Image Classification using Transfer Learning technique of Convolutional Neural Networks

Mohan Burugupalli

Adviser: Dr. Simone Ludwig

Computer Science Department

North Dakota State University

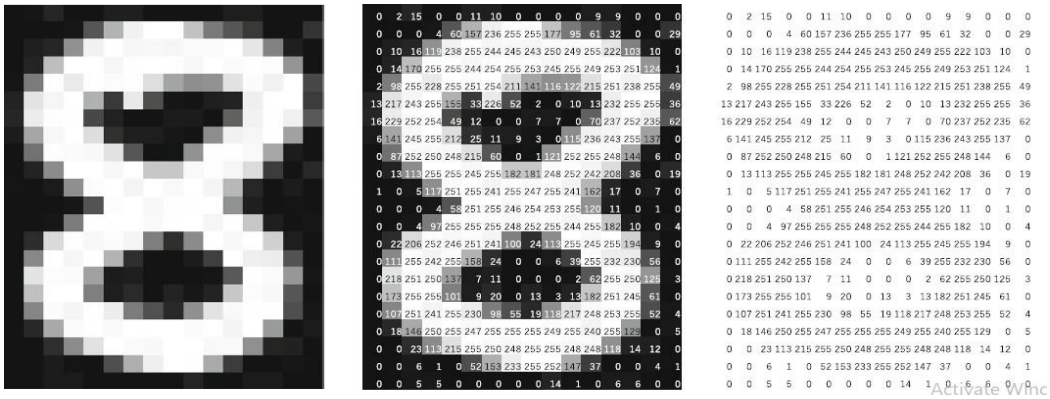
mohan.burugupalli@ndsu.edu

Introduction

Image classification is of increasing importance recently. Using conventional model, we need lots of data to be trained. Although, there is ImageNet dataset with millions of data available, we may not find data for all domains. In this scenario we can use the Transfer Learning technique which uses pre-trained Convolutional Neural Networks on new data. The idea of Transfer Learning is that, suppose if we must classify task1 images but do not have enough data to train the neural network. Find a related task2 that has vast data and train the neural network on it. Then use the whole model or few layers as desired from the task2 to solve task1. I have used top six best pre-trained networks with three types of techniques each to classify medical images and captured their accuracies for comparison.

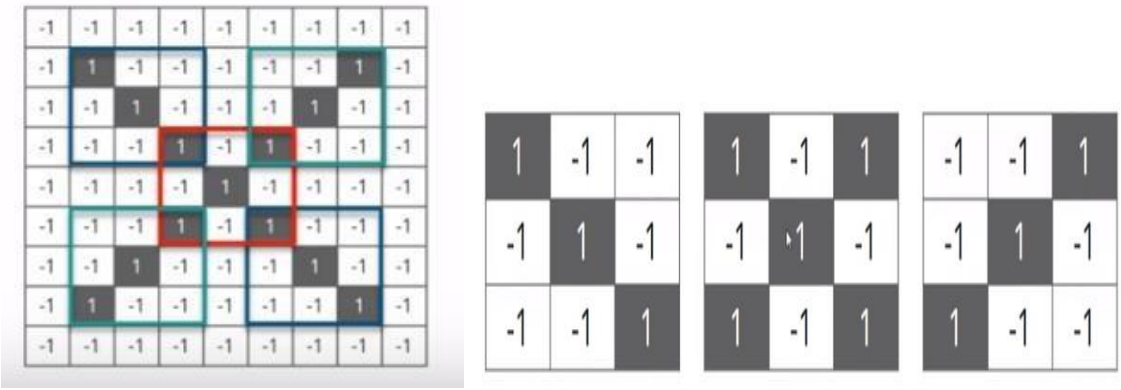
Background

Convolution Neural Networks is a concept that has multiple layers for training the data and when a test data is fed to it, the image is classified. Below is the image how computer sees an image.

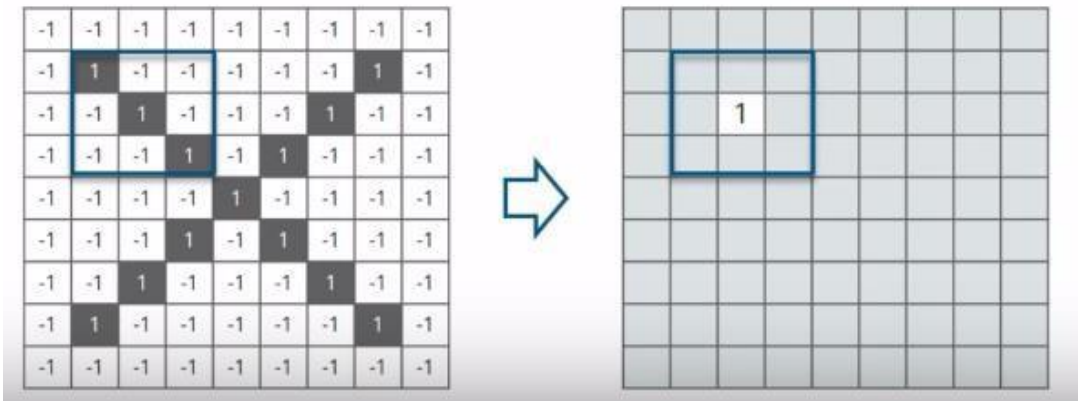


Convolution Neural Networks has the layers as Convolution layer, ReLU activation layer, Pooling layer and Fully connected layer.

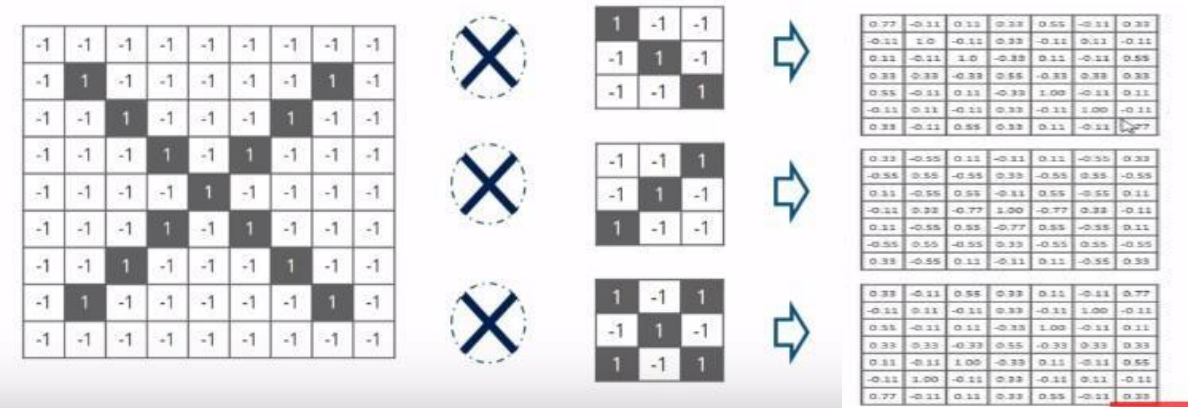
Smallest unique features of the input image are taken for training. Below is an example where small pieces/features of bigger image are taken for convolution.



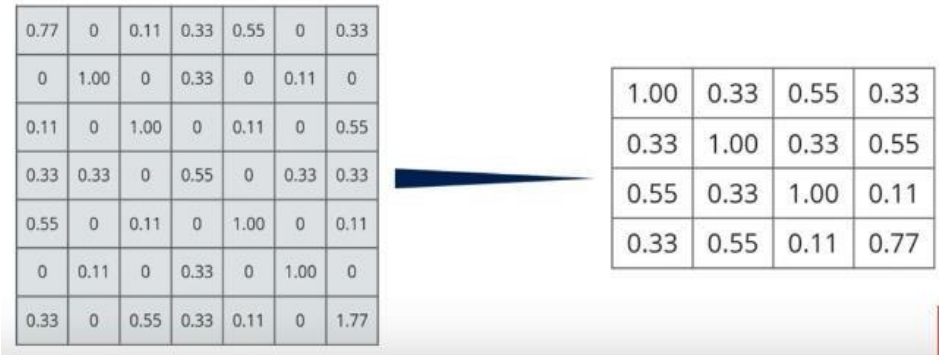
Move each feature on the big image and perform matrix calculation of the values in the feature and the position places in big image. Average all the values in the resulting 3*3 matrix and the result is taken as the final convolution value of the image for the given feature



The result of all three features convoluted over the given image is as below



Then pass through ReLU activation layer and max pooling layer. The resulting values are as below



0.77	0	0.11	0.33	0.55	0	0.33
0	1.00	0	0.33	0	0.11	0
0.11	0	1.00	0	0.11	0	0.55
0.33	0.33	0	0.55	0	0.33	0.33
0.55	0	0.11	0	1.00	0	0.11
0	0.11	0	0.33	0	1.00	0
0.33	0	0.55	0.33	0.11	0	1.77

1.00	0.33	0.55	0.33
0.33	1.00	0.33	0.55
0.55	0.33	1.00	0.11
0.33	0.55	0.11	0.77

After performing one more instance of convolution, ReLU and max pooling on the above result we get the numerical notation of the input image as below.

For testing, we will generate the numerical notation of the image that is being tested and compare its values with the existing values and classify the image accordingly.

Experiments on Datasets

'*chest_xray_fulldataset*' datafile has test, train and validation data in test, train and val subfolders. Each category(test/train/val) of data folder has NORMAL/PNEUMONIA subfolders that has the respective images.

Step-1 : Import all necessary libraries

Step-2: Initialize a 2d array, 'acc_data' to store the Image classification accuracies. Rows indicate the methodology and columns indicate the pre-trained networks.

Step-3: Chose the desired method of classification. Let us choose 3 and Enter.

```
Enter desired Image classification methodology from below
 1.Transfer Learning by Feature Extraction
 2.Transfer Learning by Feature Extraction and Image Augmentation
 3.Transfer Learning with Fine-tuning and Image Augmentation
3
```

Step-4: Chose the desired Neural Network. Let me chose 'VGG16' and Enter.

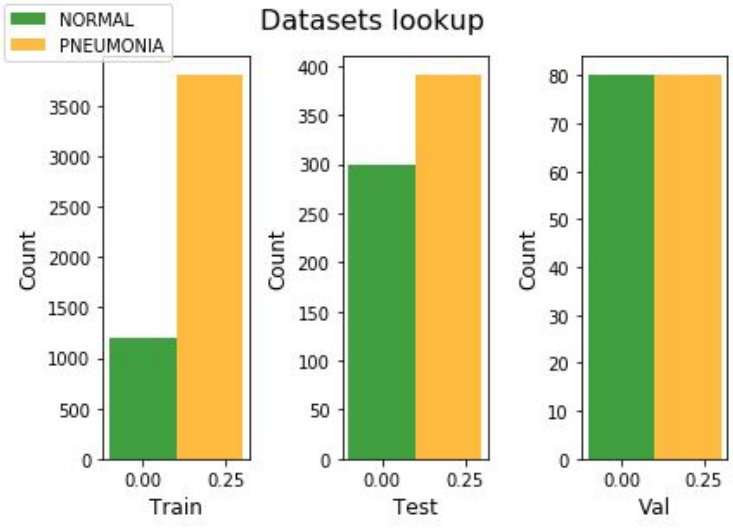
```
Step4 started

Enter the desired Neural Network from below
 1.VGG16         2.VGG19         3.ResNet50      4.InceptionV3   5.Xception      6.DenseNet
1
```

Step-5 Create train, test and validation lists. Append the images locations to these lists according to labels. Ex: x_train contains all NORMAL Images from training dataset and y_train contains all PNEUMONIA images of training dataset

Step-6: Resize and convert the lists to numpy array.

Step-7: Plot the datasets count



Step-8: Merge all training files into one folder namely training_data, testing files to test_data and validation files to validation_data respectively.

Step-9: Resize and convert the images to array using the python inbuilt library 'array_to_img'. Store the arrays in separate lists for training and validation. Split the labels and store in labels list.
Training dataset shape and Validation dataset shape now is as below

```
Training dataset shape: (5000, 150, 150, 3)     Validation dataset shape: (80, 150, 150, 3)
```

Step-10: Scale the images and transform the labels using 'LabelEncoder' library

```
['PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'PNEUMONIA', 'NORMAL',
'PNEUMONIA'] [1 1 1 1 1 1 1 1 0 1]
```

Step-11: Import and Load the chosen model with 'imagenet' as weights

Step-12: We have three methodologies of classification
<u>Method1: CNN as feature extractor</u>

a) Freeze all the convolution layers and use the pre-trained network as just for feature extraction of images.

```
for layer in cnn_model.layers:
    layer.trainable = False
```

b) The output would be like shown in next page. It shows all the layers are frozen as expected. The last activation feature map is the output from 'block5_pool' that gives the bottleneck features. All the training and validation datasets features are extracted, flattened and fed as input to classifier.

```
def get_bottleneck_features(model, input_imgs):
    features = model.predict(input_imgs, verbose=0)
    return features

train_features_cnn = get_bottleneck_features(cnn_model, train_imgs_scaled)
validation_features_cnn = get_bottleneck_features(cnn_model, validation_imgs_scaled)
```

	Layer Type	Layer Name	Layer Trainable
0	<keras.engine.topology.InputLayer object at 0x7f26c86b2518>	input_1	False
1	<keras.layers.convolutional.Conv2D object at 0x7f277c9fc080>	block1_conv1	False
2	<keras.layers.convolutional.Conv2D object at 0x7f26c86b26d8>	block1_conv2	False
3	<keras.layers.pooling.MaxPooling2D object at 0x7f26c86e6c88>	block1_pool	False
4	<keras.layers.convolutional.Conv2D object at 0x7f26c867dc18>	block2_conv1	False
5	<keras.layers.convolutional.Conv2D object at 0x7f26c8690f28>	block2_conv2	False
6	<keras.layers.pooling.MaxPooling2D object at 0x7f26c869e5c0>	block2_pool	False
7	<keras.layers.convolutional.Conv2D object at 0x7f26c863f828>	block3_conv1	False
8	<keras.layers.convolutional.Conv2D object at 0x7f26c863f128>	block3_conv2	False
9	<keras.layers.convolutional.Conv2D object at 0x7f26c86607b8>	block3_conv3	False
10	<keras.layers.pooling.MaxPooling2D object at 0x7f26c83d7d68>	block3_pool	False
11	<keras.layers.convolutional.Conv2D object at 0x7f26c83fd358>	block4_conv1	False
12	<keras.layers.convolutional.Conv2D object at 0x7f26c83fddd8>	block4_conv2	False
13	<keras.layers.convolutional.Conv2D object at 0x7f26c839da20>	block4_conv3	False
14	<keras.layers.pooling.MaxPooling2D object at 0x7f26c83ac1d0>	block4_pool	False
15	<keras.layers.convolutional.Conv2D object at 0x7f26c834e978>	block5_conv1	False
16	<keras.layers.convolutional.Conv2D object at 0x7f271a15eb38>	block5_conv2	False
17	<keras.layers.convolutional.Conv2D object at 0x7f26c8371d68>	block5_conv3	False
18	<keras.layers.pooling.MaxPooling2D object at 0x7f26c8314b00>	block5_pool	False
19	<keras.layers.core.Flatten object at 0x7f26c828bda0>	flatten_1	False

c) Build a classifier that takes the flattened bottleneck features as input and performs classification.

```
input_shape = cnn_model.output_shape[1]

model = Sequential()
model.add(InputLayer(input_shape=(input_shape,)))
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['accuracy'])
```

d) Train the model using ‘model.fit’ selecting the number of epochs, batch_size, training data and validation data.

e) Save the model to use for classifying the testing images

Method2: CNN as feature extractor with Image Augmentation

Image Augmentation : As the training data is limited, we use Image Augmentation technique. Using this we transform the existing images from training dataset by performing some operations such as rotation, shearing, flipping, rescaling etc. Now, we have more images available for training and feed them to our model that increases accuracy of image detection.

a) Perform Image augmentation on training and validation datasets
b) We keep the layers of CNN network frozen and generate the Image data using the python library 'ImageDataGenertor'
c) Build the training model on data generators rather on bottleneck features as in Method1.

```
model = Sequential()
model.add(cnn_model)
model.add(Dense(512, activation='relu', input_dim=input_shape))
model.add(Dropout(0.3))
model.add(Dense(512, activation='relu'))
model.add(Dropout(0.3))
model.add(Dense(1, activation='sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer=optimizers.RMSprop(lr=2e-5),
              metrics=['accuracy'])
```

d) Save the model as h5 file to be used for classifying training data.

```
model.save('chest_xray_fulldataset/NORMAL_PNEUMONIA_tlearn_img_aug_cnn.h5')
```

Method3: CNN with Fine-tuning and Image Augmentation

Fine-tuning: When we have option to frozen the layers, fine-tuning is one concept where we unfreeze bottom n layers as per requirement and allow the unfreeze layers to get the weights updated after each epoch. As after each epoch model iproves, this method delivers best results with more number of epochs.

a) Unfreeze last two layers – 'block5_conv1' and 'block4_conv1'

```
for layer in cnn_model.layers:
    if layer.name in ['block5_conv1', 'block4_conv1']:
        set_trainable = True
    if set_trainable:
        layer.trainable = True
    else:
        layer.trainable = False
```

b) The below image shows that the block4 and block5 are now trainable backwards after each epoch.

	Layer Type	Layer Name	Layer Trainable
0	<keras.engine.topology.InputLayer object at 0x7f26c86b2518>	input_1	False
1	<keras.layers.convolutional.Conv2D object at 0x7f277c9fc080>	block1_conv1	False
2	<keras.layers.convolutional.Conv2D object at 0x7f26c86b26d8>	block1_conv2	False
3	<keras.layers.pooling.MaxPooling2D object at 0x7f26c86e6c88>	block1_pool	False
4	<keras.layers.convolutional.Conv2D object at 0x7f26c867dc18>	block2_conv1	False
5	<keras.layers.convolutional.Conv2D object at 0x7f26c8690f28>	block2_conv2	False
6	<keras.layers.pooling.MaxPooling2D object at 0x7f26c869e5c0>	block2_pool	False
7	<keras.layers.convolutional.Conv2D object at 0x7f26c863f828>	block3_conv1	False
8	<keras.layers.convolutional.Conv2D object at 0x7f26c863f128>	block3_conv2	False
9	<keras.layers.convolutional.Conv2D object at 0x7f26c86607b8>	block3_conv3	False
10	<keras.layers.pooling.MaxPooling2D object at 0x7f26c83d7d68>	block3_pool	False
11	<keras.layers.convolutional.Conv2D object at 0x7f26c83fd358>	block4_conv1	True
12	<keras.layers.convolutional.Conv2D object at 0x7f26c83fddd8>	block4_conv2	True
13	<keras.layers.convolutional.Conv2D object at 0x7f26c839da20>	block4_conv3	True
14	<keras.layers.pooling.MaxPooling2D object at 0x7f26c83ac1d0>	block4_pool	True
15	<keras.layers.convolutional.Conv2D object at 0x7f26c834e978>	block5_conv1	True
16	<keras.layers.convolutional.Conv2D object at 0x7f271a15eb38>	block5_conv2	True
17	<keras.layers.convolutional.Conv2D object at 0x7f26c8371d68>	block5_conv3	True
18	<keras.layers.pooling.MaxPooling2D object at 0x7f26c8314b00>	block5_pool	True
19	<keras.layers.core.Flatten object at 0x7f26c828bda0>	flatten_1	True

c) Use the same model with Image generators as in Method2 to train the model.

d) Save the model for classifying the testing data.

```
model.save('chest_xray_fulldataset/NORMAL_PNEUMONIA_tlearn_finetune_img_aug_cnn.h5')
```

Step-13: Plot the changes in validation accuracy after each epoch

Step-14: Write a function to class to num label transformation and vice-versa.

```
num2class_label_transformer = lambda l: ['NORMAL' if x == 0 else 'PNEUMONIA' for x in l]
class2num_label_transformer = lambda l: [0 if x == 'NORMAL' else 1 for x in l]
```

Step-15: Load the test images. Scale the images , split the labels and save to separate lists

Step-16: Load the '.h5' file and input the test data to get the classification of data.
Step-17: Choose random test images and show their actual and predicted labels.

Step-18: Plot confusion matrix that shows the True Normal and True Pneumonia classification.

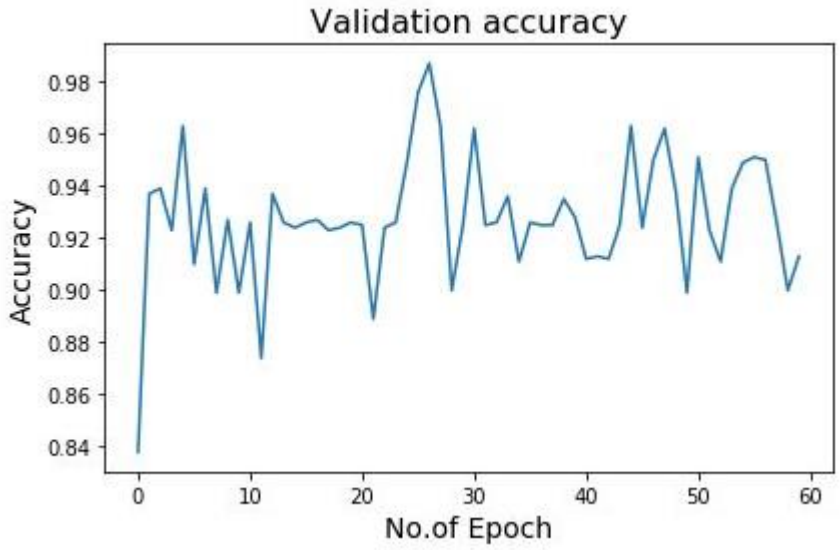
Step-19: Calculate accuracy.

Step-20: Created a 2d array with all the accuracies updated to it. Visualize it as a DataFrame and plot bar diagram of all accuracies for each pre-trained networks.

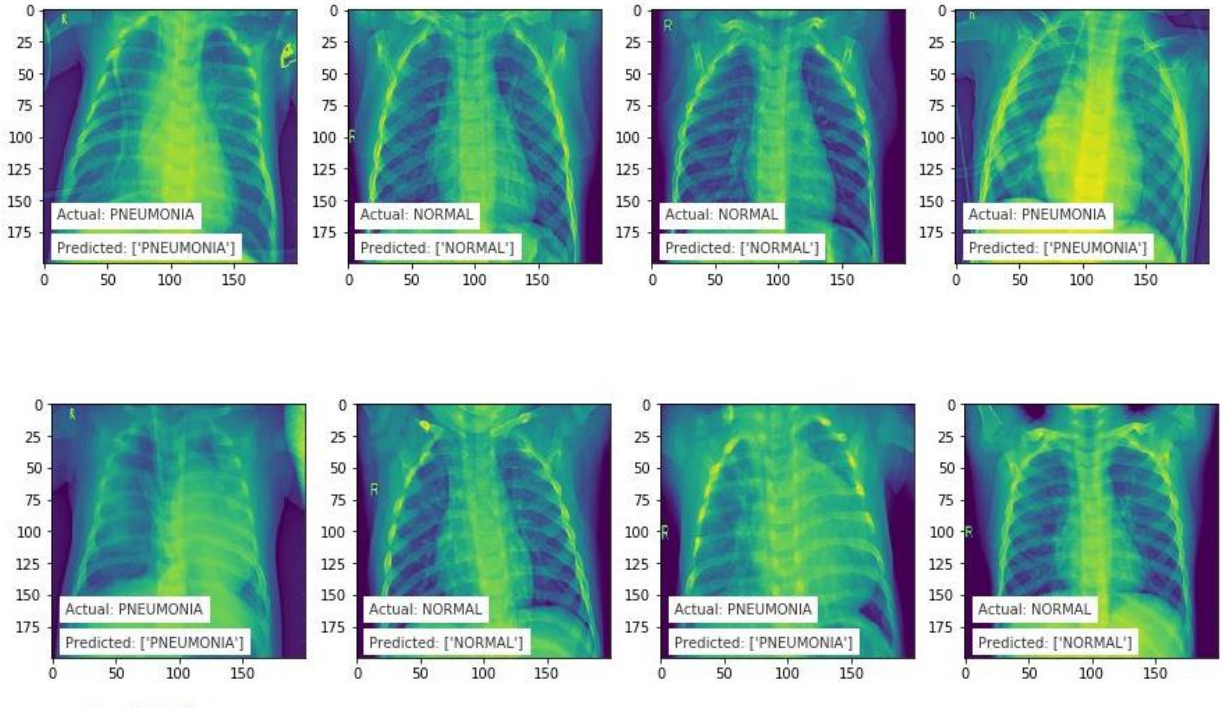
Results

VGG16 model with Image Augmentation and Fine-Tuning method

Validation accuracy changes with each epoch : Validation accuracy is maintained as 92% on an average for all epochs.

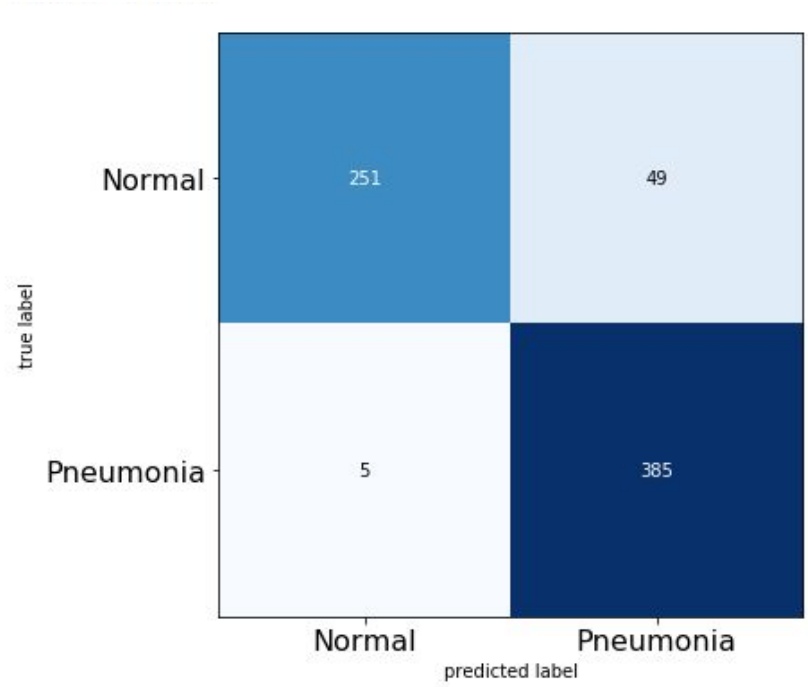


Random Test images classification: Random generation of test images and their classification is as below.



Confusion Matrix to show True Normal and True Pneumonia

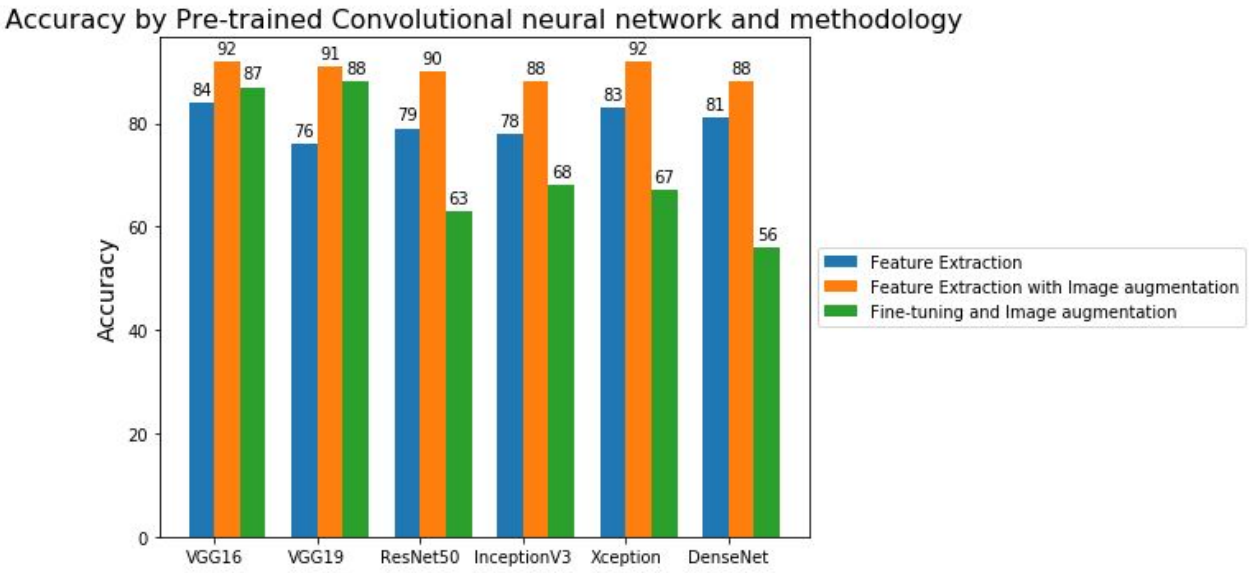
Majority of True Normal and True Pneumonia are classified well.



DataFrame of all accuracies

	conv method	Image augm	Fine Tuning
VGG16	84.750	92.150	87.17
VGG19	76.118	91.420	88.52
ResNet50	79.820	90.110	63.17
InceptionV3	78.217	88.521	68.04
Xception	83.594	92.570	67.46
DenseNet	81.560	88.980	56.66

Bar-plot of accuracies



Conclusion

In all the pre-trained networks used, Feature extraction with Image augmentation method delivered the best results. Of all the networks 'Xception' model with Image Augmentation as feature extraction method returned the highest accuracy of 92.5% followed by VGG16, VGG19, ResNet 50, InceptionV3 and DenseNet201 with 92.1%, 91.42%, 90%, 88% and 88% respectively.

Future Work

Improving accuracy. Classify multiple categories of data and further extend it to classify and recognize objects/images in videos.

References

1. **https://machinelearningmastery.com/how-to-use-transfer-learning-when-developing-convolutional-neural-network-models/**
2. **https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751**
3. **https://www.tensorflow.org/tutorials/images/transfer_learning**
4. **https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html**
5. **https://keras.io/api/applications/**
6. **https://medium.com/@ODSC/how-to-leverage-pre-trained-layers-in-image-classification-31fb9b8cdd0**
7. **https://www.kdnuggets.com/2019/11/transfer-learning-coding.html**
8. **https://www.machinelearningplus.com/plots/top-50-matplotlib-visualizations-the-master-plots-python/**
9. **https://machinelearningmastery.com/how-to-improve-performance-with-transfer-learning-for-deep-learning-neural-networks/**
10. **https://ruder.io/transfer-learning/**
11. **https://www.edureka.co/blog/deep-learning-with-python/**