CNN for Detecting Pneumonia from X-ray Images

SPRINGBOARD DATA SCIENCE CAPSTONE PROJECT 2

REETI BHAGAT

SEP 25, 2020

Table of Contents

Capstone Project 2 – Milestone Report

CNN For Detecting Pneumonia from X-ray images

1. Problem Statement

Pneumonia is lung inflammation caused by infection with virus, bacteria, fungi or other pathogens. It is an acute respiratory infection which affects the lungs which can be detected by analyzing chest x-rays. It becomes life-threatening for infants, people having other diseases, people with an impaired immune system, elderly people, people who are hospitalized and have been placed on a ventilator, people with chronic disease like asthma and people who smoke cigarettes. The cause of pneumonia also determines is severity.

The analysis of chest radiography has a crucial role in medical diagnostics and the treatment of the disease. The Centers for Disease Control and Prevention (CDC, Atlanta, GA, USA) reported that about 1.7 million adults in the US seek care in a hospital due to pneumonia every year, and about 50,000 people died in United States from pneumonia in 2015 [1]. Chronic obstructive pulmonary disease (COPD) is the primary cause of mortality in the United States, and it is projected to increase by 2020 [2]. The World Health Organization (WHO, Geneva, CH) reported that it is one of the leading causes of death all over the world for children under 5 years of age, killing an estimated 1.4 million, which is about 18% of all children deaths under the age of 5 years worldwide [3]. More than 90% of new diagnoses of children with pneumonia happen in the underdeveloped countries with few medical resources available. Therefore, the development of cheap and accurate pneumonia diagnostics is required.

Although currently, deep learning still cannot replace doctors/clinicians in medical diagnosis, it can provide support for experts in the medical domain in performing time-consuming works, such as examining chest radiographs for the signs of pneumonia. Recently, a number of researchers have proposed different artificial intelligence (AI)-based solutions for different medical problems. Due to the success of deep learning algorithms in analyzing medical images, Convolutional Neural Networks (CNNs) have gained much attention for disease classification.

2. Dataset

Data: https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia

The dataset consists of three main folders (i.e., training, testing, and validation folders) and two subfolders containing pneumonia (P) and normal (N) chest X-ray images, respectively. A total of 5,856 X-ray images of anterior-posterior chests were carefully chosen from retrospective pediatric patients between 1 and 5 years old. The entire chest X-ray imaging was conducted as part of patients' routine medical care.

2.1. Data Acquisition using Google Colab

I used Google Colab for this project knowing that it provides free GPU. Google Colab notebooks are Jupyter notebooks that run in the cloud and are highly integrated with Google Drive, making them easy to set up, access, and share. The dataset was readily available on Kaggle. I uploaded the data to my personal Google drive and mounted my drive on Colab. The base directory contains train, test and validation subdirectories for the training, testing and validation datasets, which in turn each contain normal and pneumonia subdirectories. Then I assigned variables with the proper file path for the training, validation and test sets and also assigned variables with the proper file path for the normal and pneumonia images.

2.2. Data Inspection and Visualization

After assigning variables to the subdirectories of the dataset, I found out the total number of normal and pneumonia images in each directory:

```
Number of images in train_normal_dir: 1342
Number of images in train_pneumonia_dir: 3876
Number of images in val_normal_dir: 9
Number of images in val_pneumonia_dir: 9
Number of images in test_normal_dir: 234
Number of images in test_pneumonia_dir: 390
```

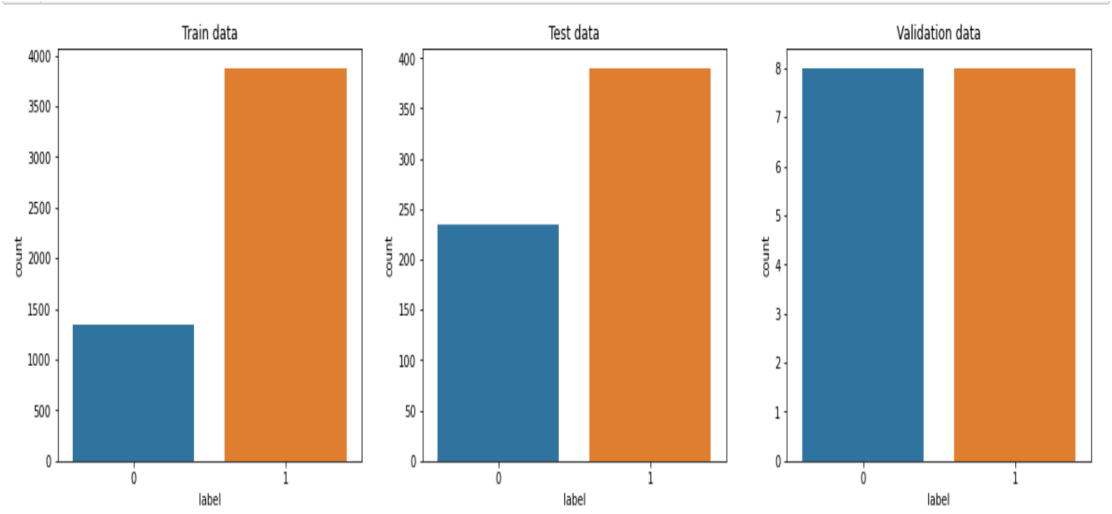


Figure 1 Number of images in each set

Figure 1 illustrates that the number of images per class is not equally distributed; the number of normal images is relatively fewer than the number of pneumonia images in train set.

Next I displayed a few images to get a better sense of what the normal and pneumonia datasets look like.

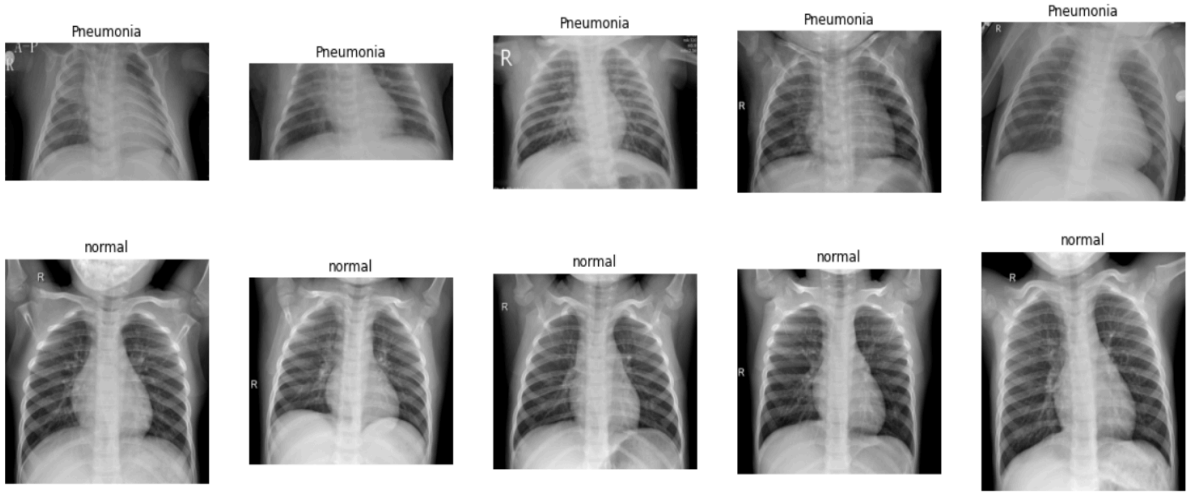


Figure 2. Sample images

Figure 2 illustrates that the x-ray images come in different sizes. These images need to be resized to the same aspect ratio before feeding them into the neural network.

2.3. Data Preprocessing

I have prepared my data for modelling by loading image, resizing it, converting to grayscale, normalization of image and reshaping the dimension required for TensorFlow. Then, I have loop through images to generate array in the form matrix and labels of images. I have created functions to split train, Val and test.

```
def process_data(img_path):
    img = cv2.imread(img_path)
    img = cv2.resize(img, (150, 150))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    img = img/255.0
    img = np.reshape(img, (150,150,1))

    return img

def compose_dataset(df):
    data = []
    labels = []

    for img_path, label in df.values:
        data.append(process_data(img_path))
        labels.append(label)

    return np.array(data), np.array(labels)
```

Using functions,let's prepare train ,test and validation arrays from dataframes

```
X_train, y_train = compose_dataset(train_df)
X_test,  y_test = compose_dataset(test_df)
X_val, y_val = compose_dataset(val_df)
```

```
print('Train data shape: {},Labels shape :{}'.format(X_train.shape ,y_train.shape))
print('Test data shape: {},Labels shape :{}'.format(X_test.shape ,y_test.shape))
print('Val data shape: {},Labels shape :{}'.format(X_val.shape ,y_val.shape))
```

```
Train data shape: (5216, 150, 150, 1),Labels shape :(5216,)
Test data shape: (624, 150, 150, 1),Labels shape :(624,)
Val data shape: (16, 150, 150, 1),Labels shape :(16,)
```

Data augmentation:

I employed several data augmentation methods to artificially increase the size and quality of the dataset. .is process helps in solving overfitting problems and enhances the model's generalization ability during training. Data transformation or augmentation is a powerful technique which helps in almost every case for improving the robustness of a model. This technique can prove to be even more helpful when the dataset is imbalanced. You can generate different samples of undersampled classes in order to try to balance the overall distribution.

In this section, I used ImageDataGenerator class provided by tf. keras which can read images from disk and convert them to float32 tensors and feed them (with their labels) to the network.

The rotation range denotes the range in which the images were randomly rotated during training, i.e., 10 degrees. Width shift is the horizontal translation of the images by 0.1 percent, and height shift is the vertical translation of the images by 0.1 percent. The zoom range randomly zooms the images to the ratio of 0.1 percent, and finally, the images were flipped horizontally.

```
:
# Image augmentation
#define generator
datagen= ImageDataGenerator(
    rotation_range=10,
    zoom_range=0.1,
    width_shift_range=0.1,
    height_shift_range=0.1,
    horizontal_flip=True,
    vertical_flip=False
)

datagen.fit(X_train)
```

3. Application of CNN

3.1 Baseline Model

Process of CNN is to detect and categorize images from learned features. It is very effective in a multi-layered structure when obtaining and assessing necessary features of graphical images. Figure 1 illustrates the CNN process.

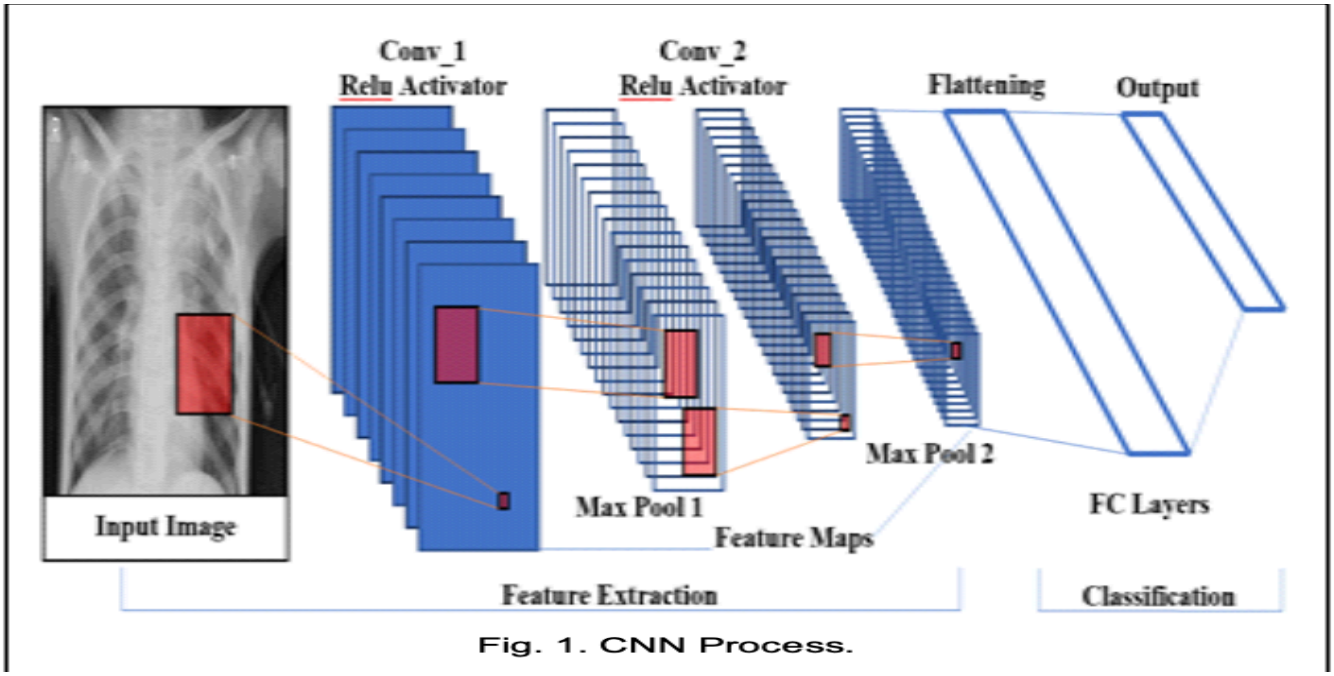


Fig. 1. CNN Process.

The convolutional neural network (CNN) is a deep learning algorithm commonly used in computer vision applications. I deployed Keras open-source deep learning framework with TensorFlow backend to build and train the convolutional neural network model from scratch.

The next step was to build the model. This can be described in the following 5 steps.

1. I used five convolutional blocks comprised of convolutional layer, max-pooling and batch-normalization.
2. On top of it I used a flatten layer and followed it by four fully connected layers.
3. Also in between I have used dropouts to reduce over-fitting.
4. Activation function was Relu throughout except for the last layer where it was Sigmoid as this is a binary classification problem.
5. I have used Adam as the optimizer and cross-entropy as the loss.

Before training the model is useful to define one or more callbacks. Pretty handy one, are: Model Checkpoint and Early Stopping.

- **Model Checkpoint**: when training requires a lot of time to achieve a good result, often many iterations are required. In this case, it is better to save a copy of the best performing model only when an epoch that improves the metrics ends.
-
- **Early Stopping**: sometimes, during training we can notice that the generalization gap (i.e. the difference between training and validation error) starts to increase, instead of decreasing.

- This is a symptom of overfitting that can be solved in many ways (*reducing model capacity*, *increasing training data*, *data augmentation*, *regularization*, *dropout*, etc). Often a practical and efficient solution is to stop training when the generalization gap is getting worse.

Our initial model will be a simple 8-layer convolutional neural network with max pooling and a ReLU activation function. Let's see how well the model does with the validation and testing set.

CNN model consists of two major parts: the feature extractors and a classifier (sigmoid activation function).

CNN model consists of two major parts: the feature extractors and a classifier (sigmoid activation function).

Each layer in the feature extraction layer takes its immediate preceding layer's output as input, and its output is passed as an input to the succeeding layers. The proposed model in Figure 3 consists of the convolution, max-pooling, and classification layers combined together. The feature extractors comprise 2 × Conv2D, 32; 2 × Conv2D, 64; 2 × Conv2D, 64; 2 × Conv2D, 128, 2 × Conv2D,256 with strides 1, max-pooling layer of size 2 × 2 with strides 2, and a relu activator between them

The first convolution layer applies 32convolutions (with a 3 × 3 kernel) to the input image followed by the relu activation function to introduce non-linearity into the model. The result is a feature map with dimensions 150 × 150 ×32. This convolution layer is followed by a max-pooling layer which downsamples the model by a factor of 2 which yields a feature map with dimensions 75 × 75 × 32. This downsampling allows each convolution stage to learn patterns at a different scale.

For the following feature maps, I also used batch-normalization in between the convolution layers and the max-pooling layers. Batch normalization has the effect of dramatically accelerating the training process of a neural network, and in some cases improves the performance of the model via a modest regularization effect. The output of the convolution and max-pooling operations with batch normalizations are 75 × 75 × 64, 38 × 38 × 64, 19 × 19 × 128 and 10 × 10 × 256 sizes of feature maps, respectively, for the convolution operations and 38× 38 × 64, 19 × 19 × 64 , 10 × 10 × 128 and 5 × 5 × 256 sizes of feature maps from the pooling operations.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 150, 150, 32)      320
_________________________________________________________________
batch_normalization_20 (Batc (None, 150, 150, 32)      128
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 75, 75, 32)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 75, 75, 64)        18496
_________________________________________________________________
dropout_16 (Dropout)         (None, 75, 75, 64)        0
_________________________________________________________________
batch_normalization_21 (Batc (None, 75, 75, 64)        256
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 38, 38, 64)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 38, 38, 64)        36928
_________________________________________________________________
batch_normalization_22 (Batc (None, 38, 38, 64)        256
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 19, 19, 64)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 19, 19, 128)       73856
_________________________________________________________________
dropout_17 (Dropout)         (None, 19, 19, 128)       0
_________________________________________________________________
batch_normalization_23 (Batc (None, 19, 19, 128)       512
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 10, 10, 128)       0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 10, 10, 256)       295168
_________________________________________________________________
dropout_18 (Dropout)         (None, 10, 10, 256)       0
_________________________________________________________________
batch_normalization_24 (Batc (None, 10, 10, 256)       1024
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 5, 5, 256)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 6400)              0
_________________________________________________________________
dense_8 (Dense)              (None, 128)               819328
_________________________________________________________________
dropout_19 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_9 (Dense)              (None, 1)                 129
=================================================================
Total params: 1,246,401
Trainable params: 1,245,313
Non-trainable params: 1,088
```

Fig 3. Model summary

The classifier is placed at the far end of the convolutional neural network (CNN) model. This classifier requires individual features (vectors) to perform computations. The output of the feature extractor (CNN part) is converted into a 1D feature vector for the classifiers. This process is known as flattening where the output of the convolution operation is flattened to generate one lengthy feature vector for the dense layer to utilize in its final classification process.

.
This model has a total of 1,246,401 parameters; 1,245,313 of them are trainable and 1,088 of them are non-trainable.

3.2 Compiling The Model

Compiling a model in Keras involves selecting a loss function, optimizer function and output metrics to be reported during training. I trained the model with the binary_crossentropy loss, because it's a binary classification problem and the final activation is a sigmoid function. Cross-entropy calculates a score that summarizes the average difference between the actual and predicted probability distributions. I used Adam as the optimizer, it automatically adapts the learning rate during training. Since this is a classification problem, I had Keras report on the accuracy metric.

3.3 Training The Model

Before training the model, I used Keras callback functions to get a view on internal states and statistics of the model during training. These functions help to visualize how the model's training is going, and help prevent overfitting by implementing early stopping or customizing the learning rate on each iteration.

The first callback function that I applied was ModelCheckpoint. ModelCheckpoint callback was used in conjunction with training using model.fit () to save a model or weights at some interval, so the model or weights can be loaded later to continue the training from the state saved. I assigned the file path to save the model and set the 'save_best_only', 'save_weights_only' parameters equal to True.

The second callback function that I applied was ReduceLROnPlateau which monitors a quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced. Without using this callback, validation accuracy did not show any improvement at a couple of training sessions. So, I applied the ReduceLROnPlateau callback and set the 'monitor'

parameter equal to ‘val_loss' and ‘mode’ parameter equal to ‘max’ so that mode learning rate would be reduced when the validation accuracy has stopped increasing.

Finally, I used EarlyStopping callback to prevent overtraining of the model by terminating the training process if it’s not really learning anything. I monitored the ‘val_loss’ parameter, set the patience parameter equal to 1 that is the n umber of epochs with no improvement after which training will be stopped.

.

3.4 Evaluating Accuracy and loss for the Model

The accuracy and loss values for the training and testing datasets were recorded over the course of training. Figure 4 illustrates the training and validation accuracy vs epoch and training and validation loss vs epoch. The training accuracy (in blue) gets close to 97% while the validation accuracy (in orange) gets close to 69%.

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.

We will get the confusion matrix from our predictions and see what is the recall and precision of our model.

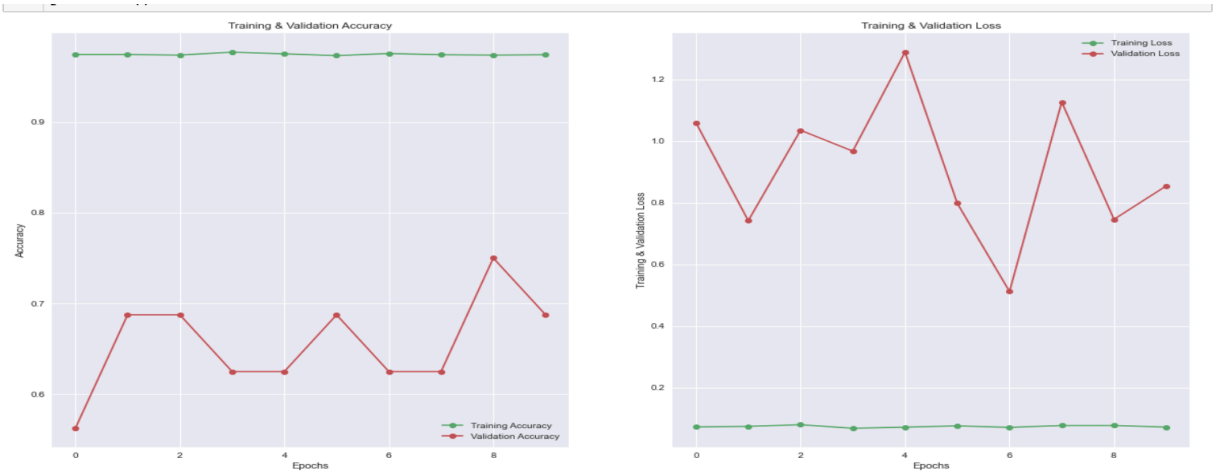


Fig. 4 Training History plots

3.5 Model Performance

The model predicted the image classes on the test data with a 90% accuracy and 0.34 loss.

The test data had 624 images, 234 of them belonged to the 'normal' class and 390 of them belonged to the 'pneumonia' class.

	precision	recall	f1-score	support
Normal (Class 0)	0.95	0.78	0.86	234
pneumonia (Class 1)	0.88	0.98	0.93	390
accuracy			0.90	624
macro avg	0.92	0.88	0.89	624
weighted avg	0.91	0.90	0.90	624

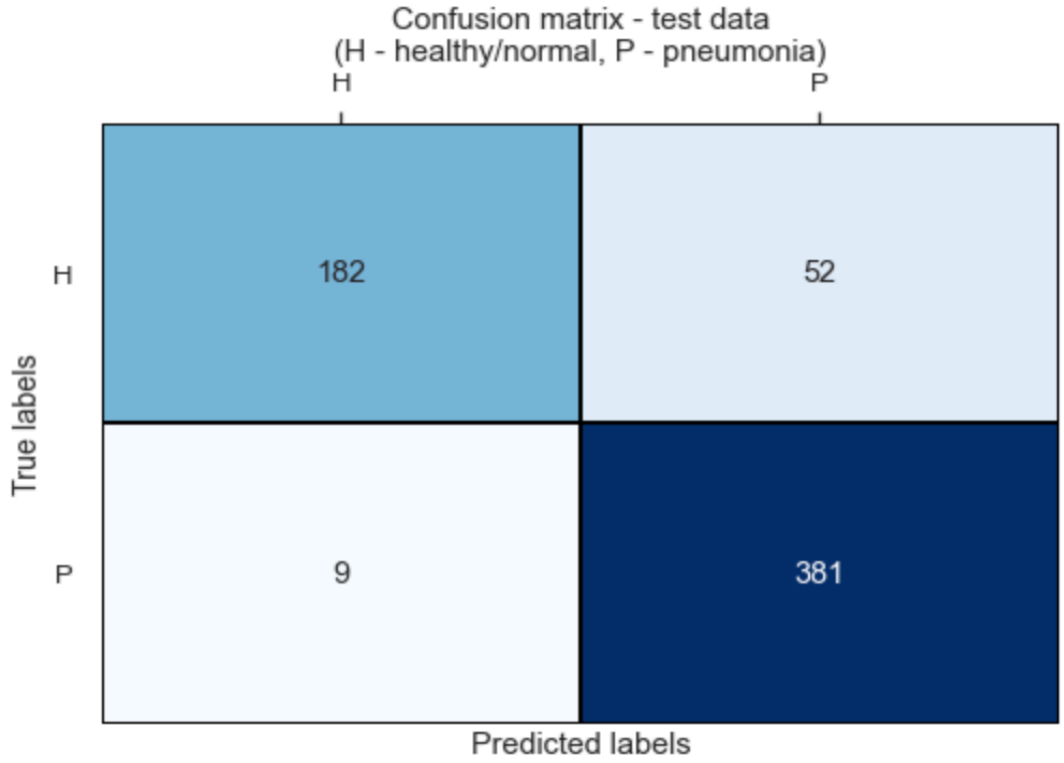


Figure 5. Classification Report and confusion matrix

The performance of the baseline model was also observed using classification report and confusion matrix as shown in Figure 5. Precision determines what percent of the model's predictions were correct. For class 0 ('normal' images), precision is 95 %, meaning that 180 images were classified as 'normal' out of a total number of 191 (182+ 9) 'normal' classified images. For class 1 ('pneumonia' images), precision is 88 %, meaning that 381 images were classified as 'pneumonia' out of a total number of 433 (381 + 52) 'pneumonia' classified images. Recall is the ability of a classifier to find all positive instances. Recall determines what percent of the positive cases the model identified. Class 1 ('pneumonia') has a higher recall score than class 0 ('normal'). This means that the model is performing better in terms of identifying the images that have pneumonia. Recall for the 'normal' class is 78 %. This means that 182 out of 234 (182+ 52) actually 'normal' images were classified as 'normal'. And recall for
the 'pneumonia' class is 98%, meaning that 381 out of 390 (381 + 9) actually 'pneumonia' images were classified as 'pneumonia '.

4. Transfer Learning with VGG16

Transfer learning involves using a pre-trained network. A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. The pretrained model can be used as is or can be used as a transfer learning to customize the model to a given task. The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. This helps to take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.

For this project, I applied the VGG16 pre-trained model to classify the chest x-ray images. There are several approaches for feature extraction using transfer learning. I chose removing the fully connected layers of the VGG16 network, placing a new set of FC layers on top of the CNN.

After loading the VGG16 model, I set the "include top" argument to False. The fully connected output layers of the model used to make predictions were not loaded. This allowed me to add and train a new output layer. I added a new Flatten layer after the last pooling layer in the VGG16 model. Then I defined a new classifier model with three Dense fully connected layers and an output layer that will predict the probability for two classes. Each of the

Dense layers were followed by a Dropout layer to prevent overfitting. I set the “trainable” property on each of the layers in the loaded VGG model to False prior to training. This would allow it to use the VGG16 model layers, and train the new layers of the model without updating the weights of the VGG16 layers.

The model was fit using an ‘adam’ optimizer with a ‘binary-cross entropy’ loss function. All of the other hyperparameters such as number of epochs.

Figure 6 illustrates the training and validation accuracy vs epoch and training and validation loss vs epoch . The training accuracy (in blue) gets close to 95% while the validation accuracy (in orange) gets close to 93% which is an indicator of overfitting. The validation loss for validation data fluctuates over several epochs, on the other hand train loss steadily decreases.

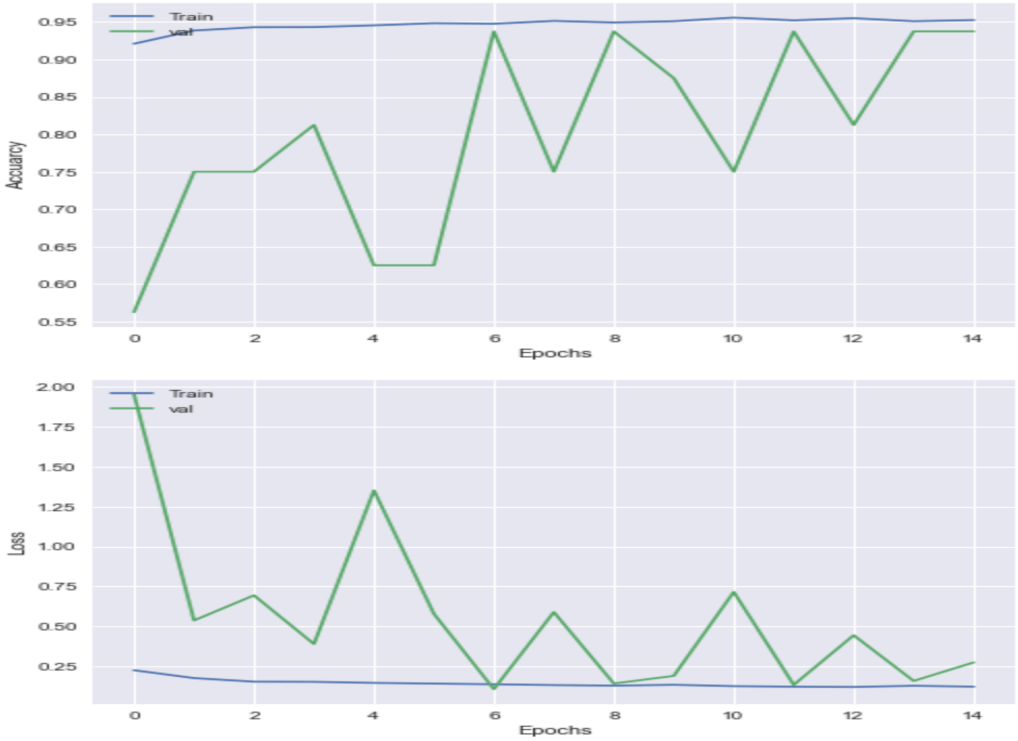


Figure 6 Training history plots

As shown in Figure 7, the transfer learning model is performing slightly lower than the CNN model which was built from scratch. The number of False positives and False Negatives are slightly higher in this model.

	precision	recall	f1-score	support
0	0.87	0.86	0.86	234
1	0.92	0.92	0.92	390
accuracy			0.90	624
macro avg	0.89	0.89	0.89	624
weighted avg	0.90	0.90	0.90	624

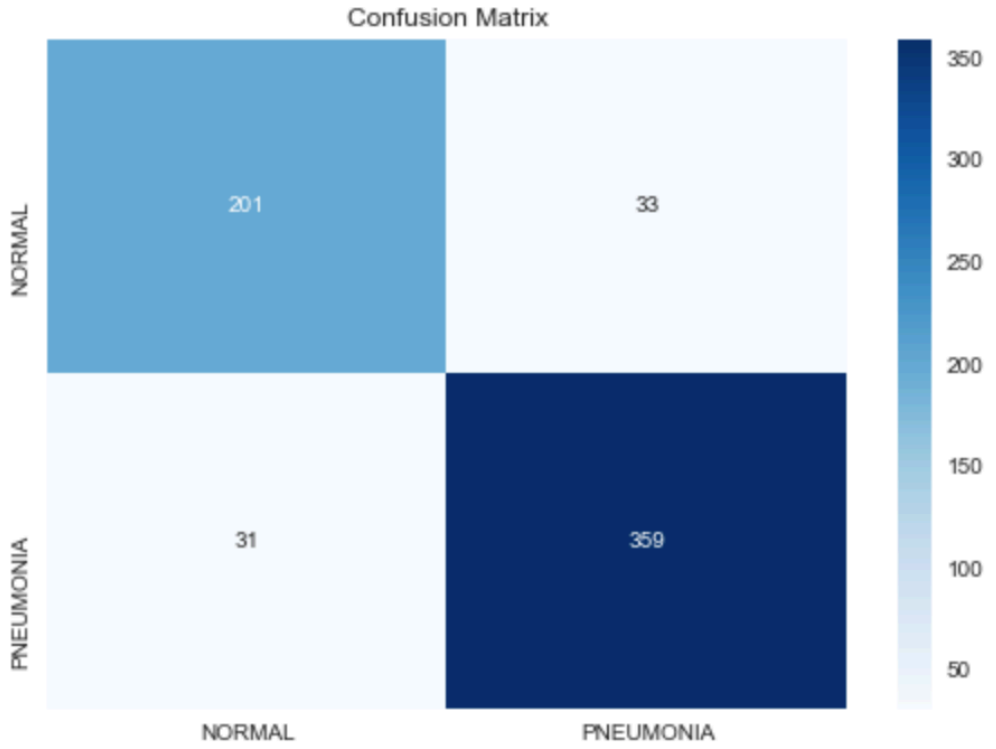


Figure 7. Classification Report and Confusion matrix

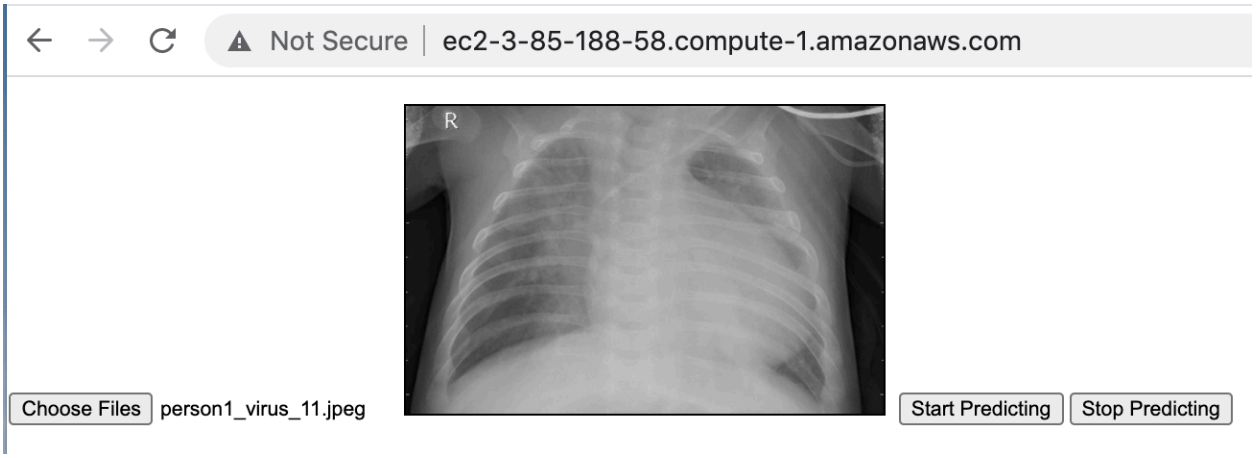
5. BUILD AND DEPLOY PNEUMONIA DETECTION WEB APP WITH GCP AUTO ML VISION WITH TENSORFLOW.JSON

1. Export tensorflow.js files from GCP to run the model in a web browser.
2. Create web app and use tensorflow.js library to call the model to predict image.

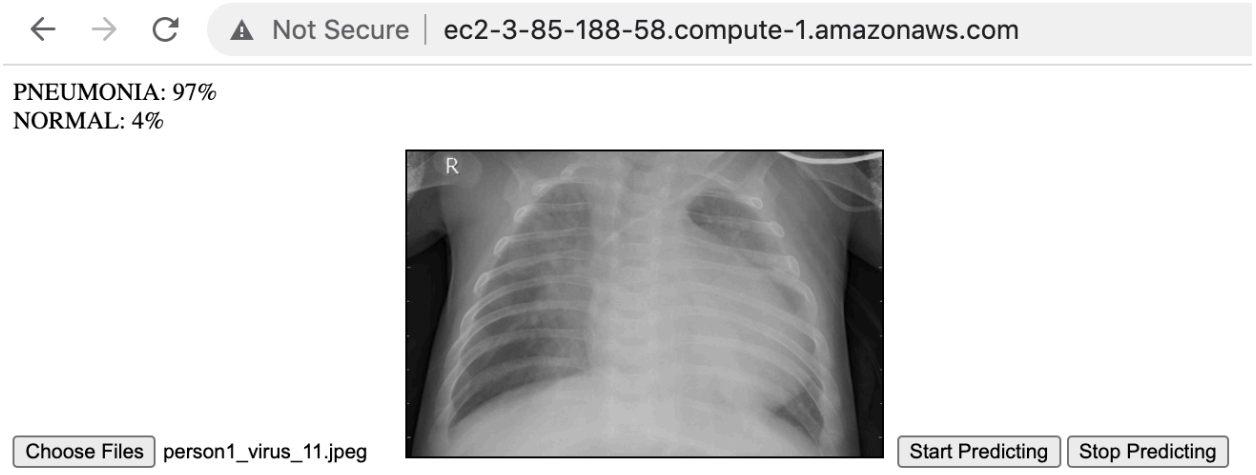
3. Deploy the web app to AWS Ec2 instance and generated public URL to use from anywhere from browser.

App is running at http://ec2-3-85-188-58.compute-1.amazonaws.com/

5.1. Upload images



5.2 Pneumonia Detection



6. Conclusion

In this project, I applied CNN which was built from scratch and also transfer learning using VGG16 to the binary classification problem of determining which of two classes ‘ normal’

or ‘pneumonia’ a chest x-ray falls under and used GCP AUTO ML Vision to build and deploy web app for pneumonia detection.

The study was limited by depth of data. With increased access to data and training of the model with radiological data from patients and nonpatients in different parts of the world, significant improvements can be made. The lack of sufficient numbers of medical images seems to be a common problem for other medical issues besides pneumonia.

Follow-on work to extend this application could include using several other ‘Transfer Learning’ techniques other than VGG16 and hyperparameters.

- Improve baseline model accuracy with different approaches
    - Increase the number of convolutional layers
    - Change the predetermined values (epochs, batch, size, callbacks…)
- Increase the number of data
    - Using data augmentation techniques
    - Different DLGAN models
- Train the data using different Transfer Learning models.

Although this project is far from complete, it is remarkable to see the success of deep learning in such varied real-world problems.