Artificial Intelligence: Pneumonia CNN Classification

November 16, 2018

This analysis builds a Convolutional Neural Network (CNN) to classify Kaggle's Chest X-Ray Images (Pneumonia) dataset of images as Healthy or Sick (with Pneumonia). The dataset consists of 5,863 images in two categories. They are of varying dimensions and channels (BW/RGB).

The chest X-ray images (anterior-posterior) were selected from retrospective cohorts of pediatric patients of one to five years old from Guangzhou Women and Children's Medical Center, Guangzhou. All chest X-ray imaging was performed as part of patients' routine clinical care.

Table of Contents

Pre-processing

Clear memory

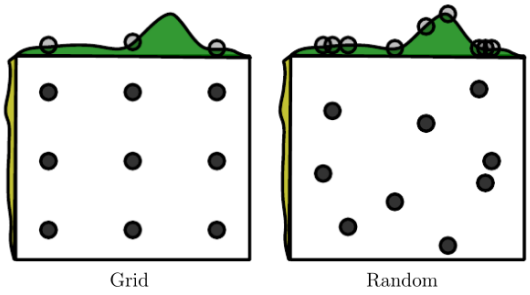
Begin by clearing the memory.

```
clear all; close all
% resets GPU memory
gpuDevice(1); % Comment out if you don't have GPU
```

Parameter Sweep

Chollet (223) encourages building simple instead of expensive models when approaching a new model. This allows you to establish a baseline model from which you can build from. According to Goodfellow, if you're progress is stopped by low training accuracy, you can grow the model by adding more layers or adding more hidden units. It's easy to increase hidden units (neurons) in fully connected layers. Otherwise, you can optimize the model by improving the learning algorithm, for example by tuning the learning rate hyperparameter.

In this section, we explore how to set our hyperparameters in order to achieve strong test accuracy. Using a grid search (outlined by Goodfellow, 420) is the common practice to follow when there are three or fewer hyperparameters to set. In this code, we can implement our own grid search (parameter sweep) using a *for loop* which uses varying step sizes. The experimental values which yield the test accuracies will be combined.



Goodfelow 420

The following loop is written to allow for varying step sizes, as opposed to a loop with a constant step rate. This proved helpful in testing what a small change in image size would have on accuracy, while still jumping up to test at larger dimension sizes. Our default, template loop is:

```
b = 1 % range starting point for b
while b < 500 % range upper limit
    if b < 50 % first range step
        b = b + 1;
    elseif b > 50 && b < 200 % second range step
        b = b + 10;
    else
        b = b + 100; % final range step
    end

    disp(b) % execute this each loop

end % place this end at end of loop, at bottom of writing results to .txt
```

Have fun testing your own variations on this loop. For a simpler, constant loop, you can use a variable like the following:

```
% from 0 to 300, step 20
b = [0:20:300]; % method 1
b = [10,50:50:300]; % method 2
```

Memory Limitations

In the initial stages of training, my image count was 200 sick, 200 normal. If minibatch = 100, then each epoch, there are 4 iterations. This can be a good starting point for your computer to begin training, before finding your memory limitations. Iterations are determined by Matlab as:

$$\text{Iterations} = \frac{\text{Total Images}}{\text{Minibatch size}}$$

Setting epochs to 5 will mean there will be 5 forward and backwards passes, where each backward pass improves weight values to minimize misclassification (loss).

However, minibatch size on this computer cannot be set too high, due to memory limitations. Though my current image set of 400 jpegs is only 8 Mbs, memory errors occur because it's too much for 16 Gb DDR RAM to handle. According to this source, this is because memory is divided by:

- Model Parameters (Weights)
- Feature Maps
- Gradient Maps
- Workspace

Image size

We can set image size to 5x5 pixels to establish a baseline performance. At such a low resolution, the model cannot learn and accuracy should be random 50 50 guess. Take note however, your image size must be as large as your smallest convolutional filter.

```
% b = 250; % range starting point for b. must be at least size of smallest filter...about 5
% while b < 400  % range upper limit
%     if b < 50 % first range step
%         b = b + 1;
%     elseif b > 50 && b < 100 % second range step
%         b = b + 5;
%     else
%         b = b + 10; % final range step
%     end
```

Batch size

Memory limitations prevent my medium-end hardware from accepting large images in large batches. From exploration, it seems image sizes of 100 x 100 in batches of 15 are acceptable. Image sizes of 500 x 500 in batch sizes of about 4 are the upper limit. Given the results from sweeping across image size and checking for accuracy, it seems images of 100 x 100 and batches of 15 are enough to learn from. When training an images or batch sizes too large for your GPU (to check for graphics card, enter gpuDevice in console), your script will halt due to a CUDA memory limit (red text).

```
% b = 1; % range starting point for b
```

```
% while b < 30  % range upper limit
%     if b < 30 % first range step
%         b = b + 1;
%     end
%
% % it otherwise causes CUDA memory crash
```

Epochs

At larger image (300 x 300) resolutions, more (50+) epochs are required to achieve learning.

```
% b = 1; % range starting point for b
% while b < 400  % range upper limit
%     if b < 10 % first range step
%         b = b + 1;
%     elseif b > 20 && b < 50 % second range step
%         b = b + 5;
%     else
%         b = b + 10; % final range step
%     end
```

Fully Connected neurons

At higher neuron counts in the fully connected layer, we should see higher training accuracy, at risk of overfitting.

The reason to increase the number of hidden units is "increasing the number of hidden units increases the representational capacity of the model," at the cost of "increas(ing) both the time and memory cost of essentially every operation on the model," (Goodfellow 419).

```
% b = 0; % range starting point for b
% while b < 200  % range upper limit
%     if b < 30 % first range step
%         b = b + 1;
%     elseif b > 30 && b < 50 % second range step
%         b = b + 5;
%     else
%         b = b + 10; % final range step
%     end
```

Hyperparameters

Set your looped hyperparameter to b and loop.

```
minibatch = 30;
imagesize = [100 100]
```

```
imagesize = 1×2
   100   100
```

```
epochs = 100;
FCneurons = 100;
```

Load train data

Load data from an explicit path.

Create an `ImageDatastore` object, by labeling each image according to it's folder name.

```
gpuDevice(1); % Comment out if you don't have GPU

% laptop
directoryTrain = 'C:\tmp\chest_xray\train';

% desktop
%directoryTrain = 'C:\tmp\chest_xray ex\chest_xray\trainMoreBalance - 400 images\';
imdsTrain = imageDatastore(fullfile(directoryTrain), 'IncludeSubfolders', true, 'FileExtensions
```

Choose what you're testing.

```
dirNormalTrain = fullfile(directoryTrain, '\NORMAL\');
dirPneuTrain = fullfile(directoryTrain, '\PNEUMONIA\');

% bacteria vs virus
% dirNormalTrain = fullfile(directoryTrain, '\bacteria\');
% dirPneuTrain = fullfile(directoryTrain, '\virus\');
```

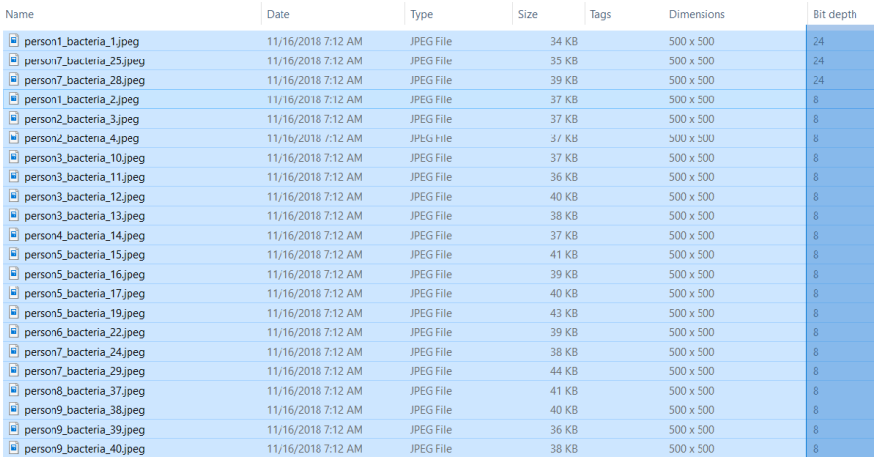
Adjust Dimensions

Check dimensions, using code found here. The original dataset had varying dimensions and aspect ratios. Convert to your preferred dimensions.

The dataset contained around 250 RGB (24 bit-depth) images. These can converted into BW (8 bit-depth) images using Matlab, a Photoshop script (easier), or simply deleted (easiest).

Augment data source for color conversion:

```
augmentedTrainingSet = augmentedImageDatastore(imageSize, imdsTrainingSet, 'ColorPreprocessing', 'gray2rgb')
augmentedTestSet = augmentedImageDatastore(imageSize, imdsTestSet, 'ColorPreprocessing', 'gray2rgb');
```



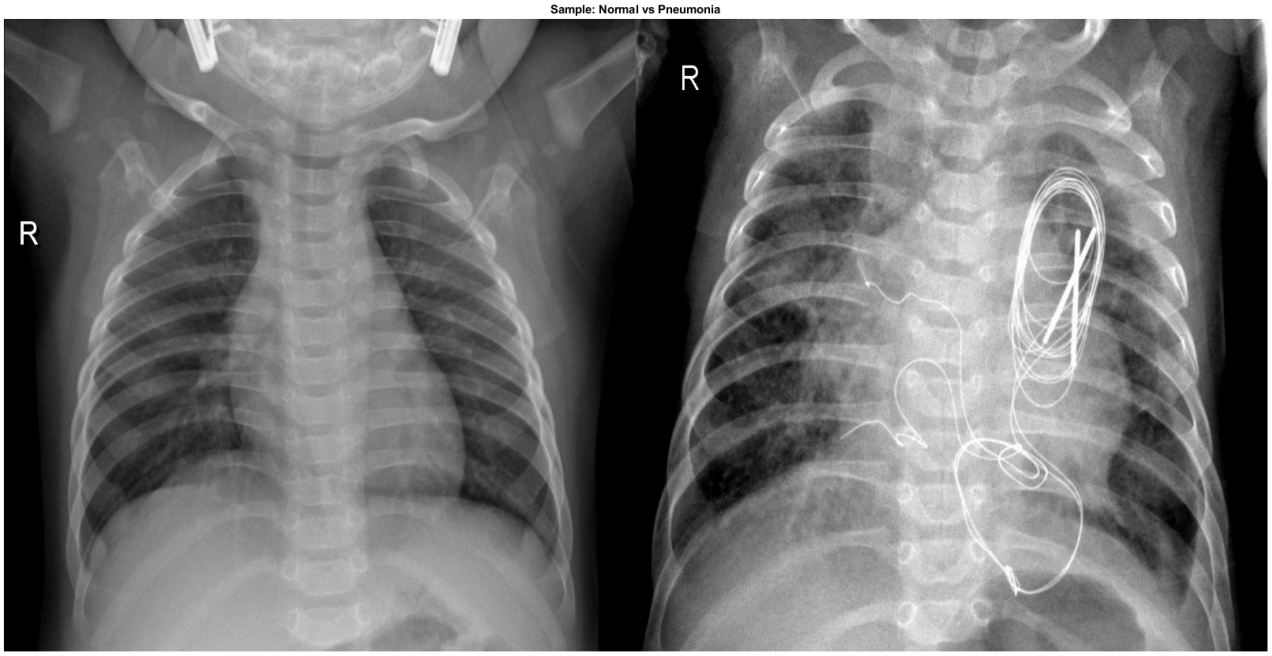
Name	Date	Type	Size	Tags	Dimensions	Bit depth
person1_bacteria_1.jpeg	11/16/2018 7:12 AM	JPEG File	34 KB		500 x 500	24
person7_bacteria_25.jpeg	11/16/2018 7:12 AM	JPEG File	35 KB		500 x 500	24
person7_bacteria_28.jpeg	11/16/2018 7:12 AM	JPEG File	39 KB		500 x 500	24
person1_bacteria_2.jpeg	11/16/2018 7:12 AM	JPEG File	37 KB		500 x 500	8
person2_bacteria_3.jpeg	11/16/2018 7:12 AM	JPEG File	37 KB		500 x 500	8
person2_bacteria_4.jpeg	11/16/2018 7:12 AM	JPEG File	37 KB		500 x 500	8
person3_bacteria_10.jpeg	11/16/2018 7:12 AM	JPEG File	37 KB		500 x 500	8
person3_bacteria_11.jpeg	11/16/2018 7:12 AM	JPEG File	36 KB		500 x 500	8
person3_bacteria_12.jpeg	11/16/2018 7:12 AM	JPEG File	40 KB		500 x 500	8
person3_bacteria_13.jpeg	11/16/2018 7:12 AM	JPEG File	38 KB		500 x 500	8
person4_bacteria_14.jpeg	11/16/2018 7:12 AM	JPEG File	37 KB		500 x 500	8
person5_bacteria_15.jpeg	11/16/2018 7:12 AM	JPEG File	41 KB		500 x 500	8
person5_bacteria_16.jpeg	11/16/2018 7:12 AM	JPEG File	39 KB		500 x 500	8
person5_bacteria_17.jpeg	11/16/2018 7:12 AM	JPEG File	40 KB		500 x 500	8
person5_bacteria_19.jpeg	11/16/2018 7:12 AM	JPEG File	43 KB		500 x 500	8
person6_bacteria_22.jpeg	11/16/2018 7:12 AM	JPEG File	39 KB		500 x 500	8
person7_bacteria_24.jpeg	11/16/2018 7:12 AM	JPEG File	38 KB		500 x 500	8
person7_bacteria_29.jpeg	11/16/2018 7:12 AM	JPEG File	44 KB		500 x 500	8
person8_bacteria_37.jpeg	11/16/2018 7:12 AM	JPEG File	41 KB		500 x 500	8
person9_bacteria_38.jpeg	11/16/2018 7:12 AM	JPEG File	40 KB		500 x 500	8
person9_bacteria_39.jpeg	11/16/2018 7:12 AM	JPEG File	36 KB		500 x 500	8
person9_bacteria_40.jpeg	11/16/2018 7:12 AM	JPEG File	38 KB		500 x 500	8

```
imdsTrain.ReadFcn = @(loc)imresize(imread(loc), imagesize); % resize
imagesTrain = read(imdsTrain); % check image size
```

Display Sample Images

Display a random sample x-ray image of a normal, healthy patient.

```
% Normal
dnorm = dir([dirNormalTrain '\*.jpeg']);
fnorm = {dnorm.name};
nnorm = numel(fnorm);
nidx = randi(numel(fnorm));
nfile = fnorm{nidx};
BWHealthy = imread(fullfile(dirNormalTrain, nfile));

% Pneumonia
dpneu = dir([dirPneuTrain '\*.jpeg']);
fpneu = {dpneu.name};
npneu = numel(fpneu);
pidx = randi(numel(fpneu));
pfile = fpneu{pidx};
BWPneu = imread(fullfile(dirPneuTrain, pfile));
title('Example: Pneumonia')

% imshowpair(dirPneum,BWHealthy)
imshowpair(BWHealthy, BWPneu, 'montage')
title('Sample: Normal vs Pneumonia')
```



Do you see any discernable difference between the heathy and pneumnia x-rays?

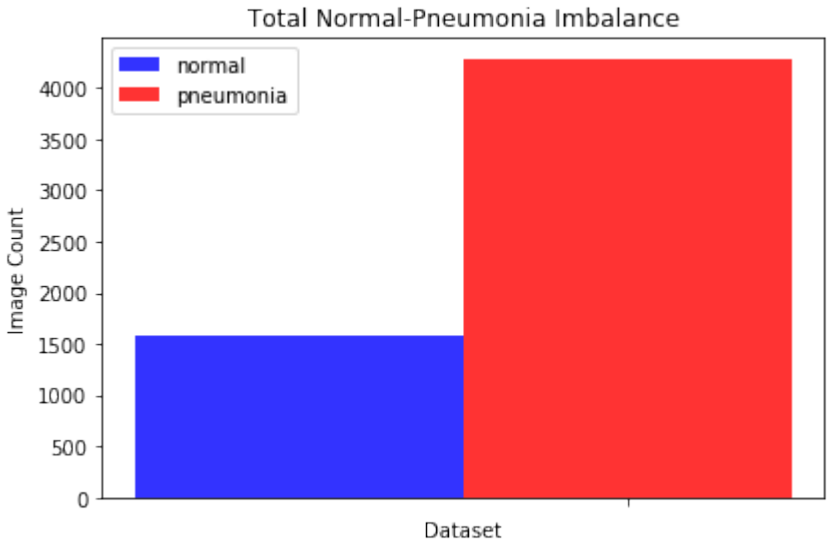
Preprocessing

This code is used later for visualizing the activations.

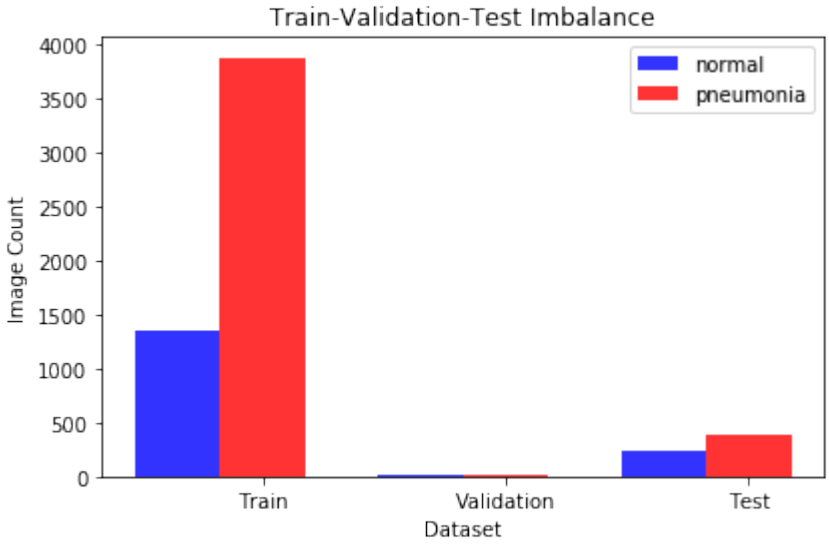
```
N = numel(imdsTrain.Files);
for i = 1:N
   fn = imdsTrain.Files{i};
   im = readimage(imdsTrain, i); % This is for activation visualization
end
```

Train-Test Split

The dataset comes with an imbalance between healthy and pneumonia patients.



It also has an imbalance between train test and validation.



Therefore, we merged all train test and validation images into a train folder in Windows, and let Matlab 70-30 split (example).

Create two new datastores from the images in `imds`, by randomly drawing from each label. Random does not randomize the samples taken from each set, but takes proportionally from the two labels, randomizing their content.

```
% Count number of images per label
labelCount = countEachLabel(imdsTrain);

% Create training and validation sets
[imdsTrainingSet, imdsTestSet] = splitEachLabel(imdsTrain, 0.7, 'randomize');
```

Modeling

Choose to either load or train a model.

A. Load Model

Load the previously trained network (which is saved after training).

```
% load GarthNet.mat
% net = GarthNet
```

B. Train Model

Possible network sweep determination

https://blogs.mathworks.com/loren/2015/08/04/artificial-neural-networks-for-beginners/

Of note, the network must be trained before visualizations can be done. If you're using a pretrained network though, you can skip below to Import Model and immediately start visualizing the CNN.

Here, a standard CNN model is used. According to Geron (p. 371), **typical CNN** architectures stack a few convolutional layers (each one generally followed by a ReLU layer), then a pooling layer, then another few convolutional layers (+ReLU), then another pooling layer, and so on. The image gets smaller and smaller as it progresses through the network, but it also typically gets deeper and deeper (i.e., with more feature maps) thanks to the convolutional layers (see Figure 13-9). At the top of the stack, a regular feedforward neural network is added, composed of a few fully connected layers (+ReLUs), and the final layer outputs the prediction (e.g., a softmax layer that outputs estimated class probabilities). *A common mistake is to use convolution kernels that are too large. You can often get the same effect as a 9 × 9 kernel by stacking two 3 × 3 kernels on top of each other, for a lot less processing power.*

This CNN also uses regularization in the form of batch normalization and dropout.

According to Geron (p. 364?), a common practice is to size them to form a **funnel**, with fewer and fewer neurons at each layer — the rationale being that many low-level features can coalesce into far fewer high-level features. For example, a typical neural network for MNIST may have two hidden layers, the first with 300 neurons and the second with 100. However, *this practice is not as common now, and you may simply use the same size for all hidden layers* — for example, all hidden layers with 150 neurons: that's just one hyperparameter to tune instead of one per layer. Just like for the number of layers, you can try increasing the number of neurons gradually until the network starts overfitting.

According to Geron (p. 284), **batch normalization** was proposed "*to address the vanishing/exploding gradients problems, and more generally the problem that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change... The technique consists of adding an operation in the model just before the activation function of each layer, simply zero-centering and normalizing the inputs, then scaling and shifting the result using two new parameters per layer."* It remains an ongoing debate whether batch normalization should be before or after activation. Based on this test, I would place batch normalization after.

According to Geron (p. 309), **dropout** is one of the most popular regularization techniques for deep neural networks. The team that proposed it (G. E. Hinton and Nitish Srivastava et al.) found that even the state-of-the-art neural networks got a 1–2% accuracy boost simply by adding dropout. Geron wxplains that "*neurons trained with dropout cannot co-adapt with their neighboring neurons; they have to be as useful as possible on their own. They also cannot rely excessively on just a few input neurons; they must pay attention to each of their input neurons. They end up being less sensitive to slight changes in the inputs. In the end you get a more robust network that generalizes better.*" Goodfellow (419) meanwhile confirms that "dropping units less often gives the units more opportunities to "conspire" with each other to fit the training set."

According to Geron (p. 364?), *a [simple] approach is to pick a model with more layers and neurons than you actually need, then use early stopping to prevent it from overfitting (and other regularization techniques, especially dropout... This has been dubbed the "stretch pants" approach: instead of wasting time looking for pants that perfectly match your size, just use large stretch pants that will shrink down to the right size.*

Specify Training Options

Define Model Architecture

Create simple CNN network.

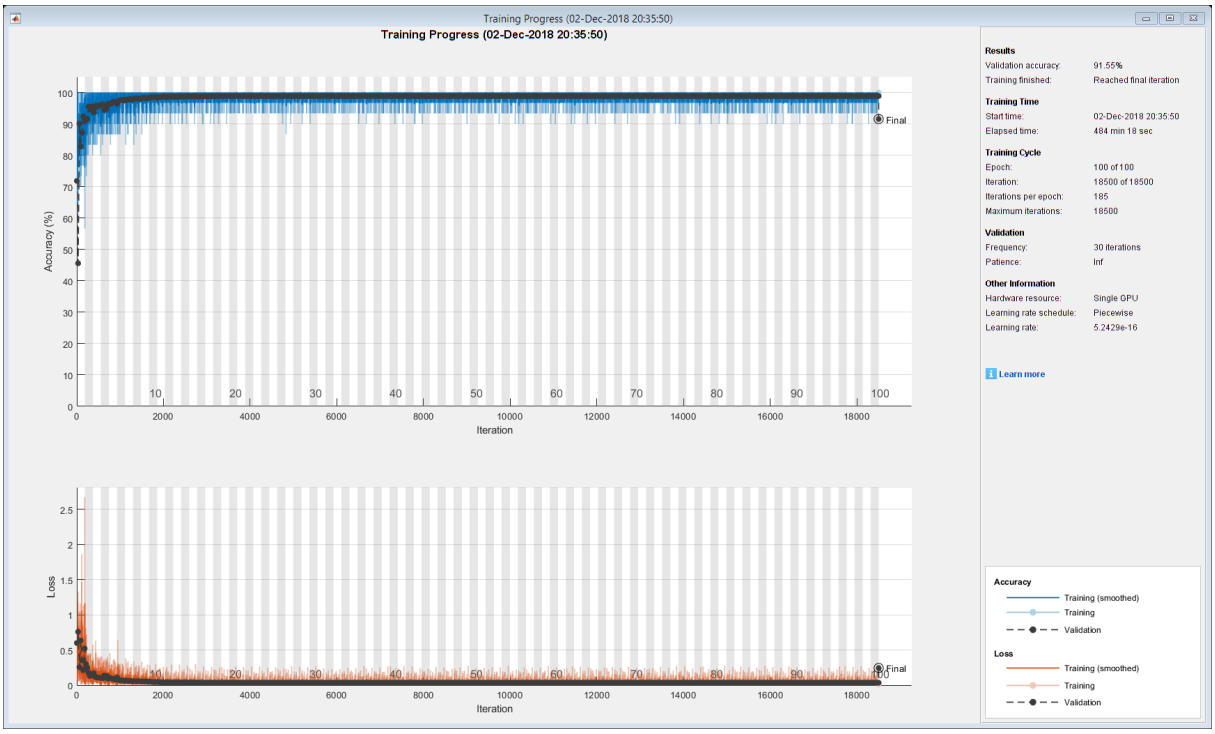
```
% Hyperparameters defined in looping segments at beginning.

% Specify the convolutional neural network architecture.
layers = [
    imageInputLayer(imagesize)
    % small filter size of 3 makes for uninteresting first layer activation and feature images
    convolution2dLayer(3, 8, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    dropoutLayer % Having more than one dropout layer seems to crash the net.
    maxPooling2dLayer(2, 'Stride', 2)
    convolution2dLayer(3, 16, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    maxPooling2dLayer(2, 'Stride', 2)
    convolution2dLayer(3, 49, 'Padding', 'same')
    batchNormalizationLayer
    reluLayer
    fullyConnectedLayer(FCneurons)
    fullyConnectedLayer(2)
    softmaxLayer
    classificationLayer];
```

Define Hyperparameters

```
% Set options
options = trainingOptions('sgdm', ...
    'LearnRateSchedule','piecewise', ... % new
    'LearnRateDropFactor',0.2, ... % new
    'LearnRateDropPeriod',5, ... % new
    'MaxEpochs', epochs, ...
    'ValidationData', imdsTrain, ...
    'ValidationFrequency', 30, ...
    'Verbose', false, ...
    'MiniBatchSize', minibatch, ...
    'Plots', 'training-progress');
```

Train Network

```
tic
% Train the network.
net = trainNetwork(imdsTrain, layers, options);
```




```
% Save model
% GarthNet = net;
% save GarthNet
toc
```

```
Elapsed time is 29183.006711 seconds.
```

Display Model

Display the network's architecture.

```
net.Layers
```

```
ans =
  17x1 Layer array with layers:

     1   'imageinput'    Image Input             100x100x1 images with 'zerocenter' normalization
     2   'conv_1'        Convolution             8 3x3x1 convolutions with stride [1  1] and padding 'same'
     3   'batchnorm_1'   Batch Normalization     Batch normalization with 8 channels
     4   'relu_1'        ReLU                    ReLU
     5   'dropout'       Dropout                 50% dropout
     6   'maxpool_1'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
     7   'conv_2'        Convolution             16 3x3x8 convolutions with stride [1  1] and padding 'same'
     8   'batchnorm_2'   Batch Normalization     Batch normalization with 16 channels
     9   'relu_2'        ReLU                    ReLU
    10   'maxpool_2'     Max Pooling             2x2 max pooling with stride [2  2] and padding [0  0  0  0]
    11   'conv_3'        Convolution             49 3x3x16 convolutions with stride [1  1] and padding 'same'
    12   'batchnorm_3'   Batch Normalization     Batch normalization with 49 channels
    13   'relu_3'        ReLU                    ReLU
    14   'fc_1'          Fully Connected         100 fully connected layer
    15   'fc_2'          Fully Connected         2 fully connected layer
    16   'softmax'       Softmax                 softmax
    17   'classoutput'   Classification Output   crossentropyex with classes 'NORMAL' and 'PNEUMONIA'
```

```
analyzeNetwork(net)
```

Having completed classification, we now turn to visualizing what the CNN learned.

Evaluate Performance

Here is a good post on displaying many performance plots.

```
% Report accuracy of baseline classifier on validation set
% OR PREDICT()?
[label score] = classify(net, imdsTestSet);
YValidation = imdsTestSet.Labels;
imdsAccuracy = sum(label == YValidation) / numel(YValidation);
```

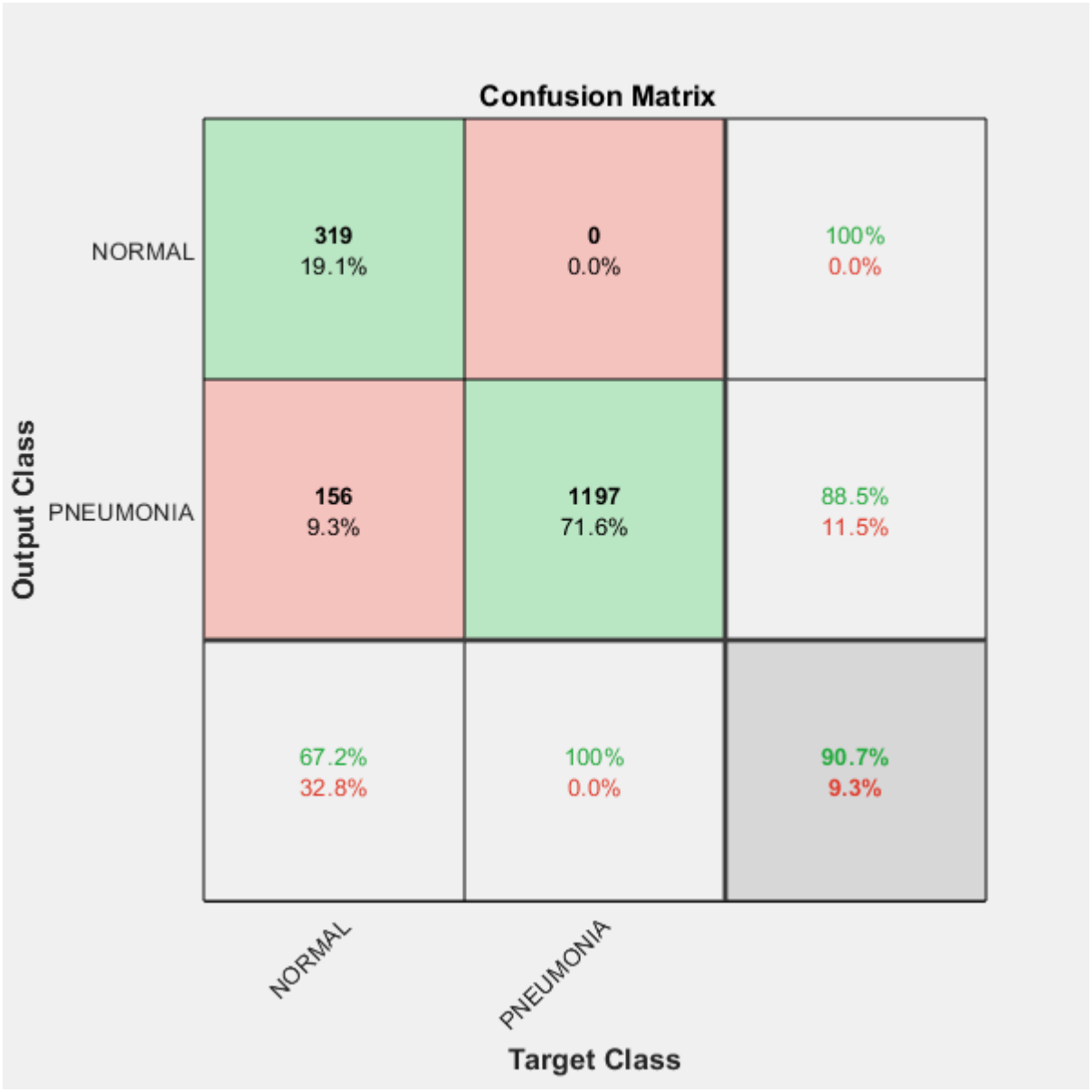
*On the confusion matrix plot, the rows correspond to the predicted class **(Output Class)**, and the columns show the true class **(Target Class)**.*

*The **column on the far right** of the plot shows the percentages of all the examples predicted to belong to each class that are correctly and incorrectly classified. These metrics are often called the **precision** (or positive predictive value) and false discovery rate, respectively. The **row at the bottom** of the plot shows the percentages of all the examples belonging to each class that are correctly and incorrectly classified. These metrics are often called the **recall** (or true positive rate) and false negative rate, respectively. The cell in the bottom right of the plot shows the overall accuracy.* Source

Calculate results.

Confusion matrix

```
a = plotconfusion(YValidation, label)
```




```
a =
  Figure (PLOTCONFUSION1) with properties:

      Number: 1
        Name: 'Confusion (plotconfusion)'
       Color: [0.9400 0.9400 0.9400]
    Position: [660 300 600 600]
       Units: 'pixels'

  Show all properties
```

```
C0 = confusionmat(label, YValidation);
TP0 = C0(1,1);
TN0 = C0(2,2);
FP0 = C0(1,2);
FN0 = C0(2,1);

precision0 = TP0/(TP0 + FP0);
recall0 = TP0/(TP0 + FN0);
```

```
fscore0 = 2*((precision0 * recall0)/(precision0 + recall0));
```

Write results to timestamped file, for tracking historical performance.

```
Filename = sprintf('output_loop_02_%s.txt', datestr(now,'yyyy.mm.dd.HH.MM.SS'));
fid = fopen(Filename, 'w');

fprintf(fid, ['N, ', num2str(nnorm + npneu)]);
fprintf(fid, ['\nNormal ratio, ', num2str(nnorm/(nnorm + npneu))]);
fprintf(fid, ['\nSick ratio, ', num2str(npneu/(nnorm + npneu))]);

fprintf(fid, ['\n\nNormal count, ', num2str(nnorm)]);
fprintf(fid, ['\nPneumonia count, ', num2str(npneu)]);

fprintf(fid, ['\n\nimagesize, ', num2str(imagesize)]);

fprintf(fid, ['\n\nepochs, ', num2str(epochs)]);
fprintf(fid, ['\nminibatch, ', num2str(minibatch)]);

fprintf(fid, ['\n\naccuracy, ', num2str(imdsAccuracy)]);
fprintf(fid, ['\nprecision, ', num2str(precision0)]);
fprintf(fid, ['\nrecall, ', num2str(recall0)]);
fprintf(fid, ['\nfscore, ', num2str(fscore0)]);

fprintf(fid, ['\n\nTP0, ', num2str(TP0)]);
fprintf(fid, ['\nTN0, ', num2str(TN0)]);
fprintf(fid, ['\nFP0, ', num2str(FP0)]);
fprintf(fid, ['\nFN0, ', num2str(FN0)]);

fprintf(fid, ['\n\nFCneurons, ', num2str(FCneurons)]);

fclose(fid)

% end
```

Predict

Predict labels for the heldout data in `Xtest` and compare them to the actual labels in `Ytest`. This metric gives the most realistic performance guage against unseen data, and is how Kaggle scores submissions (Shure).

```
% net = trainNetwork(imdsTrainingSet, layers, options);
```

```
ans = 0
ans = 0
```

https://www.mathworks.com/help/deeplearning/ref/plotconfusion.html;jsessionid=a88c8ff750e0940af3ffcf0354f6

ROC NOT WORKING

```
% ROC[x, y] = perfcurve(imdsValidationSet.Labels, score(:, 1), 'NORMAL');
[x, y] = perfcurve(imdsTestSet.Labels, score(:, 1), 'PNEUMONIA'); % is this correct?
plot(x,y)
```

For an imbalanced dataset, we should expect to see higher accuracy. That is, if there is 1,000,000 healthy patients and 1 sick, then you should expect a very high prediction accuracy for healthy patients, but low for sick. Therefori, it is more difficult to achieve high accuracy on a balanced dataset. Then, should the model be trained on a balanced dataset to determine it's predictive power?

However, in the real world, a patient may be administered an xray to check for pneumonia because they are sick and told the doctor they cough. As a result, this dataset could reflect sick patients visiting the doctor, and be a realistic balance of healthy and sick patients.

Visualize Learning

A common critism of neural networks is that [they] are not interpretable. Partly in response, research has been done to shed light on the learning process. According to Chollet (160) and Karpathy, two ways to accomplish this are:

1. Visualize intermediate convnet outputs (layer activations) - show the network's activations during forward pass.
2. Visualize convnets filters (features) - *See precisely what visual pattern each filter is receptive to. That is, visualize the weights.*

Activations

To visualize the activations, we use this example, which requires Matlab's Deep Learning Toolbox and Image Processing Toolbox.

According to Chollet, "Visualizing intermediate activations consists of displaying the feature maps that are output by various convolution and pooling layers in a network, given a certain input (the output of a layer is often called its **activation**, the output of the activation function). This gives a view into how an input is decomposed into the different filters learned by the network." The most straight-forward visualization technique is to show the activations of the network during the forward pass (Andrej Karpathy).

We display the activations of different convolutional layers. This will allow us to discover which features the network learns by comparing areas of activation with the original image. We feed an image to the model, which returns layer activation values. Investigate features by observing which areas activate and compare to the original image.

"One dangerous pitfall that can be easily noticed with this visualization is that some activation maps may be all zero for many different inputs, which can indicate dead filters, and can be a symptom of high learning rates" (source). In this code, all activations are scaled (normalized) between 0 and 1.

```
act1 = activations(net,im,'conv_1');
sz = size(act1);
act1 = reshape(act1,[sz(1) sz(2) 1 sz(3)]);

% Each activation can take any value, so normalize the output using mat2gray
% check conv1 layer's channels in net.Layers. 8 channels
I = imtile(mat2gray(act1),'GridSize',[2 4]);
imshow(I)
```

```
title('Activation for Layer: conv 1')
```

Each image is a channel from the convolutional layer.

Image + filter => activation

Good! We have visualized the first set of activations. Let's try the next two convolutional layer activations.

```
act2 = activations(net,im,'conv_2');
sz = size(act2);
act2 = reshape(act2,[sz(1) sz(2) 1 sz(3)]);

% check conv2 layer's channels in net.Layers. 16 channels
I = imtile(imresize(mat2gray(act2),[48 48]));
imshow(I)
title('Activation for Layer: conv 2')
```

```
act3 = activations(net,im,'conv_3');
sz = size(act3);
act3 = reshape(act3,[sz(1) sz(2) 1 sz(3)]);

% check conv3 layer's channels in net.Layers. 49 channels
I = imtile(imresize(mat2gray(act3),[49 49]));
imshow(I)
title('Activation for Layer: conv 3')
```

Each of these images is the convnet channel's output. White pixels are strong positive activations, while black pixels are strong negative activations. Mostly gray channels are essentially "dead filters", which, according to Andrej Karpathy, are a symptom of high learning rates.

Identifying features in this way can helps illustrate what this network has learned.

Feature Maps

To visualize what the CNN learns, use this example. Additional instructions are here.

A another common strategy to understand how CNNs learn is to visualize the weights (Andrej Karpathy). These are usually most interpretable on the first convnet layer, which looks directly at the raw pixel data. CNNs classify images using weights or *features*, which are learned during training. Here, we use `deepDreamImage see these features.` The *convolutional* layers create the output of channels, corresponding to a filter.

For an interesting look into the early, relevent neurological research which inspired CNNs, check out Fukushima's 1980 Neocognitron, which was inpired by 1958/1959 experiments by Hubel and Wiesel on primates' visual cortex.

Visualize the first **8** features learned by this layer using deepDreamImage by setting channels to be the vector of indices **1:8.** Set 'PyramidLevels' to 1 so that the images are not scaled. We can see from the Analysis of the Network using app Deep Network Designer, that the 8 channels come from the learnables for each convolutional layer.

ANALYSIS RESULT

	NAME	TYPE	ACTIVATIONS	LEARNABLES	TOTAL LEARNABLES
1	imageinput 500x500x1 images with 'zerocenter' normalization	Image Input	500×500×1		0
2	conv_1 8 3x3x1 convolutions with stride [1 1] and padding 'same'	Convolution	500×500×8	Weights 3×3×1×8 Bias 1×1×8	80
3	batchnorm_1 Batch normalization with 8 channels	Batch Normalization	500×500×8	Offset 1×1×8 Scale 1×1×8	16
4	relu_1 ReLU	ReLU	500×500×8	-	0
5	maxpool_1 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	250×250×8	-	0
6	conv_2 16 3x3x8 convolutions with stride [1 1] and padding 'same'	Convolution	250×250×16	Weights 3×3×8×16 Bias 1×1×16	1168
7	batchnorm_2 Batch normalization with 16 channels	Batch Normalization	250×250×16	Offset 1×1×16 Scale 1×1×16	32
8	relu_2 ReLU	ReLU	250×250×16	-	0
9	maxpool_2 2x2 max pooling with stride [2 2] and padding [0 0 0 0]	Max Pooling	125×125×16	-	0
10	conv_3 32 3x3x16 convolutions with stride [1 1] and padding 'same'	Convolution	125×125×32	Weights 3×3×16×32 Bias 1×1×32	4640
11	batchnorm_3 Batch normalization with 32 channels	Batch Normalization	125×125×32	Offset 1×1×32 Scale 1×1×32	64
12	relu_3 ReLU	ReLU	125×125×32	-	0
13	fc 2 fully connected layer	Fully Connected	1×1×2	Weights 2×500000 Bias 2×1	1000002
14	softmax softmax	Softmax	1×1×2	-	0
15	classoutput crossentropyex	Classification Output	-	-	0

As explained here, the convolutional layers towards the beginning of the network have a small receptive field size and learn small, low-level features.

```
layer = 2; % this is the conv layer index
name = net.Layers(layer).Name

channels = 1:8; % this is conv layer # of filters
% 'PyramidLevels' 1 = images are not scaled.
I = deepDreamImage(net,layer,channels,'PyramidLevels',1);

figure
I = imtile(I,'ThumbnailSize',[64 64]);
imshow(I)
title(['Layer ',name,'Feature Maps'])
```

Dense layers learn global patterns, while convolutional layers (displayed above) learn local patterns:

Local patterns (Chollet 122)

These are mainly edge detectors. These can be combined and constructed into more complex features, further in the network. According toi Chollet, at [this] stage, the activations retain almost all of the information present in the initial picture.

Do the filters appear noisy? Noisy patterns can be an indicator of a network that hasn't been trained for long enough, or possibly a very low regularization strength that may have led to overfitting (Andrej Karpathy).

Now the next conv layer. The layers towards the end of the network have larger receptive field sizes and learn larger features.

```
layer = 7; % this is the conv layer index
channels = 1:16; % this is conv layer # of filters
I = deepDreamImage(net,layer,channels,'PyramidLevels',1);
```

```
figure
I = imtile(I,'ThumbnailSize',[64 64]);
imshow(I)
name = net.Layers(layer).Name;
title(['Layer ',name,'Features Maps'])
```

Now the next conv layer. We can see that these learned features become much mopre elaborate. Fascinating!

```
layer = 11; % this is the conv layer index
channels = 1:32; % this is conv layer # of filters
I = deepDreamImage(net,layer,channels,'PyramidLevels',1);

figure
I = imtile(I,'ThumbnailSize',[64 64]);
imshow(I)
name = net.Layers(layer).Name;
title(['Layer ',name,'Features Maps'])
```

Strongest Activation Channel (Max Activation)

discussion - http://cs231n.github.io/understanding-cnn/ - Retrieving images that maximally activate a neuron.

Having explored all the activation channels, we can now find the strongest of them.

Each tile in the grid of activations is the output of a channel in the `conv1` layer. White pixels represent strong positive activations and black pixels represent strong negative activations. A channel that is mostly gray does not activate as strongly on the input image. The position of a pixel in the activation of a channel corresponds to the same position in the original image. A white pixel at some location in a channel indicates that the channel is strongly activated at that position.

Resize the activations in channel 32 to have the same size as the original image and display the activations.

```
act1ch32 = act1(:,:,:,8);
act1ch32 = mat2gray(act1ch32);
act1ch32 = imresize(act1ch32,imagesize);

I = imtile({im,act1ch32});
imshow(I)
```

Find the Strongest Activation Channel

You also can try to find interesting channels by programmatically investigating channels with large activations. Find the channel with the largest activation using the `max` function, resize, and show the activations.

```
[maxValue,maxValueIndex] = max(max(max(act1)));
act1chMax = act1(:, :, :, maxValueIndex);
act1chMax = mat2gray(act1chMax);
act1chMax = imresize(act1chMax, imagesize);

I = imtile({im, act1chMax});
```

```
imshow(I)
```

Compare to the original image and notice that this channel activates on edges. It activates positively on light left/dark right edges, and negatively on dark left/light right edges.

Investigate a Deeper Layer

Most convolutional neural networks learn to detect features like color and edges in their first convolutional layer. In deeper convolutional layers, the network learns to detect more complicated features. Later layers build up their features by combining features of earlier layers (Mathworks). We now rexamine the 5th convolutional activations.

```
act3 = activations(net,im,'conv_3');
sz = size(act3);
act3 = reshape(act3,[sz(1) sz(2) 1 sz(3)]);

I = imtile(imresize(mat2gray(act3),[98 98]));
imshow(I)
```

There are too many images to investigate in detail, so focus on some of the more interesting ones. Display the strongest activation in the `conv5` layer.

```
[maxValue3,maxValueIndex3] = max(max(max(act3)));
act3chMax = act3(:,:,:,maxValueIndex3);
imshow(imresize(mat2gray(act3chMax),imagesize))
```

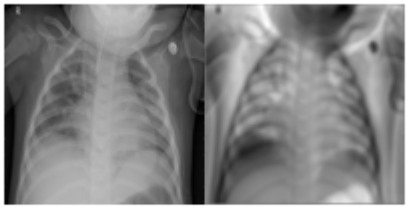
In this case, the maximum activation channel is not as interesting for detailed features as some others, and shows strong negative (dark) as well as positive (light) activation. This channel is possibly focusing on faces.

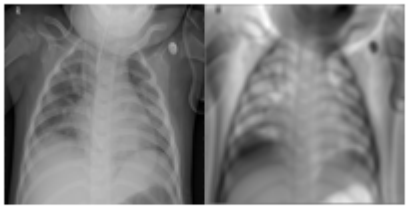
In the grid of all channels, there are channels that might be activating on eyes. Investigate channels 3 and 5 further.

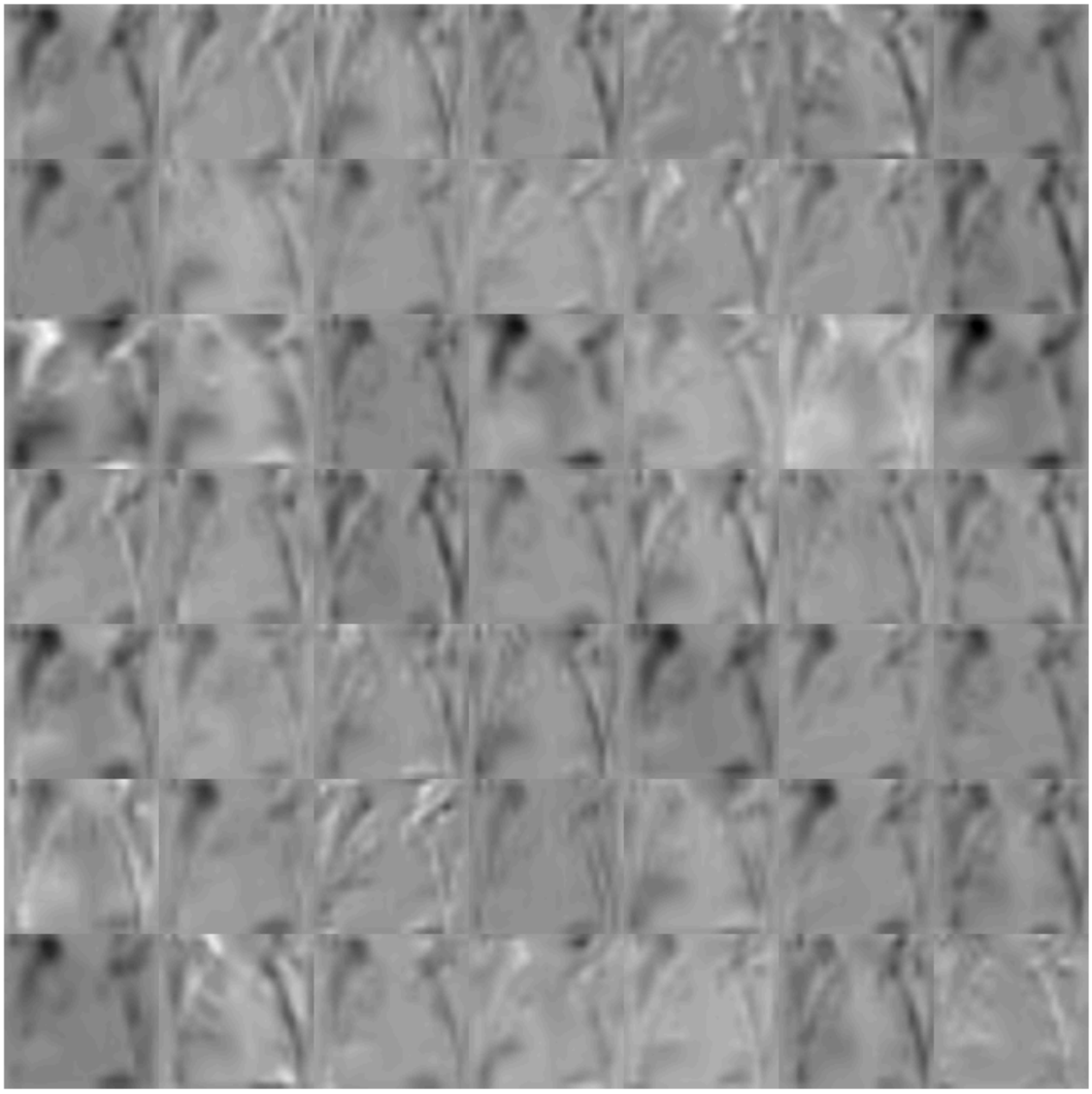
```
I = imtile(imresize(mat2gray(act3(:,:,:,[15 25])),imagesize));
imshow(I)
```

Many of the channels contain areas of activation that are both light and dark. These are positive and negative activations, respectively. However, only the positive activations are used because of the rectified linear unit (ReLU) that follows the `conv5` layer. To investigate only positive activations, repeat the analysis to visualize the activations of the `relu5` layer.

```
act3relu = activations(net,im,'relu_3');
sz = size(act3relu);
act3relu = reshape(act3relu,[sz(1) sz(2) 1 sz(3)]);

I = imtile(imresize(mat2gray(act3relu(:,:,:,[15 25])),imagesize));
imshow(I)
```







Compared to the activations of the `conv5` layer, the activations of the `relu5` layer clearly pinpoint areas of the image that have strong features.

Through this work, we will not have only classified the x-ray images, but also opened the black box to understand its operatations and determination of important predictors.

Appendix

Practical Methodology

Goodfellow discusses how to improve model performance (p. 414).

They say that once your end-to-end system is complete, then measure the algorithm's performance and see if you can improve it.

Is training set performance acceptable?

A) **No**. The algo isn't effectively learning from your existing data, making additional data useless.

My problem is that learning keeps flatlining.

1) Grow the model by adding more layers or adding more hidden units.

2) Optimize the model by improving the learning algorithm, for example by tuning the learning rate hyperparameter.

3) If growing and optimizing the models don't work well, it might be due to poor data quality. It could be too noisy or not include the right inputs. Restart with cleaner data or better features.

B) **Yes**. **Is test set performance good?**

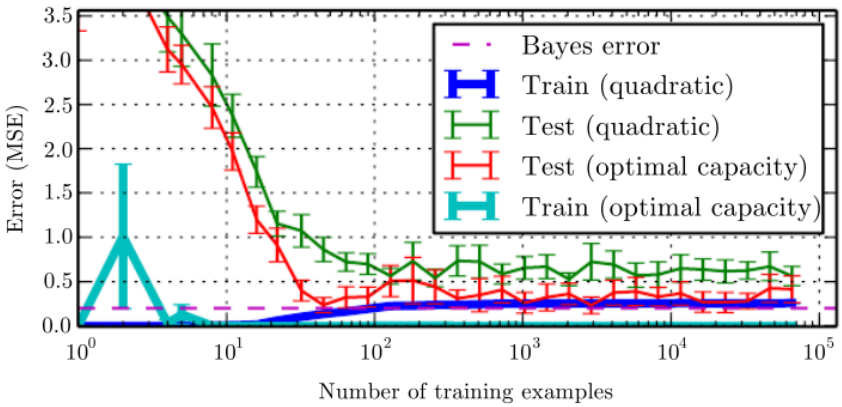
1) **Yes**. You're done!

2) **No**. Gathering more data is one of the most effective solutions. In medical applications, it may be costly or infeasible to gather more data.

i) Reduce model size

ii) Increase regularization, by adjusting hyperparameters such as weight decay coefficients, or by adding regularization strategies such as dropout.

iii) If train and test performance gap is still high, then gather more data. When determining how much data you need, plot the train and test size and generalization error (see below). By extrapolating such curves, you can predict how much additional training data would be needed to achieve a certain level of performance.

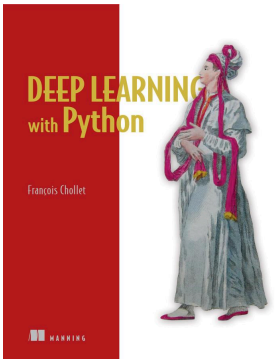


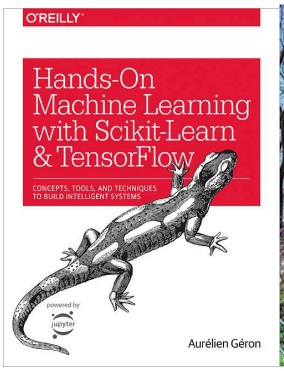
If we examine our error (for classification, loss instead of MSE), we see that our level of performance could improve. Since our data is already plentiful, we should instead focus on optimization and selecting hyperparameters.

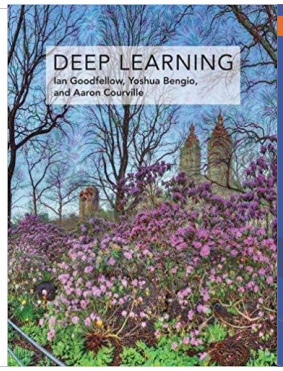
Deep Learning (TL;DR)

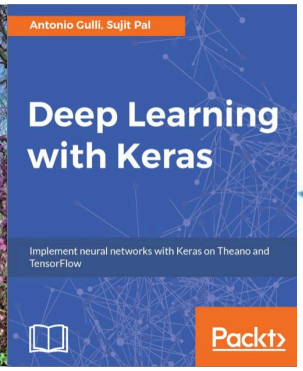
>> In deep learning, everything is a vector: everything is a point in a geometric space.Model inputs (text, images, and so on) and targets are first vectorized: turned into an initial input vector space and target vector space. Each layer in a deep-learning model operates one simple geometric transformation on the data that goes through it. Together, the chain of layers in the model forms one complex geometric transformation, broken down into a series of simple ones. This complex transformation attempts to map the input space to the target space, one point at a time. This transformation is parameterized by the weights of the layers, which are iteratively updated based on how well the model is currently performing. A key characteristic of this geometric transformation is that it must be differentiable, which is required in order for us to be able to learn its parameters via gradient descent. Intuitively, this means the geometric morphing from inputs to outputs must be smooth and continuous—a significant constraint. (Chollet, 316)

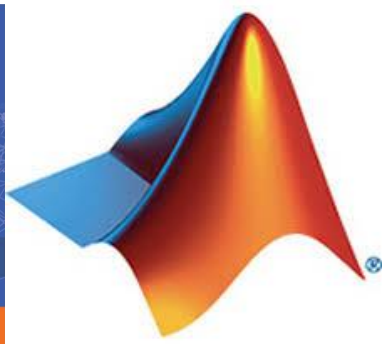
Resources











- Deep Learning with Python, François Chollet
- Hands On Machine Learning with Scikit Learn, Aurélien Géron
- Deep Learning, Ian Goodfellow, Aaron Courville, Yoshua Bengio
- Deep Learning with Keras, Antonio Gulli and Sujit Pal
- Mathworks Forums and Examples
- CS231n Convolutional Neural Networks for Visual Recognition, Andrej Karpathy