Importing the data and setting paths to read inside the python file

project7.py - C:\Users\atche\Desktop\Study zone\AI\Project 7\project7.py (3.8.1)

File Edit Format Run Options Window Help

```
from keras import backend as K

batch_size = 128
nb_classes = 2
nb_epoch = 12

# input image dimensions
img_rows, img_cols = 128, 128
# number of convolutional filters to use
nb_filters = 32
# size of pooling area for max pooling
pool_size = (2, 2)
# convolution kernel size
kernel_size = (3, 3)

# define path to images:

# This is the path of our positive input dataset
pos_im_path = r"positive"

# define the same for negatives
neg_im_path= r"negative"

# read the image files:

pos_im_listing = os.listdir(pos_im_path) # it will read all the files in the pos
neg_im_listing = os.listdir(neg_im_path)
num_pos_samples = size(pos_im_listing) # simply states the total no. of images
num_neg_samples = size(neg_im_listing)

print('training data :')
print('No. of positive samples  '+str(num_pos_samples))# prints the number value
print('No. of Negative samples  '+str(num_neg_samples))
```

```
Using TensorFlow backend.
training data :
No. of positive samples  264
No. of Negative samples  1000
[ 685  575  431 ...  860  189 1175]
normalizing
x_train shape: (843, 128, 128, 1)
Number of images in x_train 843
Number of images in x_test 421
```

Task #1: Implemented and select the best DCNN model that provides a good accuracy for COVID-CT dataset (you should get something around 80s % testing accuracy with any general DCNN model).

For designing the best DCNN model, consider the following criteria:

• Good initialization method (for example: Xavier or he initializer)

• Activation function: ReLU

```
"""shuffle dataset"""
p = np.random.permutation(len(dataset_x))
print(p)
dataset_x = dataset_x[p]
for i in p:
    ds_y.append(dataset_y[i])

X_test = dataset_x[:int(len(dataset_x)/3)]
Y_test = ds_y[:int(len(dataset_x)/3)]
X_train = dataset_x[int(len(dataset_x)/3):]
Y_train = ds_y[int(len(dataset_x)/3):]

print("normalizing")

# Reshaping the array to 4-dims so that it can work with the Keras API
X_train = X_train.reshape(X_train.shape[0], img_rows, img_cols, 1)
X_test = X_test.reshape(X_test.shape[0], img_rows, img_cols, 1)
input_shape = (img_rows, img_cols, 1)
# Making sure that the values are float so that we can get decimal points after division
X_train = X_train.astype('float32')
X_test = X_test.astype('float32')
# Normalizing the RGB codes by dividing it to the max RGB value.
X_train /= 255
X_test /= 255
print('x_train shape:', X_train.shape)
print('Number of images in x_train', X_train.shape[0])
print('Number of images in x_test', X_test.shape[0])
```

Dividing the data into training and test data after normalising the pixel data

```
normalizing
x_train shape: (843, 128, 128, 1)
Number of images in x_train 843
Number of images in x_test 421
```

Model generation and evaluation

Task #2: Evaluated the performance of the model for different optimization functions:

- SGD
- Adam

```
# Importing the required Keras modules containing model and layers
from keras.models import Sequential
from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
# Creating a Sequential Model and adding the layers
model = Sequential()
model.add(Convolution2D(nb_filters, kernel_size[0], kernel_size[1],
                        border_mode='valid',
                        input_shape=input_shape, init='glorot_normal'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Flatten()) # Flattening the 2D arrays for fully connected layers
model.add(Activation('relu'))
model.add(Dropout(0.2))
model.add(Activation('relu'))

model.compile(optimizer='adam',
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])

model.fit(x=X_train,y=Y_train, epochs=10)


print("Miss Rate and Accuracy for adam optimizer: ", model.evaluate(X_test, Y_test))
```

- RMSProp

Same as the above code with RMSProp optimizer

```
 32/421 [=>.............................] - ETA: 3s 64/421 [===>.........................] - ETA: 2s 96/421 [=====>.......................] - ETA: 2s128/421 [========>.....................] - ETA: 2s160/421 [==========>...................] - ETA: 1s192/421 [============>.................] - ETA: 1s224/421 [==============>...............] - ETA: 1s256/421 [================>.............] - ETA: 1s288/421 [==================>...........] - ETA: 1s320/421 [====================>.........] - ETA: 0s352/421 [======================>.....] - ETA: 0s384/421 [========================>...] - ETA: 0s416/421 [==========================>.] - ETA: 0s421/421 [==============================] - 4s 8ms/step
Miss Rate and Accuracy for adam optimizer:  [11.753525149510763, 0.7743467688560486]
```

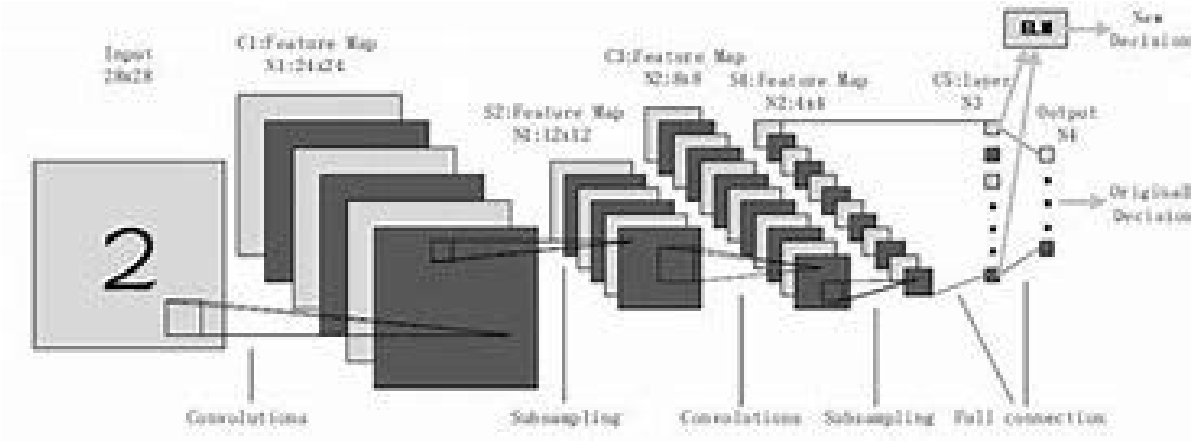
```
 32/421 [=>.............................] - ETA: 3s 64/421 [===>.........................] - ETA: 2s 96/421 [=====>.......................] - ETA: 2s128/421 [========>.....................] - ETA: 2s160/421 [==========>...................] - ETA: 1s192/421 [============>.................] - ETA: 1s224/421 [==============>...............] - ETA: 1s256/421 [================>.............] - ETA: 1s288/421 [==================>...........] - ETA: 1s320/421 [====================>.........] - ETA: 0s352/421 [======================>.....] - ETA: 0s384/421 [========================>...] - ETA: 0s416/421 [==========================>.] - ETA: 0s421/421 [==============================] - 3s 8ms/step
Miss Rate and Accuracy for SGD optimizer:  [11.753525149510763, 0.7743467688560486]
```

Task #3: In your report, show the structure of the model that you are using in this project.

```
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 126, 126, 32)      320
_________________________________________________________________
activation_1 (Activation)    (None, 126, 126, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 124, 124, 32)      9248
_________________________________________________________________
activation_2 (Activation)    (None, 124, 124, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 62, 62, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 60, 60, 64)        18496
_________________________________________________________________
activation_3 (Activation)    (None, 60, 60, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 30, 30, 64)        0
_________________________________________________________________
dropout_2 (Dropout)          (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 28, 28, 128)       73856
_________________________________________________________________
activation_4 (Activation)    (None, 28, 28, 128)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 128)       0
_________________________________________________________________
dropout_3 (Dropout)          (None, 14, 14, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_1 (Dense)              (None, 256)               6422784
_________________________________________________________________
activation_5 (Activation)    (None, 256)               0
_________________________________________________________________
dropout_4 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 514
_________________________________________________________________
activation_6 (Activation)    (None, 2)                 0
=================================================================
Total params: 6,525,218
Trainable params: 6,525,218
Non-trainable params: 0
_________________________________________________________________
None
```

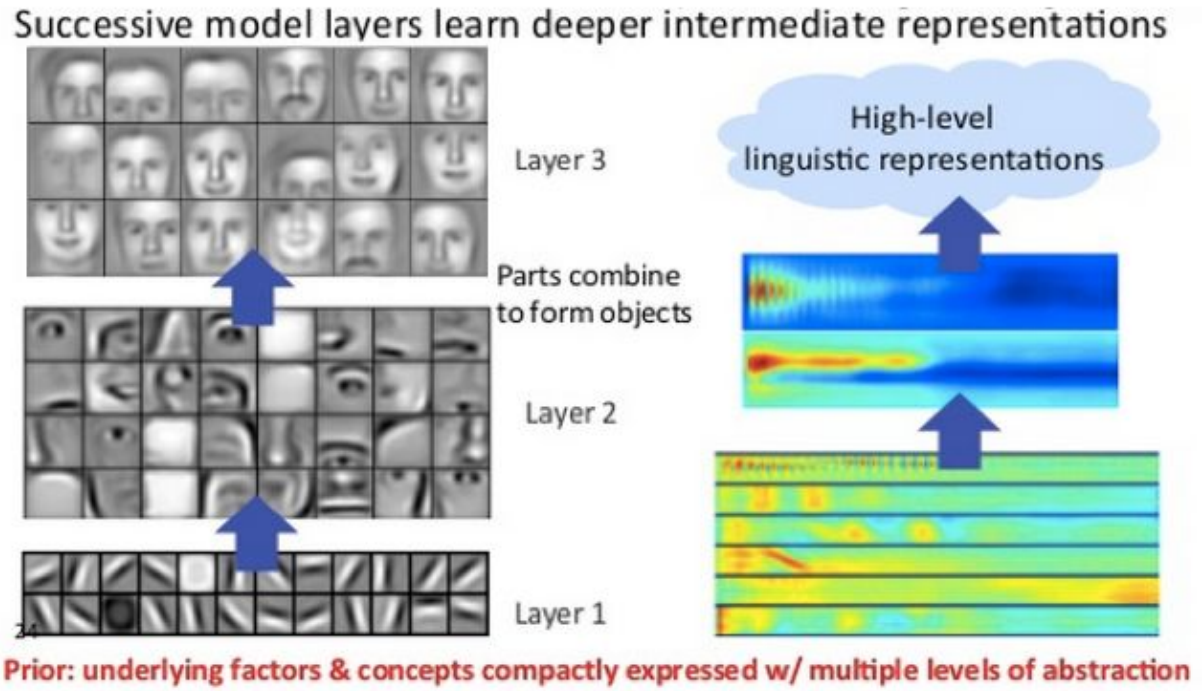
Key Concepts of Deep Neural Networks

Deep-learning networks are distinguished from the more commonplace single-hidden-layer neural networks by their **depth**; that is, the number of node layers through which data must pass in a multistep process of pattern recognition.



Earlier versions of neural networks such as the first perceptrons were shallow, composed of one input and one output layer, and at most one hidden layer in between. More than three layers (including input and output) qualifies as “deep” learning. So *deep* is not just a buzzword to make algorithms seem like they read Sartre and listen to bands you haven’t heard of yet. It is a strictly defined term that means more than one hidden layer.

In deep-learning networks, each layer of nodes trains on a distinct set of features based on the previous layer’s output. The further you advance into the neural net, the more complex the features your nodes can recognize, since they aggregate and recombine features from the previous layer.



This is known as **feature hierarchy**, and it is a hierarchy of increasing complexity and abstraction. It makes deep-learning networks capable of handling very large, high-dimensional data sets with billions of parameters that pass through nonlinear functions.

Above all, these neural nets are capable of discovering latent structures within **unlabeled, unstructured data**, which is the vast majority of data in the world. Another word for unstructured data is *raw media*; i.e. pictures, texts, video and audio recordings. Therefore, one of the problems deep learning solves best is in processing and clustering the world's raw, unlabeled media, discerning similarities and anomalies in data that no human has organized in a relational database or ever put a name to.

For example, deep learning can take a million images, and cluster them according to their similarities: cats in one corner, ice breakers in another, and

in a third all the photos of your grandmother. This is the basis of so-called smart photo albums.

Now apply that same idea to other data types: Deep learning might cluster raw text such as emails or news articles. Emails full of angry complaints might cluster in one corner of the vector space, while satisfied customers, or spambot messages, might cluster in others. This is the basis of various messaging filters, and can be used in customer-relationship management (CRM). The same applies to voice messages.