as using Tensorflow backend with Tensorflow image dimension ordering

# Input shape is 299 \* 299 \* 3

x = stem(init) # Output: 35 \* 35 \* 384

# 4 x Inception A

for i in range(4):

x = inception\_A(x)

# Output: 35 \* 35 \* 384

# Reduction A

x = reduction\_A(x, k = 192, l = 224, m = 256, n = 384) # Output: 17 \* 17 \* 1024

# 7 x Inception B

for i in range(7):

x = inception\_B(x)

# Output: 17 \* 17 \* 1024

# Reduction B

x = reduction\_B(x) # Output: 8 \* 8 \* 1536

# 3 x Inception C

for i in range(3):

x = inception\_C(x)

# Output: 8 \* 8 \* 1536

# Average Pooling

x = AveragePooling2D((1, 1))(x) # Output: 1536

# Dropout

x = Dropout(0.2)(x) # Keep dropout 0.2 as mentioned in the paper

x = Flatten()(x) # Output: 1536

# Output layer

output = Dense(units = nb\_classes, activation = "softmax")(x) # Output: 1000

model = Model(init, output, name = "Inception-v4")

return model

# from keras.utils.visualize\_util import plot

model = inception\_v4(load\_weights=True)

#optimizer = RMSprop(lr=0.001, rho=0.9, epsilon=1e-08, decay=0.0)

model.compile(optimizer = 'adam', loss = "categorical\_crossentropy", metrics=["accuracy"])

for layer in model.layers:

print(layer.name)

print(len(testX))

print(model.summary())

learning\_rate\_reduction = ReduceLROnPlateau(monitor='val\_acc',

patience=3,

verbose=1,

factor=0.5,

min\_lr=0.00001)

datagen = ImageDataGenerator(

featurewise\_center=False, # set input mean to 0 over the dataset

samplewise\_center=False, # set each sample mean to 0

featurewise\_std\_normalization=False, # divide inputs by std of the dataset

samplewise\_std\_normalization=False, # divide each input by its std

zca\_whitening=False, # apply ZCA whitening

rotation\_range=10, # randomly rotate images in the range (degrees, 0 to 180)

zoom\_range = 0.15, # Randomly zoom image

width\_shift\_range=0.1, # randomly shift images horizontally (fraction of total width)

height\_shift\_range=0.1, # randomly shift images vertically (fraction of total height)

horizontal\_flip=True, # randomly flip images

vertical\_flip=True) # randomly flip images

datagen.fit(trainX)

history = model.fit\_generator(datagen.flow(trainX,trainY, batch\_size=4),

epochs = 200, validation\_data = (testX,testY),

verbose = 2, steps\_per\_epoch=trainX.shape[0] // 4

, callbacks=[learning\_rate\_reduction])

#model.fit(trainX,trainY,epochs=200,batch\_size=4,verbose=2)

score = model.evaluate(testX,testY,verbose=1)

print(score)

y\_pred = model.predict(testX)

a = np.argmax(y\_pred,axis=1)

print(a)

print('Baseline Error: %.2f%%' %(100-score[1]\*100))

model.save('model\_EMG\_cnnnet.h5')

np.save('groundTruth\_cnnnet.npy',testY)

np.save('testData\_cnnnet.npy',testX)

for z in range(len(a)):

q = open('est\_cnnnet.txt','a')

if testY[z][0] == 1:

co0 = co0+1

elif testY[z][1] == 1:

co1 = co1+1

elif testY[z][2] == 1:

co2 = co2+1

elif testY[z][3] == 1:

co3 = co3+1

elif testY[z][4] == 1:

co4 = co4+1

if testY[z][0] == 1 and a[z] == 0:

c0 = c0+1

q.write("line %d has"%z + "0\n")

elif testY[z][1] == 1 and a[z] == 1:

c1 = c1+1

q.write("line %d has"%z + "1\n")

elif testY[z][2] == 1 and a[z] == 2:

c2 = c2+1

q.write("line %d has"%z + "2\n")

elif testY[z][3] == 1 and a[z] == 3:

c3 = c3+1

q.write("line %d has"%z + "3\n")

elif testY[z][4] == 1 and a[z] == 4:

c4 = c4+1

q.write("line %d has"%z + "3\n")

foo = open("expectation\_cnnnet.txt",'a')

foo.write(str(co0) + ", " + str(co1) + ", " + str(co2) + ", " + str(co3) + ", " + str(co4) + ", " + str(co5) + "\n")

boo = open("correct\_caps\_cnnnet.txt",'a')

boo.write(str(c0) + ", " + str(c1) + ", " + str(c2) + ", " + str(c3) + ", " + str(co4) + ", " + str(co5) + "\n")