Identifying Pneumonia by Chest X-Ray Images with Deep Learning

Name: Heng Wang / Jiahao Zhao

ID: 001822598 / 001839375

Abstract

In our final project, the technical stack includes: deep learning, transfer learning, and data augmentation, using TensorFlow and Keras. We implement Data Augment to deal with imbalanced data, used transfer learning from VGG16 to initialize weights of first few layers, and built a depthwise separable CNN network by Keras and TensorFlow.

As a result, we got 92.33% of accuracy and 13.76% of loss. Besides, our model has 90% of recall and 76% of precision, which is perfect for this kind of disease detection problem.

In this report, we discussed the general idea about our image data set in 2.1, and data augmentation to deal with imbalanced data in 2.2; applied depthwise seperable CNN and transfer learning to build our model in section 2.3; initialized the first two layers' weight from VGG16 in 2.4; the impeletation of Depthwise Separable CNN in next layers in 2.5; the training result shows up in section 3 and conclusion are made in section 4.

1. Introduction

Pneumonia is a very common disease. It can be either: 1) Bacterial pneumonia 2) Viral Pneumonia 3) Mycoplasma pneumonia and 4) Fungal pneumonia. Our dataset consists pneumonia samples belonging to the first two classes.

Figure 1. Normal Bacterial and Viral Pneumonia

You can tell from the images above that it is hard to figure out the difference of each classes by eyes. The aim of this kernel is to develop a robust deep learning model from scratch on this limited amount of data. Imagine how efficient it would be to diagnose pneumonia only by analyzing few chest X-Ray images on computers, it would be much more faster and accuracy than viewing by a doctor. It would not only improve the efficiency of identifying pneumonia, but also save more time and effort of professionals.

2. Code with Documentation

2.1 Data Set

We have three data sets: train, test, and validation:

	Normal	Bacteria	Virus
Train	1341	2530	1345
Test	234	242	148
Validation	8	8	8

2.2 Data Augmentation

Data augmentation is a powerful technique which helps in almost every case for

improving the robustness of a model. But augmentation can be much more helpful where the dataset is imbalanced. You can generate different samples of over sampled class in order to try to balance the overall distribution.

The training dataset is highly unbalanced, which is almost 1 : 2 : 1. Therefore, we use data augmentation to generate more image for "Normal Class" and "Virus Class".

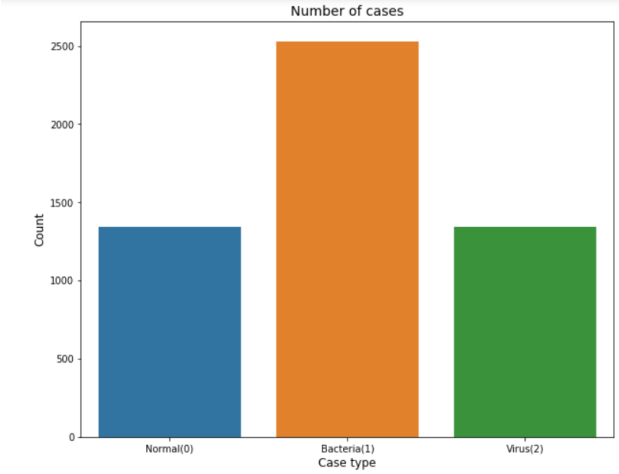

Figure 2. Image proportion in train dataset

Augmentation sequence:

seq = iaa.OneOf([

iaa.Fliplr(),

iaa.Affine(rotate=20),

iaa.Multiply((1.2, 1.5))])

First, we flipped the image, then rotate 20 degrees ,and finally, set the random brightness of new image as (1.2, 1.5).

We generated a new image followed the sequence above when we randomly selected a "normal" or "virus" image from the train data set, so finally, we have a dataset as the proportion of 2 : 1 : 2 to balance these three classes of data.

2.3 Transfer learning from ImageNet

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.

ImageNet is an image dataset organized according to the WordNet hierarchy.

We initialized the first few layers from a network that is pre-trained on ImageNet.

This is because first few layers capture general details like color blobs, patches, edges, etc. Instead of randomly initialized weights for these layers, it would be much better if you fine tune them.

Layer (type)	Output Shape	Param #
ImageInput (InputLayer)	(None, 224, 224, 3)	0
Conv1_1 (Conv2D)	(None, 224, 224, 32)	896
Conv1_2 (Conv2D)	(None, 224, 224, 32)	9248
pool1 (MaxPooling2D)	(None, 112, 112, 32)	0

2.4 VGG16

The VGG neural network is an image classification convolutional neural network. Given an image, the VGG network will output probabilities of the different classes that an image could potentially belong to.

```
# Open the VGG16 weight file
f = h5py.File('../input/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5', 'r')

# Select the layers for which you want to set weight.

w,b = f['block1_conv1']['block1_conv1_W_1:0'], f['block1_conv1']['block1_conv1_b_1:0']
model.layers[1].set_weights = [w,b]

w,b = f['block1_conv2']['block1_conv2_W_1:0'], f['block1_conv2']['block1_conv2_b_1:0']
model.layers[2].set_weights = [w,b]

w,b = f['block2_conv1']['block2_conv1_W_1:0'], f['block2_conv1']['block2_conv1_b_1:0']
model.layers[4].set_weights = [w,b]

w,b = f['block2_conv2']['block2_conv2_W_1:0'], f['block2_conv2']['block2_conv2_b_1:0']
model.layers[5].set_weights = [w,b]

f.close()
model.summary()
```

We will initialize the weights of first two convolutions with VGG16 weights, which reduce tons of time and efforts for tuning the weight by the model itself.

2.5 Depthwise Separable Convolution

We chose Depthwise Separable Convolution instead of normal Convolution because it introduce a lesser number of parameters, and as different filters are applied to each channel, it captures more information.

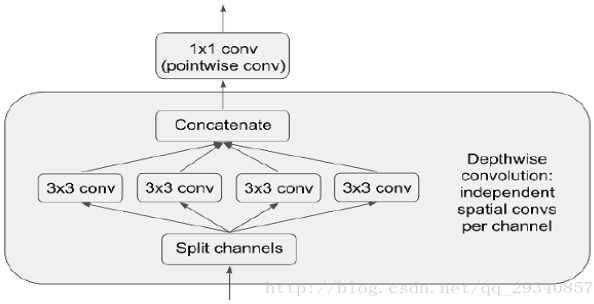

Figure 3. Depthwise Convolution

For example, a 3 * 3 convolutional layer on 16 input channels and 32 output channels, there are total

$$3 * 3 * 16 * 32 = 4608$$

parameters in a normal; however, there is only

$$16 * 3 * 3 + 16 * 32 * 3 * 3 = 656$$

parameters in Depthwise Convolution.

After the convolutional layers, we set up a Batch Normal layers to normalize the input of next layer.

For example, when we have features from 0 to 1 and some from 1 to 1000, we should normalize them to speed up learning. If the input layer is benefiting from it, why not do the same thing also for the values in the hidden layers, that are changing all the time, and get 10 times or more improvement in the training speed.

Conv2_1 (SeparableConv2D)	(None, 112, 112, 64)	2400
Conv2_2 (SeparableConv2D)	(None, 112, 112, 64)	4736
pool2 (MaxPooling2D)	(None, 56, 56, 64)	0
Conv3_1 (SeparableConv2D)	(None, 56, 56, 128)	8896
bn1 (BatchNormalization)	(None, 56, 56, 128)	512
Conv3_2 (SeparableConv2D)	(None, 56, 56, 128)	17664
bn2 (BatchNormalization)	(None, 56, 56, 128)	512
Conv3_3 (SeparableConv2D)	(None, 56, 56, 128)	17664
pool3 (MaxPooling2D)	(None, 28, 28, 128)	0
Conv4_1 (SeparableConv2D)	(None, 28, 28, 256)	34176
bn3 (BatchNormalization)	(None, 28, 28, 256)	1024
Conv4_2 (SeparableConv2D)	(None, 28, 28, 256)	68096
bn4 (BatchNormalization)	(None, 28, 28, 256)	1024
Conv4_3 (SeparableConv2D)	(None, 28, 28, 256)	68096
pool4 (MaxPooling2D)	(None, 14, 14, 256)	0

Figure 4. General picture about our CNN

2.6 Start Training and Validation

We used "Adam" as optimizer function and "categorical_crossentropy" as loss function.

```
# opt = RMSprop(lr=0.0001, decay=1e-6)
opt = Adam(lr=0.0001, decay=1e-5) #lr=Learning rate, decay: learning rate decay rate, adam learning function
es = EarlyStopping(patience=5)
chkpt = ModelCheckpoint(filepath='best_model_todate', save_best_only=True, save_weights_only=True)
model.compile(loss = 'categorical_crossentropy', metrics=['accuracy'],optimizer=opt)
```

Figure 5. Optimizer and loss functions

We set up the batch size as 20 and epochs as 10 to train the model on GPU.

```
batch_size = 20 #Bigger, more accurate
nb_epochs = 10

# Get a train data generator
train_data_gen = data_gen(data=train_data, batch_size=batch_size)

# Define the number of training steps
nb_train_steps = train_data.shape[0]//batch_size

print("Number of training and validation steps: {} and {}".format(nb_train_steps, len(valid_data)))
```

```
Number of training and validation steps: 260 and 16
```

Figure 6. Batch size and epochs

3. Result

3.1 Evaluate Convolutional Network (Loss and Accuracy)

The lower the loss, the better a model (unless the model has over-fitted to the training data). The loss is calculated on training and validation and its interperation is how well the model is doing for these two sets. Unlike accuracy, loss is not a percentage. It is a summation of the errors made for each example in training or validation sets.

In the case of neural networks, the loss is usually negative log-likelihood and residual sum of squares for classification and regression respectively. Then naturally, the main objective in a learning model is to reduce (minimize) the loss function's value with respect to the model's parameters by changing the weight vector values through different optimization methods, such as backpropagation in neural networks.

Loss value implies how well or poorly a certain model behaves after each iteration of optimization. Ideally, one would expect the reduction of loss after each, or several, iteration(s).

The accuracy of a model is usually determined after the model parameters are learned and fixed and no learning is taking place. Then the test samples are fed to the model and the number of mistakes (zero-one loss) the model makes are recorded, after comparison to the true targets. Then the percentage of misclassification is calculated.

The number of our test samples is 624, and model classifies 474 of those correctly, then the model's accuracy is 75.96%; at the same time, the model classifies 500 of those loss, then the model's loss is 80.21%.

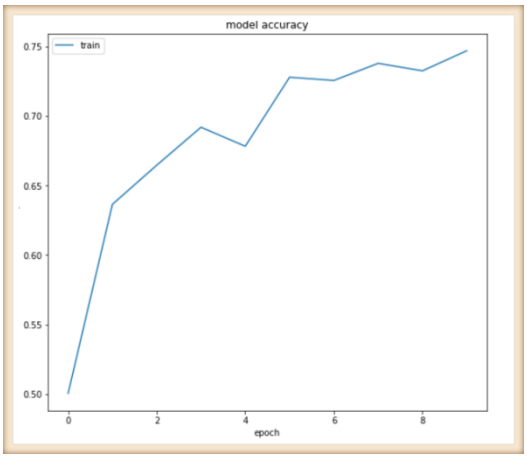

Figure 7. Training accuracy

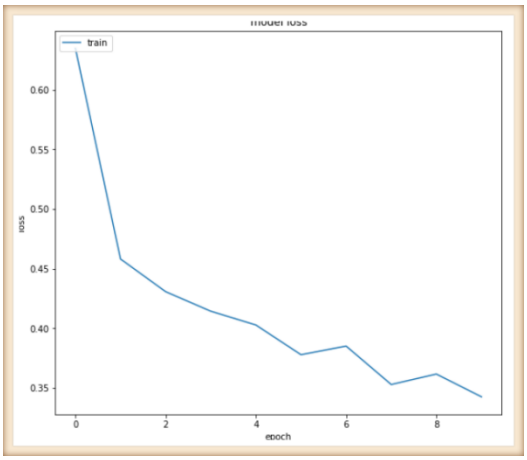

Figure 8. Training loss

There are also some subtleties while reducing the loss value. For instance, you may run into the problem of over-fitting in which the model "memorizes" the training examples and becomes kind of ineffective for the test set. Over-fitting also occurs in cases where you do not employ a regularization, you have a very complex model (the number of free parameters W is large) or the number of data points N is very low.

3.2 Prediction Accuracy of Test Dataset

We got 90.57% of loss, and 82.7% accuracy of test set.

```
624/624 [==============================] - 7s 11ms/step
Loss on test set:  0.905697194154084
Accuracy on test set:  0.8269230769230769
```

3.3 Recall and Precision form Prediction

When a particular problem includes an imbalanced dataset, then accuracy isn't a good metric to look for. For example, if your dataset contains 95 negatives and 5 positives, having a model with 95% accuracy doesn't make sense at all. The classifier might label every example as negative and still achieve 95% accuracy. Hence, we need to look for alternative metrics. Precision and Recall are really good metrics for such kind of problems.

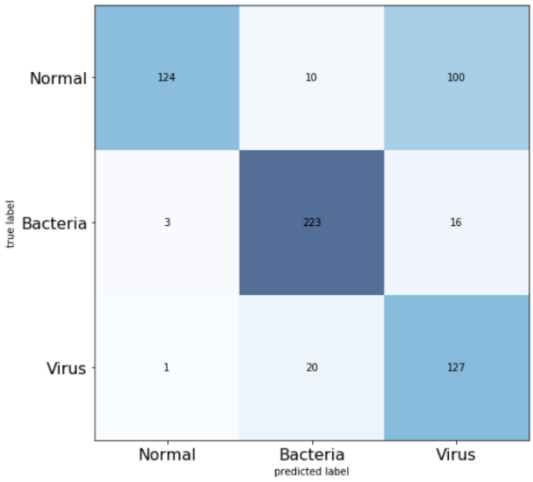

Figure 9. confusion matrix from prediction

In the field of information retrieval, precision is the fraction of retrieved documents that are relevant to the query:

$$\text{precision} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{retrieved documents}\}|}$$

$$\text{precision} = \frac{223 + 127}{223 + 127 + 16 + 20} = 0.90$$

In information retrieval, recall is the fraction of the relevant documents that are successfully retrieved.

$$\text{recall} = \frac{|\{\text{relevant documents}\} \cap \{\text{retrieved documents}\}|}{|\{\text{relevant documents}\}|}$$

$$\text{recall} = \frac{223 + 127}{223 + 127 + 10 + 100} = 0.76$$

4. Conclusion and Discussion:

Nice!!! So, our model has a 90% recall. In such problems, a good recall value is expected. But if you notice, the precision is only 76%. This is one thing to notice. Precision and Recall follows a trade-off, and you need to find a point where your recall, as well as your precision, is more than good but both can't increase simultaneously.

5. Reference:

Daniel S. Kermany , Michael Goldbaum , Wenjia Cai , Carolina C.S. Valentim , Huiying Liang , M. Anthony Lewis , Huimin Xia ,Kang Zhang, "*Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning",* VOLUME 172, ISSUE 5, P1122-1131.E9, FEBRUARY 22, 2018 http://www.cell.com/cell/fulltext/S0092-8674(18)30154-5

Eli Bendersky, "*Depthwise separable convolutions for machine learning",* https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning

Krizhevsky, Alex, Ilya Sutskever, Geoffrey E. Hinton. *"Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems."* https://www.nvidia.cn/content/tesla/pdf/machine-learning/imagenet-classification-with-deep-convolutional-nn.pdf

Wikipedia https://en.wikipedia.org/wiki/Precision_and_recall

6. GitHub Link:

https://github.com/HardyWN/INFO7390-Project/tree/master/Project

https://github.com/jiahao2/INFO-7390-Advances-in-Data-Sciences