Machine Learning Engineer Nanodegree

Capstone Project
Xun Ding
July 28, 2018

I. Definition

Project Overview

Pneumonia is one of the leading infectious cause of death among children under five, killing approximately 2,400 children a day. Pneumonia accounted for approximately 16 percent of the 5.6 million children under the age of five in 2016. Most of its victims were less than 2 years old.[1] Different types of pneumonia require different care and support. Two main types of pneumonia are bacterial and viral pneumonia. Physicians often order chest X-rays for more accurate diagnosis.

In recent years, with the explosion of available data and rapid growth of computational powers, deep learning has made astonishing progress in image classification. Furthermore, the success of transfer learning has proven it possible to train a neural network with great accuracy and a relatively small dataset. This project applies a pertained CNN architecture on a dataset of pediatric chest X-ray images to detect pneumonia from normal x-ray, and identify the cause of pneumonia as being bacterial or viral.

The dataset can be downloaded from here[2].

Problem Statement

Traditional approach in detecting pneumonia in chest radiography can be difficult. The results obtained from radiologists can be confusing. Given the same X-ray image, different radiologists can give different diagnoses.[3]

On the other hand, deep learning applications has been demonstrated to be able to consistently deliver high quality result in medical imaging diagnostics, such as detection of diabetic retinopathy[4] and skin cancer[5].

---

[1] Pneumonia claims the lives of the world's most vulnerable children
[2] Dataset of chest X-ray images can be downloaded from here (1.2 gb), which is obtained from the original complete dataset of both OCT and chest X-ray images (7.7 gb).
[3] MISDIAGNOSIS OF PNEUMONIA
[4] Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs
[5] Dermatologist-level classification of skin cancer with deep neural networks

One common approach of deep learning in image classification is transfer learning. That is, rather than building and training a complex neural network architecture from ground up, it repurposes pre-trained networks on a different and specific category. This project examines and experiments with a few of the existing predominant CNN architectures to train on a set of chest X-ray images. In the end, it chooses one of the best performing models to predict if a given image indicates normal or pneumonia, and whether the pneumonia is viral or bacterial.

Metrics

Accuracy, specificity and sensitivity

Metrics used for evaluation are accuracy, specificity and sensitivity scores calculated from binary comparisons between normal and pneumonia, bacterial results vs otherwise.

Normal vs Abnormal (Healthy vs Pneumonia)	Accuracy	How often the model correctly label the x-ray result normal /healthy vs abnormal (pneumonia) ?
	Sensitivity	Of all the images with pneumonia, how many are correctly identified?
	Specificity	Of all the normal images, how many are correctly identified?
Bacterial Pneumonia vs Otherwise (Urgent care vs Non-urgent care)	Accuracy	How often the model correctly label the x-ray result as bacterial vs viral?
	Sensitivity	Of all the images with bacterial pneumonia, how many are correctly identified?
	Specificity	Of all the images with bacterial pneumonia, how many are correctly identified?

Using the following notation:

TP: (**True Positives**) Sick people that we **correctly** diagnosed as sick (pneumonia or bacterial pneumonia).

TN: (**True Negatives**) Healthy people that we **correctly** diagnosed as healthy (normal or non-bacterial pneumonia).

FP: (**False Positives**) Healthy people that we **incorrectly** diagnosed as sick.

FN: (**False Negatives**) Sick people that we **incorrectly** diagnosed as healthy.

We have the following formulas:

Accuracy = (TP + TN) / (TP + TN + FP + FN)

Sensitivity = TP / (TP + FN)

Specificity = TN / (TN + FP)

Confusion Matrix

Additionally, this project uses a confusion matrix to summarize the results of predictions.

II. Analysis

Data Exploration

The labelled X-ray dataset consists of a total of 5856 chest X-ray images. All of them are in JPEG format. They are distributed as following:

	Normal	Pneumonia		**Total**
		Virus	Bacteria	
Train	1349	1345	2538	**5232**
Test	234	148	242	**624**
Total	**1583**	**1493**	**2780**	**5856**

Sample labeled images are included in the project folder: *chest-xray*. The full dataset can be downloaded from here[6]. The images are in grayscale (black for dense structures, white for metal and contrast media, gray for muscles, fat, and fluids). The images have varied dimensions.

Exploratory Visualization

[6] Dataset of chest X-ray images can be downloaded from here (1.2 gb), which is obtained from the original complete dataset of both OCT and chest X-ray images (7.7 gb). The intended original source from kaggle is taken offline.

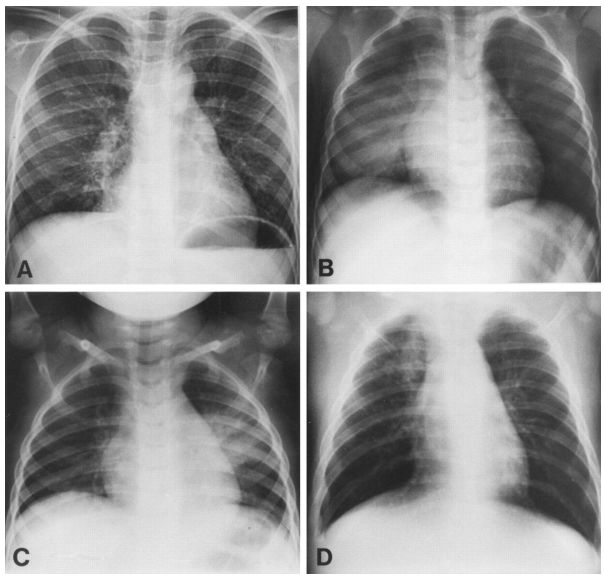
Normal

Bacterial Pneumonia

Viral Pneumonia

The above are three chest X-ray images, each belonging to one of three categories we aim to classify later. When interpreting the x-rays, radiologists look for white spots in the lungs (called infiltrates) that identify an infection.

More examples from the internet:



Chest radiographs of (A) a 6.5 year old girl with widespread interstitial changes in both lungs caused by bacteria; (B) a 1.9 year old boy with alveolar changes in the right lower lobe caused by bacteria; (C) a 2.8 year old girl with alveolar changes in the left lower lobe associated with rhinovirus; (D) a 0.3 year old girl with alveolar changes in the right upper lobe associated with virus.[7]

Pneumonia is airspace disease. The air spaces are filled with bacteria or other microorganisms and pus[8].

Algorithms and Techniques

This project uses transfer learning, a highly effective technique in image classification. It explores a few of the highly successful CNN architectures: Xception, Resnet 50, Inception V2, Inception V3 and use them for pneumonia classification. The steps are below:

[7] https://thorax.bmj.com/content/57/5/438 Differentiation of bacterial and viral pneumonia in children
[8] https://www.med-ed.virginia.edu/courses/rad/cxr/pathology3chest.html

1. Pre-process data
2. Use each of the pretrained models to extract features on train, validation and test dataset. Store the extracted features in separate files;
3. Using the extracted features as input, build a top layer model;
4. Fit and train the top layer model on training and validating dataset to get the best weights. Apply early stopping and model checkpoint to keep track of and save the best weights;
5. With features extracted in step 2 as input and the weights obtained in step 4, run the model on the test sets and output predictions.
6. Obtain accuracy, specificity and sensitivity score from the results from step 5 and choose the best CNN architecture
7. Fine tune the hyper parameters, including target image size, learning rate and drop-out layers.

Benchmark

Baseline Model

During the exploration phase (at the time of proposal writing), I built a baseline model that uses extracted features from Keras' VGG-16 model and a *GlobalAveragePooling2D* pooling and fully-connected top layer. The result is as the following:

	Accuracy	Sensitivity	Specificity
Normal vs Pneumonia	76.1%	81.3%	52.2%
Bacterial vs Others	82.2%	78.4%	65.5%

Secondary Model

The team at University of California has built a model that trained on both Optical Coherence Tomography (OCT) and Chest X-Ray Images and published their results in this article Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning. Their result is the following[9]:

	Accuracy	Sensitivity	Specificity

[9] The team also has accuracy, sensitivity and specificity score for **Binary** comparison of bacterial vs viral pneumonia. However, it is not clear to me how they derive the scores. Other than training the images with pneumonia separately, I am not sure how to reach a binary comparison of only bacterial and viral pneumonia).

Normal vs Pneumonia	92.8%	93.2%	90.1%

III. Methodology

Data Preprocessing

There are 2 steps in data-preprocessing.

1. Dataset restructuring:

I start by re-organizing the images in the given train and test directories into three directories that corresponds to the three categories: *normal, pneumonia_virus, pneumonia_bacteria*. Doing so allows me to easily take advantage of the keras `ImageDataGenerator` and `flow_from_directory` functionality.

The original image dataset does not have a separate validation set. To have a good distribution of validation and training set, I applied a 20/80 split on the train set.

2. Image pre-processing, resize and rescale

Image resizing is done in batches using keras' ImageDataGenerator and its flow_from_directory method. The default image target size is set to 224 by 224 pixels. Image pixel values are normalized to 0.0 to 1.0. This helps the image classification task better remove distortions. Keras image process methods also transformed the images from 3D array to 4D tensor as input with shape

(nb_samples, rows, columns, channels)

When applying the trained-model on individual images to make predictions, we will pre-process image to resize, rescale and reshape the data array to a 4D tensor.

Implementation

Phase one of the implementation is exploration and experimentation of the currently available deep learning models. All of them are from `keras.applications.`

Over the last few years, Deep convolutional neural networks have had a series of breakthroughs in image classification. This project focuses on the following models:

`Resnet50`: `Resnet50` is one of the residual learning networks that tries to tackle the vanishing gradients problem that arises from very deep neural networks. Instead trying to learn the full

features from its previous layer (an underlying mapping from x to H(x)), resnet tries to learn the difference between the two, or the “residual.” Because of that, a lot more layers can be added to the model and can still continuously learn from the data. Resnet proves to be a great improvement from earlier VGG models (VGG-16 and VGG-19)[10].

`InceptionV3`: Inception model employes multiple configurations in parallel in each step. At each layer, Inception uses a set of transformation layers with a pooling layer, then lets the model pick the "winner". To solve the enormous computation cost, Inception also uses 1x1 convolutions to perform dimensionality reduction[11].

`InceptionResnetV2`: a hybrid of inception and resnet networks. Combining residual networks with the latest generation Inception architecture has been demonstrated to be able to outperform Inception networks alone. There are a few variants of Inception and Resnet hybrid[12].

`Xception`: Xception stands for “extreme inception”. It uses a spatial convolution performed independently for each channel followed by a 1×1 convolution across channels. In other words, it looks for correlations on a 2D space and then looks for correlations across a 1D space. Xception is hugely successful. It is also resource efficient[13].

Step 1: Obtain Bottleneck features

I start by instantiating the deep learning model with the weights pre-trained from imagenet and removing the top layer by setting `include_top = false`. For example,

```
Xception(include_top=False, weights='imagenet')
```

Next I run the pre-processed images batches through the model. This is done using `predict_generator` which generates predictions for the input samples. The results are saved in .npy features files.

```
features_train = pretrained_model.predict_generator(generator, predict_size)
np.save(feature_file, features_train)
```

[10] Understanding and Implementing Architectures of ResNet and ResNeXt for state-of-the-art Image Classification: From Microsoft to Facebook
[11] Google AI blog: Train your own image classifier with Inception in TensorFlow
[12] Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning
[13] An Intuitive Guide to Deep Network Architectures

Without GPU, it takes around 6 hours to extract all of the test, train, and validation features using each of the four CNN models.

Step 2: Construct and train top model

Once the features are extracted, I am ready to construct a top layer model for final prediction.

```
model = Sequential()
model.add(GlobalAveragePooling2D(input_shape=data.shape[1:]))
model.add(Dropout(.2))
model.add(Dense(num_classes, activation='softmax'))
```

The top layer model used for prediction is defined as a global average pooling layer followed by a fully connected dense layer of 3 neurons (one for each output class). That is, the extracted features are outputted as spatial average via the global average pooling layer, then it is subsequently fed into the softmax layer. To combat overfitting issues, I added a dropout layer with a dropout rate of .2.

The model architecture's summary:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d_1 ( (None, 2048)              0
_________________________________________________________________
dropout_1 (Dropout)          (None, 2048)              0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 6147
=================================================================
Total params: 6,147
Trainable params: 6,147
Non-trainable params: 0
```

Before I use the model to fit the data, I use the `dataGenerator` to get the class labels for each of the training/validation samples. I also convert the labels to categorical vectors.

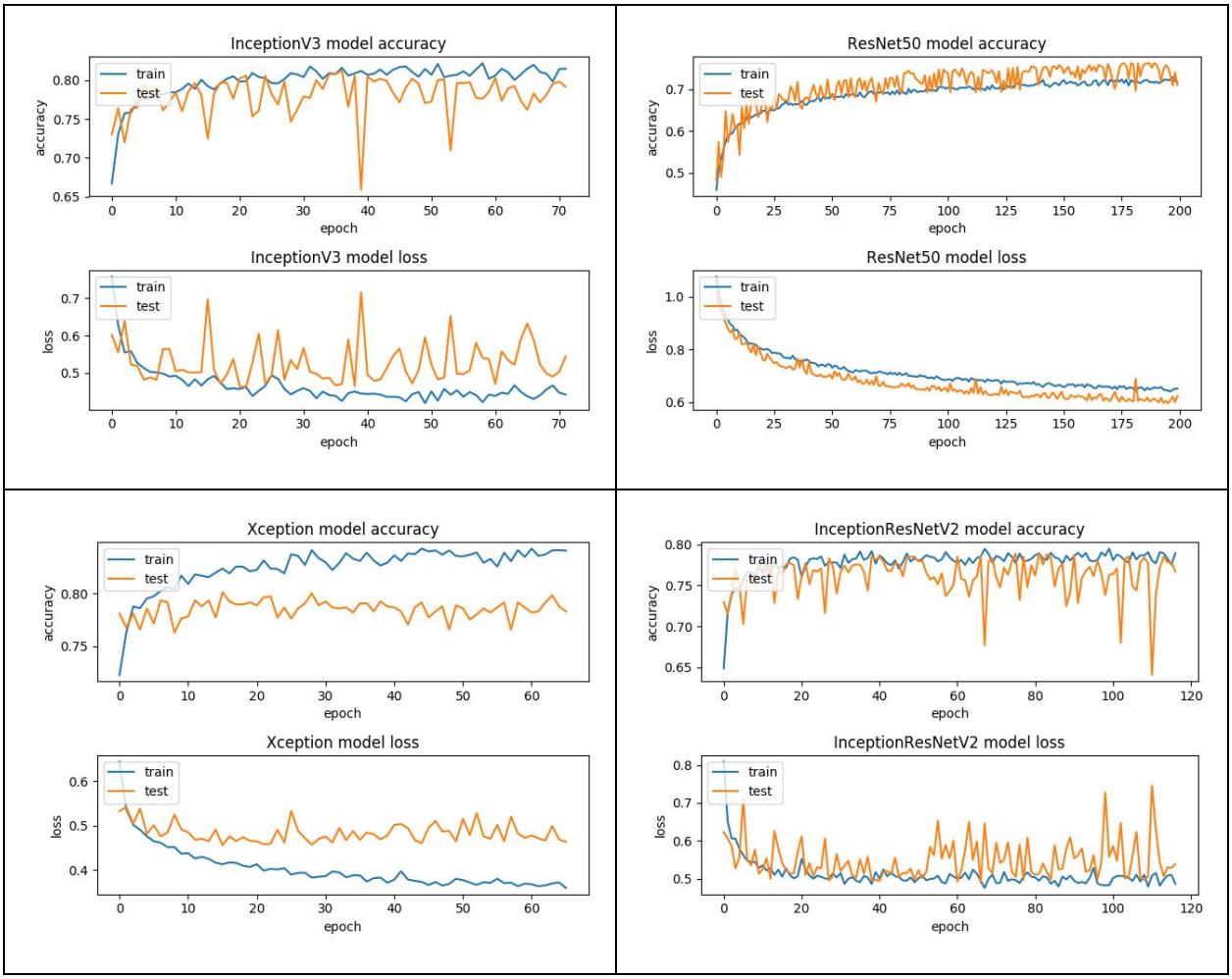
Then I compile the model and train and fit the model. I use an `Adam` optimizer and `categorical_crossentropy loss` function. To keep track of and save the best weights I use `ModelCheckPoint`; To end training when there are no improvements in validation loss, I use early stopping. However, I also set `patience` to be very high, because I observed that training improvement sometimes takes a lot of episode to archive.

```
model = get_top_model(train_data, num_classes)
```

```
earlystop = EarlyStopping(monitor='val_loss', min_delta=0, patience=30, verbose=1,
mode='auto')
model.compile(optimizer=optimizers.Adam(),  loss='categorical_crossentropy',
metrics=['accuracy'])
checkpointer = ModelCheckpoint(filepath=model_weights_path, verbose=1, save_best_only=True)
callbacks_list = [earlystop, checkpointer]
history = model.fit()
model.save_weights(saved_model_weights_path.format(model_name))
```

I plot the training history of categorical_crossentropy loss and accuracy against episode.

The following is the latest training history plot:



Step 3: Obtain predictions and evaluation metrics on the test set

Once I have obtained and saved the best weights from training and validation, I run the models on the test set and get a baseline performance metrics on the models.

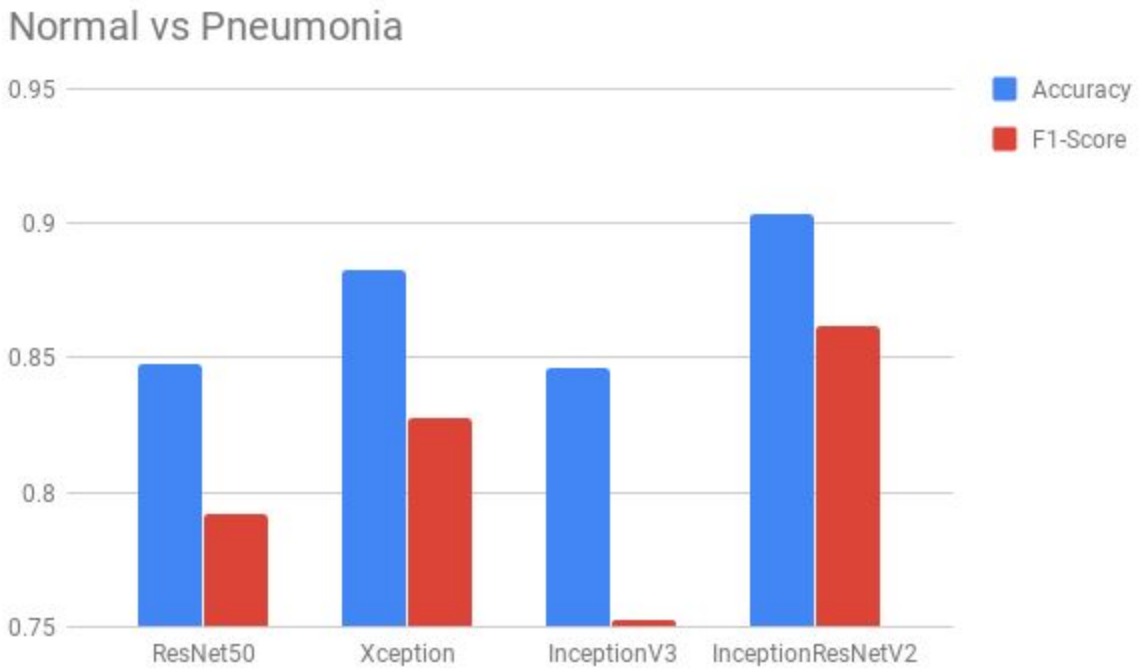
Using the top layer model I defined before, I load the saved weights and run predictions on the test features extracted earlier.

```
model.load_weights(weights_path)
preds = model.predict(test_data, batch_size=batch_size)
```

Running the prediction results though the accuracy, specificity, sensitivity and F1 score calculations, I get the following results:

	Accuracy	Specificity	Sensitivity	F1-Score
ResNet50				
Normal vs pneumonia	0.8478	0.6795	0.9487	0.7918
Bacterial Pneumonia vs otherwise	0.6731	0.5052	0.938	0.6567
Viral Pneumonia vs otherwise	0.7965	0.9916	0.1689	0.2887
InceptionV3				
Normal vs pneumonia	0.8462	0.6068	0.9897	0.7524
Bacterial Pneumonia vs otherwise	0.7676	0.6309	0.9835	0.7687
Viral Pneumonia vs otherwise	0.8349	0.9433	0.4865	0.6419
Xception				
Normal vs pneumonia	0.883	0.7137	0.9846	0.8275
Bacterial Pneumonia vs otherwise	0.8846	0.8377	0.9587	0.8941
Viral Pneumonia vs otherwise	0.8542	0.895	0.723	0.7998
InceptionResNetV2				
Normal vs pneumonia	0.9038	0.765	0.9872	0.862
Bacterial Pneumonia vs otherwise	0.9071	0.9162	0.8926	0.9042
Viral Pneumonia vs otherwise	0.8686	0.8676	0.8716	0.8696

Performance comparison among the models:



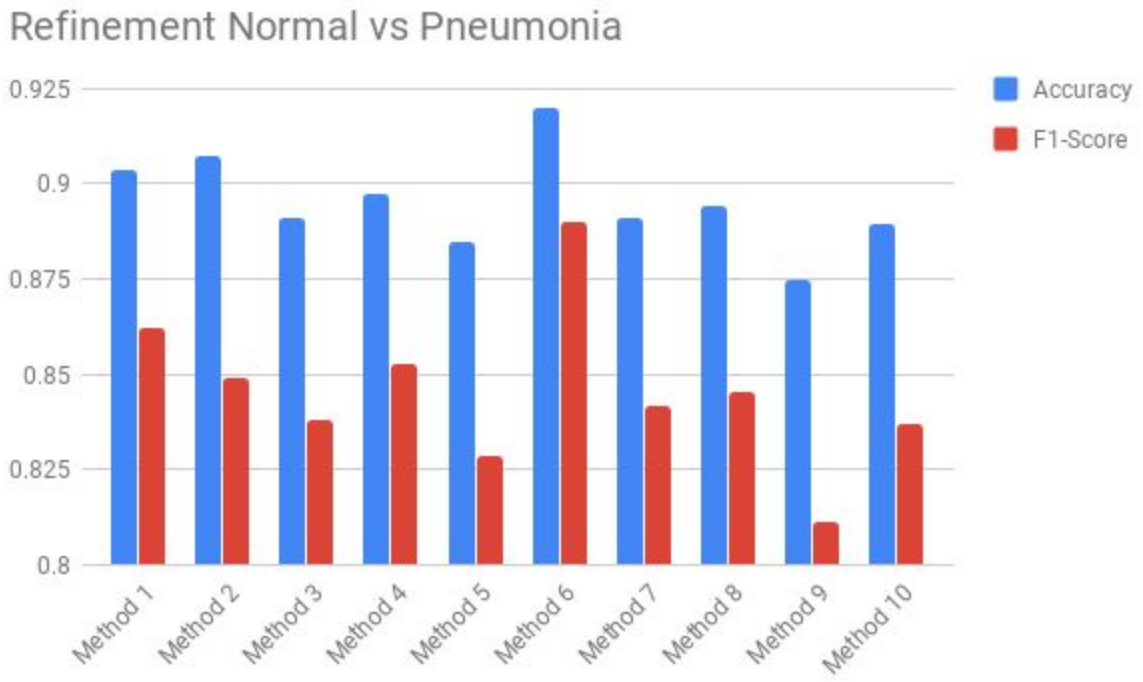
Step 4: Choosing the model and refinement

As seen from the above comparison, `InceptionResNetV2` has outperformed the rest of the models, so now I am able to focus on the `InceptionResNetV2` model and proceed to refine and improve.

I try to adjust the images to different target sizes, for example, 299 by 299, 320 by 320. Different learning rates (.0001, .0002, .0005, 0.001) and dropout rates are also applied (.2, .3, .4, .5).

I also attempt to use dynamic learning rate methods to gradually reduce learning rate through Keras' `LearningRateScheduler` and a step decay method. This effort proves to be ineffective.

Overall, with a learning rate of 0.0005 and dropout rate of .2 and image target size of 299 by 299, I am able to achieve a better result than the default setup in the model experimentation phase.



Note:

*Method 1: Target Image size: **224** / Learning Rate: .001 / .2 drop out*
*Method 2: Target Image size: **299** / Learning Rate: .001 / .2 drop out*
*Method 3: Target Image size: **320** / Learning Rate: .001 / .2 drop out*
*Method 4: Target Image size: **299** / Learning Rate: **.0001** / .2 drop out*
*Method 5: Target Image size: **299** / Learning Rate: **.0002** / .2 drop out*
*Method 6: Target Image size: **299** / Learning Rate: **.0005** / .2 drop out*
*Method 7: Target Image size: **299** / Learning Rate: .0005 / **.1 drop out***
*Method 8: Target Image size: **299** / Learning Rate: .0005 / **.3 drop out***
*Method 9: Target Image size: **299** / Learning Rate: .0005 / **.4 drop out***
*Method 10: **Dynamic learning rate with step decay***

IV. Results

Model Evaluation and Validation

So in summary, after exploring the most successful CNN models made available from Keras' application packages and running them through the train, validation and test sets, after comparing the accuracy, sensitivity and specificity metrics obtained from the testing set, I choose the one that performs the best, which is the `InceptionResnetV2` model.

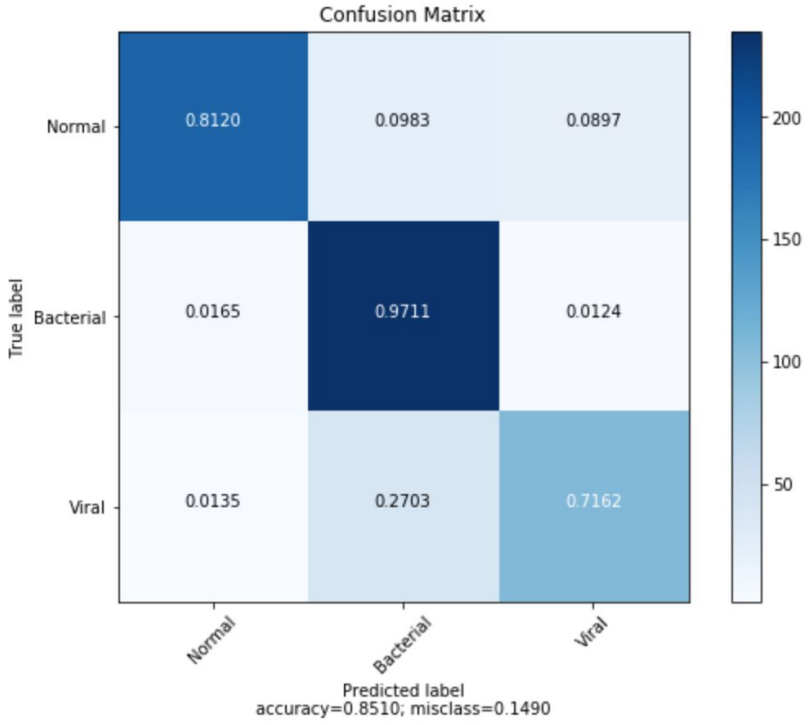
Afterwards I created a top layer model that consists of a global average pooling layer, a dropout layer and a fully connected layer for the final prediction. After various attempts to tune the

learning rate, dropout rate, target image size, I found that with a learning rate of 0.0005, dropout rate of 0.2 and image target size of 299, the CNN model achieves the best result.

Final Result:

	Accuracy	Specificity	Sensitivity	F1-Score
Normal vs pneumonia	0.92	0.81	0.98	0.89
Bacterial vs otherwise	0.89	0.84	0.97	0.90

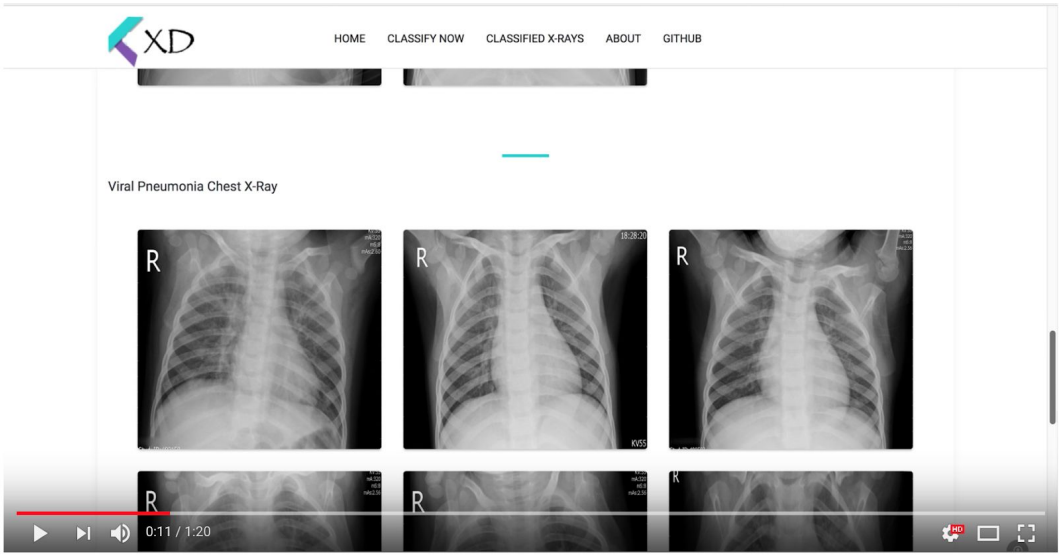
Confusion Matrix Chart:



Further testing on other images

I have used the model to test a set of chest x-ray images found online. The CNN model is able to correctly identify them.

I have also built a small web app to make a prediction in real time. Unfortunately, I have not been able to deploy it successfully to production without incurring huge cost. A short demo video of this app running locally can be viewed here.



https://youtu.be/RfauqDz8W5k

Justification

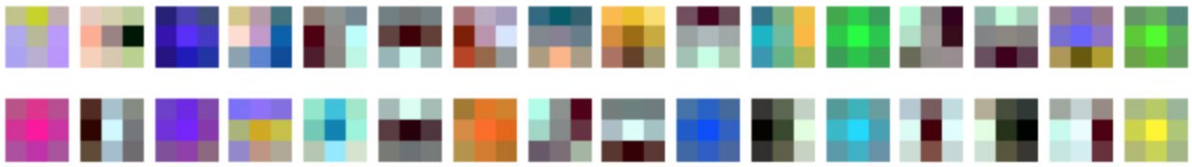
The final model performs very well and has greatly surpassed the VGG-16 baseline model I used during the project exploration phase. It is comparable to the performance published in this article Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning (it scores high on sensitivity, but lags behind in terms of specificity)

V. Conclusion

Free-Form Visualization

Visualize the weights

To better understand the inner working of the CNN model and the InceptionResnetV2 model used for transfer learning, I plotted the weights of one of the first layers.



Visualize Activations

Output of one of the very first layers in InceptionResnetV2:

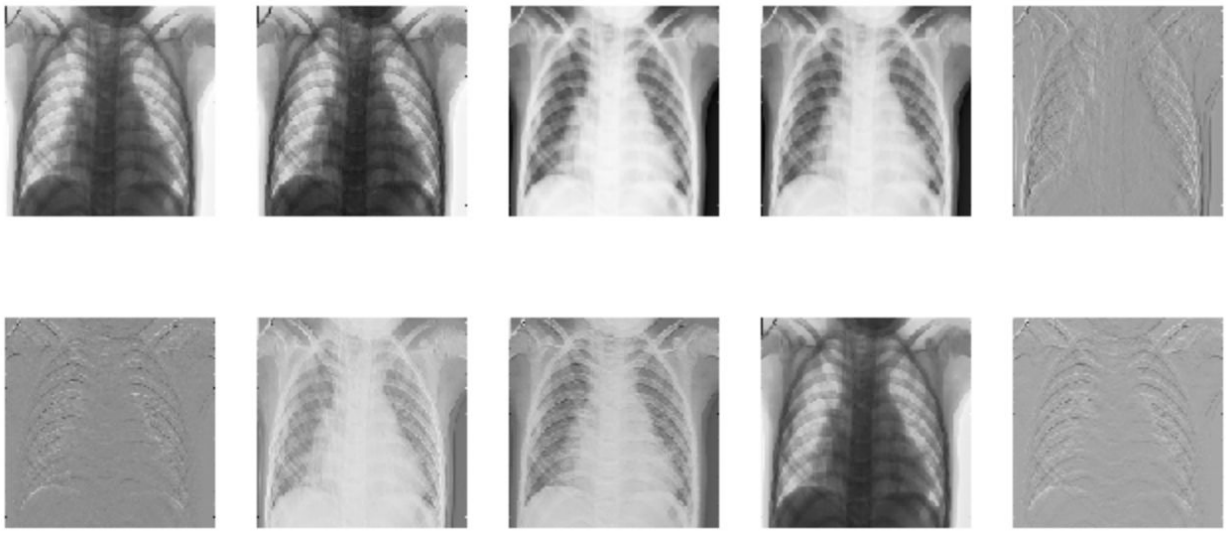
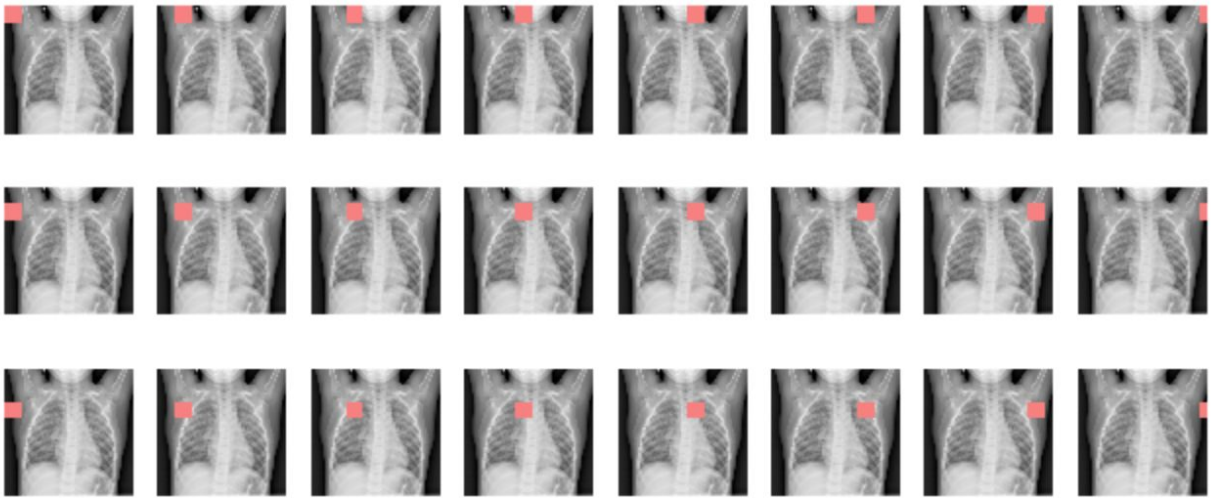


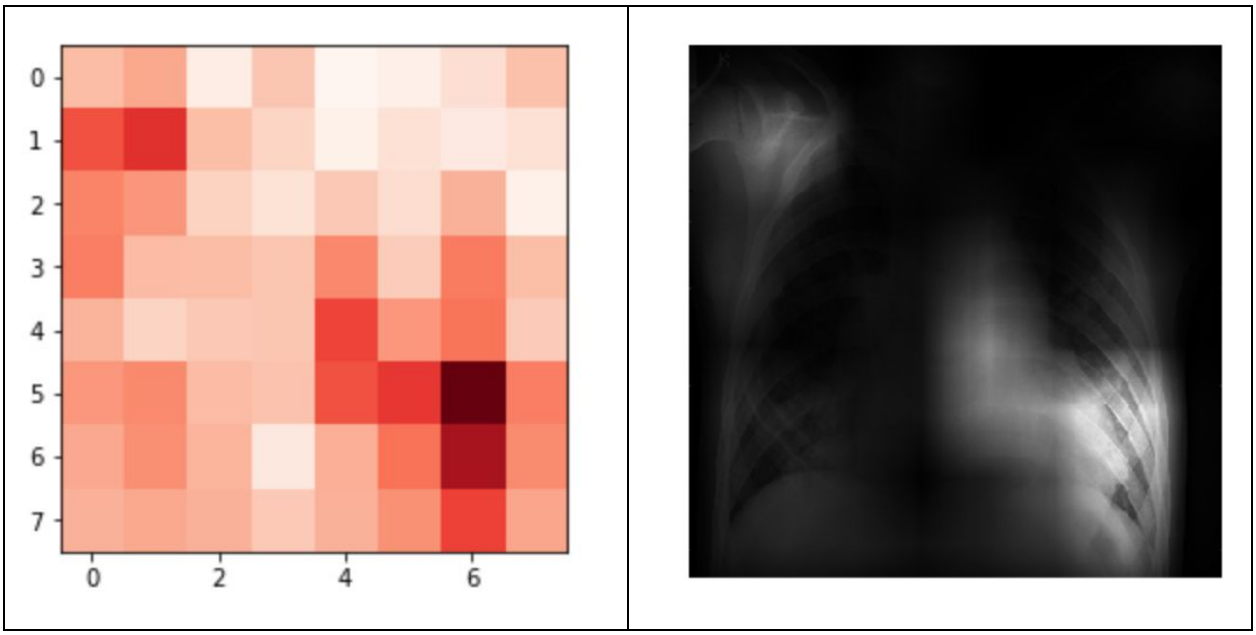
Image Occlusion:

To have a better sense of exactly how the CNN networks derive the final prediction and what location gives a high probability for the final prediction, I use a occlusion method that systematically occludes a small square of the input image and monitor the output with its probability.

Occluding output:



Resulting Heatmap and output image with heatmap mask applied:

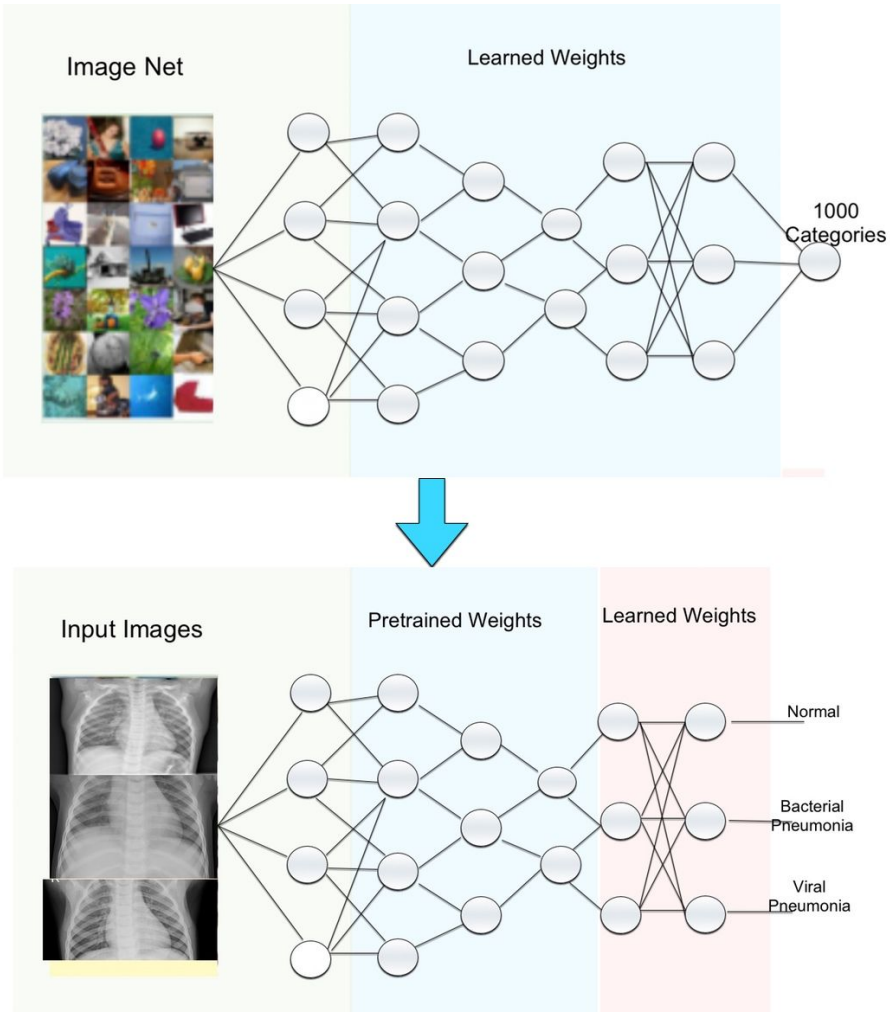


This shows the model is localizing the correct spot for its prediction.

Reflection

There has been tremendous advancement in machine learning across many fields, including health care. Deep learning neural networks has proven that it can perform on par or outperform the best medical professionals. This project in particular uses a pre-trained deep learning network as a feature extractor to extract bottleneck features from a dataset of 5856 chest x-ray images, then build a CNN model that uses the features as input to predict whether a given chest x-ray radiograph is normal or pneumonia, furthermore, it identifies whether an abnormal result (i.e., pneumonia) is caused by bacteria or virus, therefore aiding patients for further treatment if needed.

This is made possible by the transfer learning method in image classification and the ever improving and more powerful CNN neural networks that are trained with imagenet that has over a million images spanning over a thousand of categories.



This project uses the InceptionResnetV2 model, a hybrid of Inception and Resnet networks.

Difficulties

Though it is relatively easy to repurpose a pre-trained network to train on a new category of images and get reasonable results, it is still difficult and even mysterious to fine tune and have great insights how the model predicts the way it does.

I found it hard to choose from the many tools and parameters and fine tune them. I have tried various ways to adjust the drop-out rate, including adding different layers and using a flatten layer instead of global average pooling. The effort generally proves to be counter-productive.

I also find it hard, even impossible, to grasp the deep architecture of the pre-trained CNN models and identify their strengths and weaknesses. I was surprised to see that the InceptionV3 model does not perform as well as I expected.

Improvement

Other than resizing and normalizing the image pixels (by dividing each pixel with 255), the project has not performed any other image augmentation. In the future, I think I can apply some image distortion and transformation to enhance the robustness of the CNN model, though without actual implementation I cannot be certain that it will truly help. Also as a person who has no medical expertise, I am not sure if image transformation would alter the pre-labelled outcome.

I would also like to do research on how to better visualize the CNN network, layer by layer, all the way to the very top fully connected layer.

I am also very disappointed that I have not been able to deploy the model to production and make a web application available to all so live testing can be performed. To me, it is still a very difficult and expensive process trying to deploy a flask web application with a resource-heavy CNN model. I have tried with heroku, Google Cloud and FloydHub. Unfortunately, none of the many attempts succeeded (unless I were willing to pay for high cost).