Neural Network Project

K Abhishek Das

April 2, 2019

1 Topic

I am working on classfying the chexpert data set [1], which is a dataset of chest x-ray images used for the identificaion of pathologies which can be present in the patient. This will help me further in forming the base for the student-teacher reinforcement based learning framework.

2 Dataset

The dataset consists of 224,316 chest radiographs of 65,240 patients labeled for the presence of 14 common chest radiographic observations. Before processing the images I have made them 320x320 and they are all grayscale images.

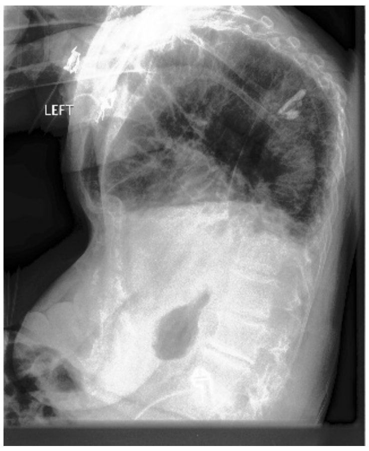


Figure 1: Sample X-Ray

3 DNN Model

For classification I am using a DenseNet121 model, I am using a implementation which is provided by the Keras team itself. Following is a sample declaration of the model for the pathology Atelectasis. For this pathology I have made the uncertain classes to be positive before training the model and all other classed to negative 0. But in case of the other patholoyg Cardiomegaly, I have treated all the 3 classes separately which is uncertain, positive and negative.

```
model = keras.applications.densenet.DenseNet121(
                                    include_top=True,
                                    weights=None,
                                    input_shape=(320, 320, 1),
                                    pooling='max',
                                    classes=2)
```

Input tensor shape = (224316, 320, 320, 1)
Output tensor shape = (224316, 2) for Atelectasis

Output tensor shape = (224316, 3) for Cardiomegaly
Following images give a high level overview of the DenseNet Architecture

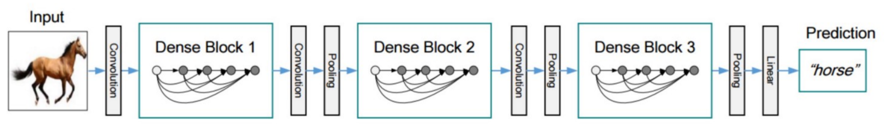
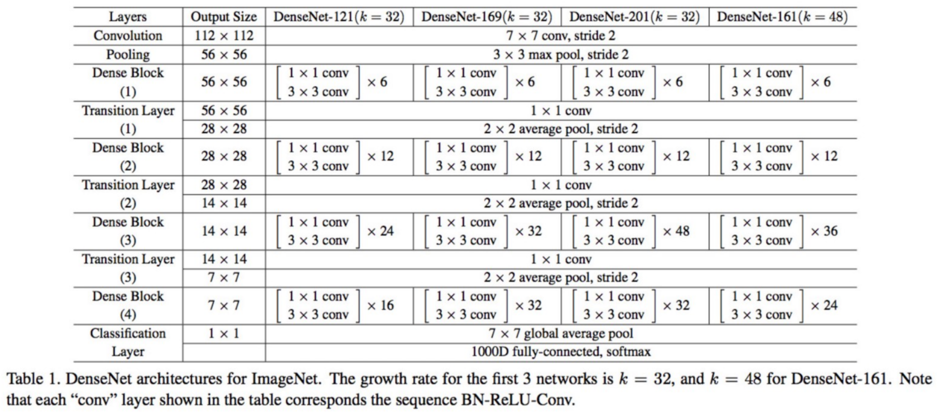


Figure 2. A deep DenseNet with three dense blocks. The layers between two adjacent blocks are referred to as transition layers and change feature map sizes via convolution and pooling.

Figure 2: Overall DenseNet Architecture



Layers	Output Size	DenseNet-121($k = 32$)	DenseNet-169($k = 32$)	DenseNet-201($k = 32$)	DenseNet-161($k = 48$)
Convolution	112×112	7×7 conv, stride 2			
Pooling	56×56	3×3 max pool, stride 2			
Dense Block (1)	56×56	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 6$
Transition Layer (1)	56×56	1×1 conv			
	28×28	2×2 average pool, stride 2			
Dense Block (2)	28×28	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 12$
Transition Layer (2)	28×28	1×1 conv			
	14×14	2×2 average pool, stride 2			
Dense Block (3)	14×14	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 48$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 36$
Transition Layer (3)	14×14	1×1 conv			
	7×7	2×2 average pool, stride 2			
Dense Block (4)	7×7	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 16$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 32$	$\begin{bmatrix} 1 \times 1 \text{ conv} \\ 3 \times 3 \text{ conv} \end{bmatrix} \times 24$
Classification Layer	1×1	7×7 global average pool			
		1000D fully-connected, softmax			

Table 1. DenseNet architectures for ImageNet. The growth rate for the first 3 networks is $k = 32$, and $k = 48$ for DenseNet-161. Note that each "conv" layer shown in the table corresponds the sequence BN-ReLU-Conv.

Figure 3: A Single DenseNet block

4 Hyperparameters

After trying various hyperparameters I went ahead with the following hyperparameters,
learning rate = 10^{-4}
$\beta_1 = 0.9$
$\beta_2 = 0.999$
And batch size of 16
Training time for each single pathology was around 7 hours on Tesla V100 machine

5 Code

Following is the code for data generator since such a huge dataset can't fit into the memory all at once.

```
class DataGenerator(keras.utils.Sequence):
        'Generates␣data␣for␣Keras'

        def __init__(self,
                                list_IDs,
                                labels,
                                batch_size=32,
                                dim=(320, 320),
                                n_channels=1,
                                n_classes=10,
                                shuffle=True):
                self.dim = dim
                self.batch_size = batch_size
                self.labels = labels
                self.list_IDs = list_IDs
                self.n_channels = n_channels
                self.n_classes = n_classes
                self.shuffle = shuffle
                self.on_epoch_end()

        def __len__(self):
                'Denotes␣the␣number␣of␣batches␣per␣epoch'
                return int(np.floor(len(self.list_IDs) / self.batch_size))

        def __getitem__(self, index):
                'Generate␣one␣batch␣of␣data'
                # generates the indexes of the batch
                indexes = self.indexes[index * self.batch_size: (index + 1) * self.
                    ↪ batch_size]

                # find the list of ids
                list_IDs_temp = [self.list_IDs[k] for k in indexes]
                # print(list_IDs_temp)

                # data generation
                X, y = self.__data_generation(list_IDs_temp)

                return X, y

        def on_epoch_end(self):
                'Updates␣indexes␣after␣each␣epoch'
                self.indexes = np.arange(len(self.list_IDs))
                if self.shuffle == True:
                    np.random.shuffle(self.indexes)

        def __data_generation(self, list_IDs_temp):
                'Generates␣data␣containing␣batch_size␣samples' # X : (n_samples, *dim,
                    ↪ n_channels)
                X = np.empty((self.batch_size, *self.dim, self.n_channels))
                y = np.empty((self.batch_size), dtype=int)

                cur_dir = os.getcwd()

                # Generate data
                for i, ID in enumerate(list_IDs_temp):
                    # Store sample

                    img_path = os.path.join(cur_dir, ID)
                    img_path = img_path.replace('home/CheXpert-v1.0-small', 'input/
                        ↪ chexpert')
                    img = image.load_img(img_path, target_size=(WIDTH, HEIGHT),
                        ↪ grayscale=True)

                    X[i,] = image.img_to_array(img) / 255

                    # store the class
                    y[i] = self.labels[ID]

                return X, keras.utils.to_categorical(y, num_classes=self.n_classes)
```

Following is the code for the callbacks and the fitting of the model, the Adam optimizer is used with the set of hyper parameters as mentioned above. I am using categorical crossentropy as the loss function, and accuracy as the metric.

```
callbacks_list = [
    keras.callbacks.EarlyStopping(
        monitor='acc',
        patience=1,
    ),
    keras.callbacks.ModelCheckpoint(
        filepath='chexpert_weights.h5',
        monitor='val_acc',
        save_best_only=True,
    )
]

optimizer = optimizers.Adam(lr=1e-4, beta_1=0.9, beta_2=0.999, epsilon=1e-08)

model.compile(loss='categorical_crossentropy',
                          optimizer=optimizer,
                          metrics=["accuracy"])

model.fit_generator(
        training_generator,
        steps_per_epoch=13963,
        epochs=10,
        callbacks=callbacks_list,
        validation_data=validation_generator,
        validation_steps=14,
)
```

6 Results

For Atelectasis, if we keep the threshold for classification as 0.35 then the f1 score of the validation data comes to be around 0.64583 and the validation accuracy comes to be around 0.65811.

For Cardiomegaly, if we keep the threshold for classification as 0.35 then the f1 score of the validation data comes to be around 0.6279 and the validation accuracy comes to be around 0.7370.

7 Running the code

To run the gui demo open the folder and type in terminal

```
python3 gui_chexpert_rl_python3.py
```

Here is the link for the demo

References

[1] https://stanfordmlgroup.github.io/competitions/chexpert/
Chexpert Dataset.