Lung Inflammation Detection

Ashutosh Kaushik (ICM2017002), Arya Krishnan (ICM2017501), Priyal Gupta (IHM2017004)

Under the supervision of
Dr. Pavan Chakraborty

Under the guidance of
Snigdha Sen

Problem statement :

Lung inflammation Detection using different Convolutional neural network architectures and a comparison of their performances. Presenting a qualitative analysis of the limitations and advantages of each of them in the field of medical imaging.

1. Dataset used

We will be using the Kaggle dataset **Chest X-Ray Images** (Pneumonia). The dataset is organized into 3 folders (train, test, val) and contains subfolders for each image category (Pneumonia/Normal). There are over 5,000 X-Ray images (JPEG) and 2 categories (Pneumonia/Normal).

5216 images belonging to 2 classes in the training set.

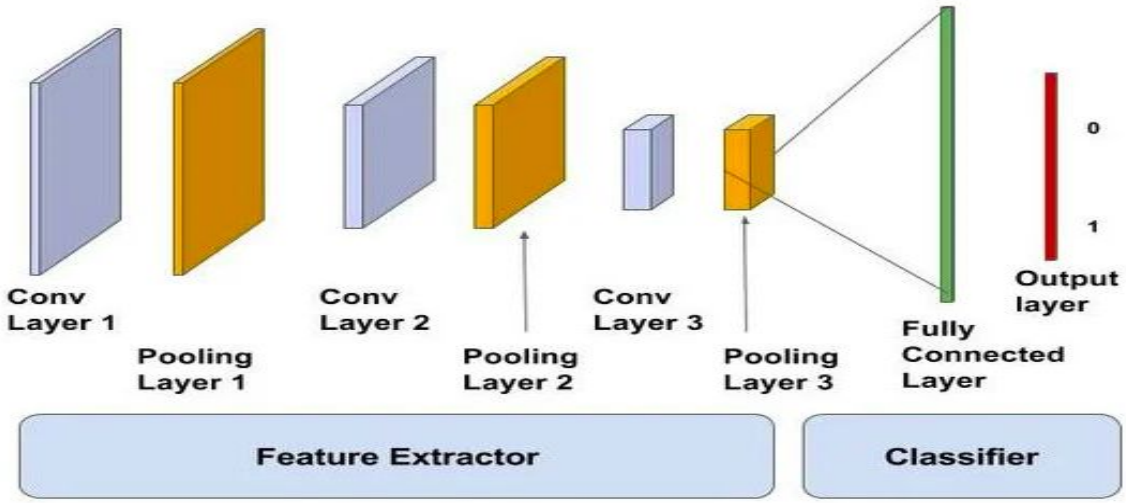
16 images belonging to 2 classes in the validation set.

624 images belonging to 2 classes in the test set.

2. Models Trained

2.1. CNN

Convolutional Neural Networks are a type of feedforward neural network. The network consists of two parts. The first part of the network consists of convolutional and max-pooling layers which act as feature extractor. The second part consists of the fully connected layer which performs non-linear transformations of the extracted features and acts as the classifier.



Convolutional Neural Network

The pneumonia dataset we're using contains three sets of images:

1. The training set. Images we're going to train the neural network on.
2. The validation set. Images we're going to use to check if the model is underfitting or overfitting.
3. The test set. These are images we're going to use to check how good our neural network is with data it has not seen before.

Model-1: Simple CNN

- Default initialization
- No Padding
- No Dropout, No Batch Normalization

Here is a summary of the Simple CNN model we trained and tested on TensorFlow using **Adam Optimizer** for 10 epochs.

Preprocessing and data augmentation has been performed.
For our training and validation sets, we will zoom the image randomly by a factor of 0.2, Rescale pixel values to 0 to1, randomly flip half the images horizontally and vertically, and apply shear based transformations randomly.
We will run for 326 steps per epoch with a batch size of 16.
We will then evaluate on a test set of 624 images.

Layer (type)	Output Shape	Param #
conv2d_3 (**Conv2D**)	(None, 62, 62, 32)	896
max_pooling2d_3 (**MaxPooling2**)	(None, 31, 31, 32)	0
conv2d_4 (**Conv2D**)	(None, 29, 29, 32)	9248
max_pooling2d_4 (**MaxPooling2**)	(None, 14, 14, 32)	0
flatten_2 (**Flatten**)	(None, 6272)	0
dense_3 (**Dense**)	(None, 128)	802944
dense_4 (**Dense**)	(None, 1)	129

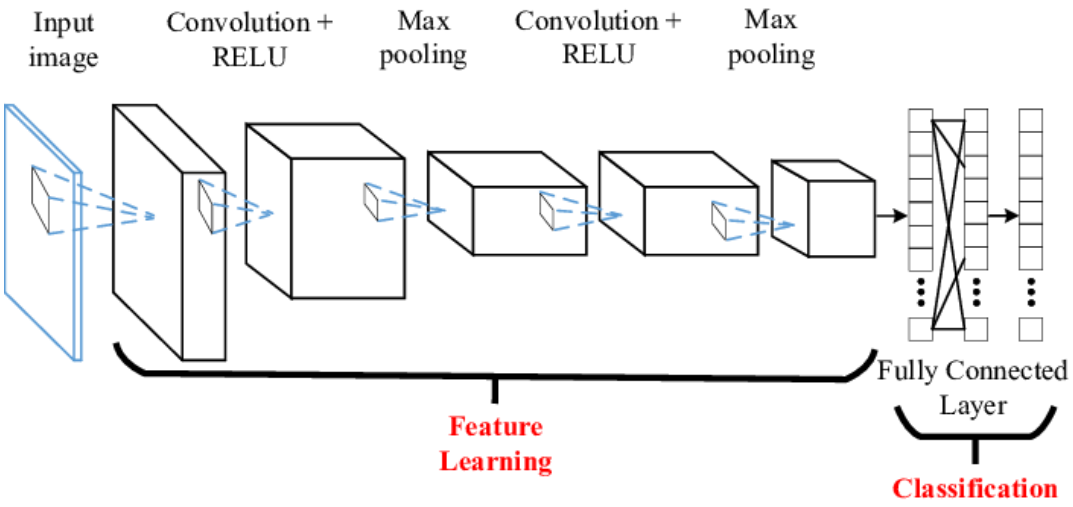
==========

Total params: 813,217

Trainable params: 813,217

Non-trainable params: 0

_____



Model-2: Modified CNN

- ReLU Activation
- He initialization
- Padding added of size 1 on each side
- ADAM optimizer
- Dropout + Batch Normalization

Batch Normalization :- It is used to increase the stability of a neural network. It normalises the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. Batch Normalisation, basically allows each layer of a network to learn by itself a little bit more independently of other layers.

Dropout :- It is a technique used to prevent a model from overfitting. Dropout works by randomly setting the outgoing edges of hidden units (neurons that make up hidden

layers) to 0 at each update of the training phase. It's computationally not too heavy and effective, nodes are dropped layer wise. It also helps in thinning and reducing the capacity of the network and ultimately producing better results.

He initialization :- This is an improved version of Random initialization (which was to assign weights randomly) where we initialize layer wise, the range of value of the initialization depends on the size of the previous layer directly. It has better results as compared to zero or random initialization.

Since dropout allows training for more epochs without overfitting we run the training through 652 steps per epoch and with a batch size of 32.
Preprocessing and data augmentation has been performed.
For our training and validation sets, we will zoom the image randomly by a factor of 0.2, Rescale pixel values to 0 to1, randomly flip half the images horizontally and vertically, and apply shear based transformations randomly.
Here is a new architecture with improvements to the previous architecture, run for 10 epochs. The preprocessing used was rescaling and some data augmentations provided by Keras-

Layer (type)	Output Shape	Param #
zero_padding2d_1 (ZeroPadding)	(None, 66, 66, 3)	0
conv2d_1 (**Conv2D**)	(None, 64, 64, 64)	1792
max_pooling2d_1 (**MaxPooling2**)	(None, 32, 32, 64)	0
dropout_1 (Dropout)	(None, 32, 32, 64)	0
zero_padding2d_2 (ZeroPadding)	(None, 34, 34, 64)	0
conv2d_2 (**Conv2D**)	(None, 32, 32, 64)	36928
dropout_2 (Dropout)	(None, 32, 32, 64)	0

flatten_1 (**Flatten**)	(None, 65536)	0
dense_1 (**Dense**)	(None, 256)	16777472
batch_normalization_1 (Batch	(None, 256)	1024
dropout_3 (Dropout)	(None, 256)	0
dense_2 (**Dense**)	(None, 128)	32896
batch_normalization_2 (Batch	(None, 128)	512
dropout_4 (Dropout)	(None, 128)	0
dense_3 (**Dense**)	(None, 1)	129

Total params: 16,850,753
Trainable params: 16,849,985
Non-trainable params: 768

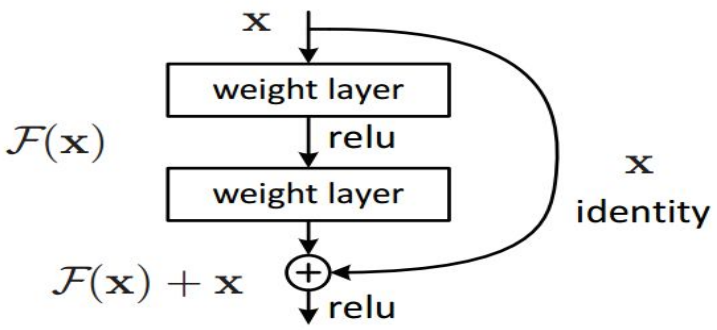
Model-3: ResNet50

RESNET50 : - ResNet50, short for Residual Networks - 50, is a classic neural network used as a backbone for many computer vision tasks.

Deep networks are hard to train because of the notorious vanishing gradient problem — as the gradient is back-propagated to earlier layers, repeated multiplication may make the gradient extremely small. As a result, as the network goes deeper, its performance gets saturated or even starts degrading rapidly.

To solve this problem **skip connection** is introduced. In skip connection other than stacking convolutional layers we also add the original input to the output to the convolutional block.

A residual block is when the activation is fed directly to a deeper layer skipping over the block.

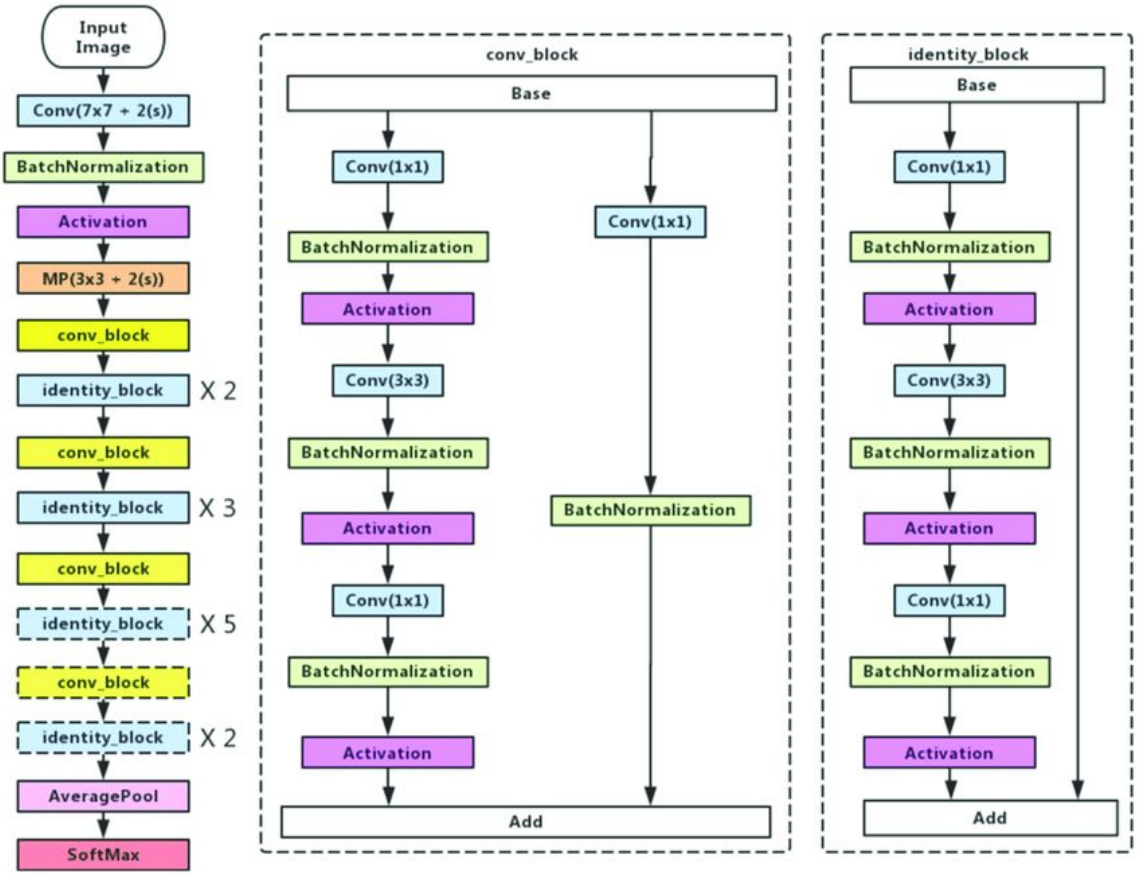


This is a residual block.

- ResNet50 is often used in **Transfer Learning** in the medical field, pretrained on ImageNet as a feature extractor.
- It is then fine tuned using the **Stochastic Gradient Descent optimizer** with a learning rate of 0.0001 to fit the given data distribution.
- We have also added Pooling, Dense and a sigmoid activation layer for the purpose of **binary classification**.

Preprocessing and data augmentation has been performed.
For our training and validation sets, we will zoom the image randomly by a factor of 0.2, Rescale pixel values to 0 to1, randomly flip half the images horizontally and vertically, and apply shear based transformations randomly.
We will run it for 326 steps per epoch and with a batch size of 16 and for 16 epochs since it is a relatively large model.

RESNET50 architecture-



Taken from: Ji, Qingge & Huang, Jie & He, Wenjie & Sun, Yankui. (2019). Optimized Deep Convolutional Neural Networks for Identification of Macular Diseases from Optical Coherence Tomography Images. Algorithms. 12. 51. 10.3390/a12030051.

We will add **flatten, dense (or fully connected), dropout and sigmoid activation** layers for our classifier utilizing RESNET50 classification and modify the architecture to:

Layer (type)	Output Shape	Param #
resnet50 (Model)	(None, None, None, 2048)	23587712
global_average_pooling2d_1	(None, 2048)	0
dense_1 (Dense)	(None, 512)	1049088

_____

dense_2 (Dense) (None, 2) 1026

==========

Total params: 24,637,826

Trainable params: 24,584,706

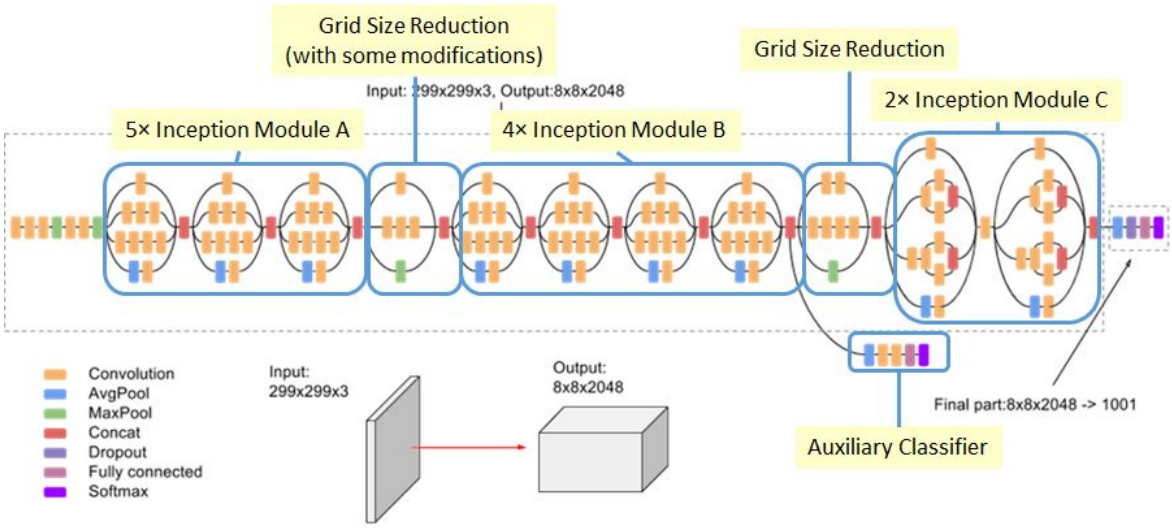
Non-trainable params: 53,120

_____

Model-4: InceptionV3

Inception V3-The Inception Network uses Inception layers which parallelly run different sizes of filters on the image and lets each layer pick which filter size learns best. Inception V3(version 3) is improved from v2, choosing the best kernel size is way more difficult for varying pictures, and an earlier network was made deep to get the best CNN which was computation heavy. Inception v1 is a new idea in which instead of using a single kernel size we use different (which are more suitable) sizes which make the network wider instead of making it deep and ultimately helping us in improving the overall performance.

- Inception V3 is often used in **Transfer Learning** in the medical field, pretrained on ImageNet as a feature extractor.
- It is then fine tuned using the **Stochastic Gradient Descent optimizer** with a learning rate of 0.0001 to fit the given data distribution.
- We have also added Pooling, Dense and a sigmoid activation layer for the purpose of **binary classification**.



Inception-v3 Architecture (Batch Norm and ReLU are used after Conv)

Preprocessing and data augmentation has been performed.

For our training and validation sets, we will zoom the image randomly by a factor of 0.2, Rescale pixel values to 0 to1, randomly flip half the images horizontally and vertically, and apply shear based transformations randomly.

We will run for 326 steps per epoch with a batch size of 16 and for 16 epochs since it is a relatively large model.

We will add **dense (or fully connected) and sigmoid activation** layers for our classifier utilizing InceptionV3 classification and modify the architecture to:

Layer (type)	Output Shape	Param #
inception_v3 (Model)	(None, None, None, 2048)	21802784
global_average_pooling2d_1 (	(None, 2048)	0
dense_1 (Dense)	(None, 512)	1049088

dense_2 (Dense) (None, 2) 1026

==========

Total params: 22,852,898

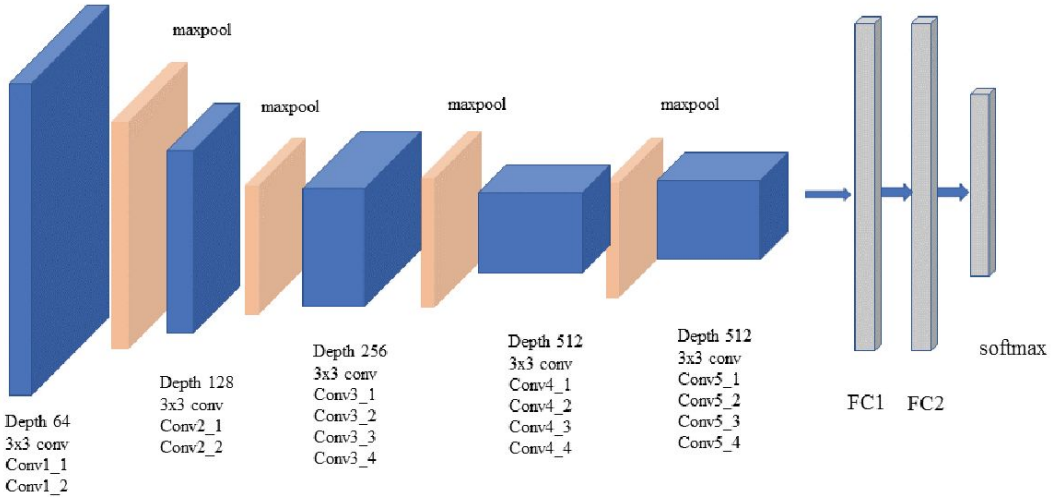
Trainable params: 22,818,466

Non-trainable params: 34,432

_____

Model-5: VGG16

VGG16- Visual geometry group 16 (VGG 16) is a convolution neural network architecture, 16 stands for the number of layers having weights. It is a highly dense network with around 140,000,000 parameters. It does not work on a large number of hyperparameters, instead it contains the same convolution 3x3 layer with the stride length 1, and the exact same other parameters such maxpool layer and padding. Finally we have 2 fully connected layers having the output of softmax function. This VGG was proposed by K. Simonyan and A. Zisserman from the University of Oxford in the paper “Very Deep Convolutional Networks for Large-Scale Image Recognition”. Below is the image attached depicting the above explained architecture.



- VGG16 is often used in **Transfer Learning** in the medical field, pretrained on ImageNet as a feature extractor.
- It is then fine tuned using the **Stochastic Gradient Descent optimizer** with a learning rate of 0.0001 to fit the given data distribution.
- We have also added Pooling, Dense and a sigmoid activation layer for the purpose of **binary classification**.

Preprocessing and data augmentation has been performed.
For our training and validation sets, we will zoom the image randomly by a factor of 0.2, Rescale pixel values to 0 to1, randomly flip half the images horizontally and vertically, and apply shear based transformations randomly.
We will run for 326 steps per epoch with a batch size of 16 and for 16 epochs since it is a relatively large model.

Layer (type)	Output Shape	Param #
vgg16 (Model)	(None, None, None, 512)	14714688
global_average_pooling2d_3	(None, 512)	0
dense_5 (Dense)	(None, 512)	262656
dense_6 (Dense)	(None, 2)	1026

Total params: 14,978,370
Trainable params: 14,978,370
Non-trainable params: 0

3. Results - The networks are evaluated on their Accuracy measure and Binary Cross-Entropy loss.

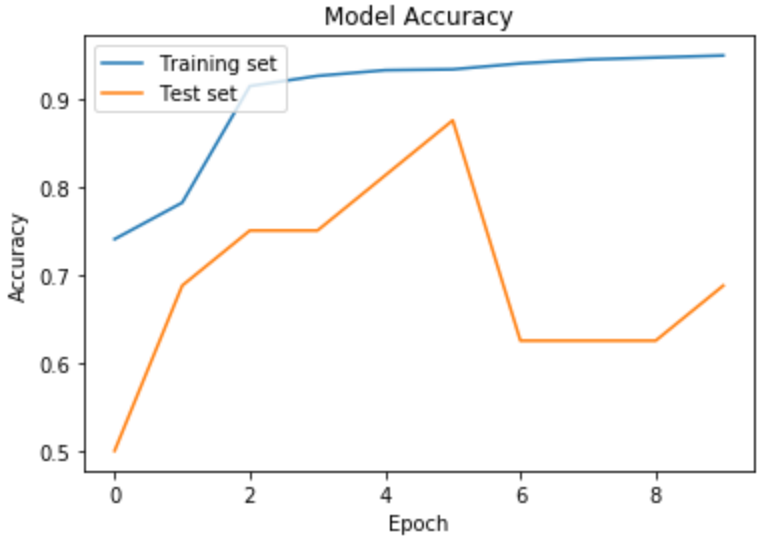
Model-1:

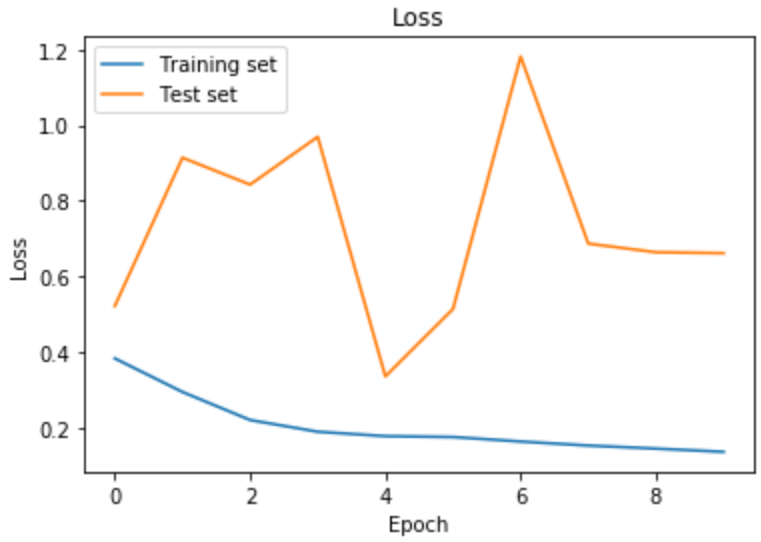
```
Epoch 1/10
326/326 [==============================] - 155s 477ms/step - loss: 0.3828 - acc: 0.7402 - val_loss: 0.5216 - val_acc: 0.5000
Epoch 2/10
326/326 [==============================] - 133s 409ms/step - loss: 0.2940 - acc: 0.7816 - val_loss: 0.9141 - val_acc: 0.6875
Epoch 3/10
326/326 [==============================] - 132s 405ms/step - loss: 0.2198 - acc: 0.9139 - val_loss: 0.8429 - val_acc: 0.7500
Epoch 4/10
326/326 [==============================] - 128s 393ms/step - loss: 0.1887 - acc: 0.9254 - val_loss: 0.9695 - val_acc: 0.7500
Epoch 5/10
326/326 [==============================] - 129s 395ms/step - loss: 0.1771 - acc: 0.9319 - val_loss: 0.3353 - val_acc: 0.8125
Epoch 6/10
326/326 [==============================] - 127s 391ms/step - loss: 0.1747 - acc: 0.9329 - val_loss: 0.5137 - val_acc: 0.8750
Epoch 7/10
326/326 [==============================] - 128s 392ms/step - loss: 0.1628 - acc: 0.9396 - val_loss: 1.1821 - val_acc: 0.6250
Epoch 8/10
326/326 [==============================] - 128s 393ms/step - loss: 0.1522 - acc: 0.9440 - val_loss: 0.6867 - val_acc: 0.6250
Epoch 9/10
326/326 [==============================] - 128s 394ms/step - loss: 0.1441 - acc: 0.9463 - val_loss: 0.6638 - val_acc: 0.6250
Epoch 10/10
326/326 [==============================] - 132s 405ms/step - loss: 0.1356 - acc: 0.9486 - val_loss: 0.6614 - val_acc: 0.6875
```

Best Validation accuracy epoch-

Epoch 6/10

loss: 0.1747 - acc: 0.9329 - val_loss: 0.5137 - val_acc: 0.8750





Tested on a blind test set of 624 images-

[9]:
```
scores = model.evaluate_generator(test_generator)
print('acc =',scores[1]*100)
```

```
acc = 83.17307692307693
```

[10]:
```
print(scores[0],scores[1])
```

```
0.4904923223150082 0.8317307692307693
```

Test Loss:0.49

Test Accuracy: 83.2%

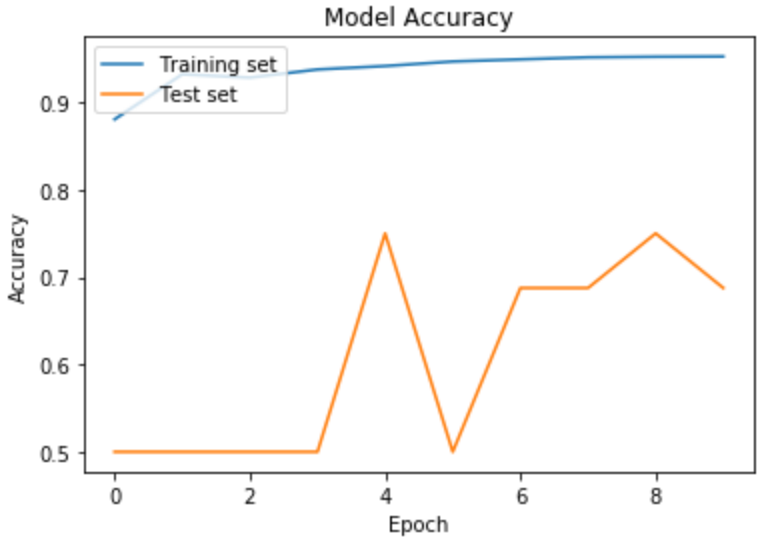
Model-2:

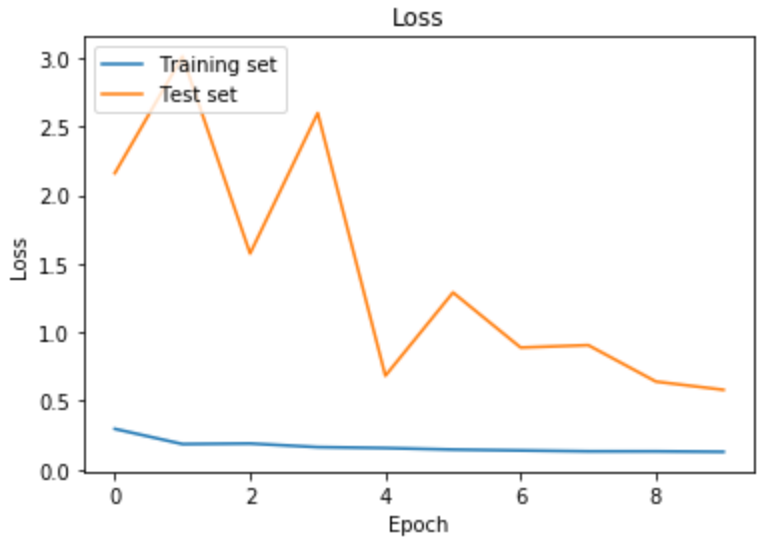
```
Epoch 1/10
652/652 [==============================] - 333s 511ms/step - loss: 0.2962 - acc: 0.8805 - val_loss: 2.1595 - val_acc: 0.5000
Epoch 2/10
652/652 [==============================] - 311s 478ms/step - loss: 0.1855 - acc: 0.9321 - val_loss: 3.0100 - val_acc: 0.5000
Epoch 3/10
652/652 [==============================] - 309s 473ms/step - loss: 0.1894 - acc: 0.9282 - val_loss: 1.5741 - val_acc: 0.5000
Epoch 4/10
652/652 [==============================] - 307s 471ms/step - loss: 0.1632 - acc: 0.9376 - val_loss: 2.5978 - val_acc: 0.5000
Epoch 5/10
652/652 [==============================] - 308s 472ms/step - loss: 0.1566 - acc: 0.9415 - val_loss: 0.6832 - val_acc: 0.7500
Epoch 6/10
652/652 [==============================] - 309s 473ms/step - loss: 0.1452 - acc: 0.9467 - val_loss: 1.2900 - val_acc: 0.5000
Epoch 7/10
652/652 [==============================] - 306s 469ms/step - loss: 0.1389 - acc: 0.9491 - val_loss: 0.8889 - val_acc: 0.6875
Epoch 8/10
652/652 [==============================] - 306s 470ms/step - loss: 0.1326 - acc: 0.9515 - val_loss: 0.9062 - val_acc: 0.6875
Epoch 9/10
652/652 [==============================] - 306s 470ms/step - loss: 0.1324 - acc: 0.9522 - val_loss: 0.6409 - val_acc: 0.7500
Epoch 10/10
652/652 [==============================] - 306s 469ms/step - loss: 0.1290 - acc: 0.9525 - val_loss: 0.5808 - val_acc: 0.6875
```

Best Validation accuracy epoch-

Epoch 9/10

loss: 0.1324 - acc: 0.9522 - val_loss: 0.6409 - val_acc: 0.7500





Tested on a blind test set of 624 images-

```
[11]: model.evaluate_generator(test_set)

Out[11]: [0.2834451022820595, 0.8894230769230769]
```

Test Loss:0.28

Test Accuracy: 88.94%

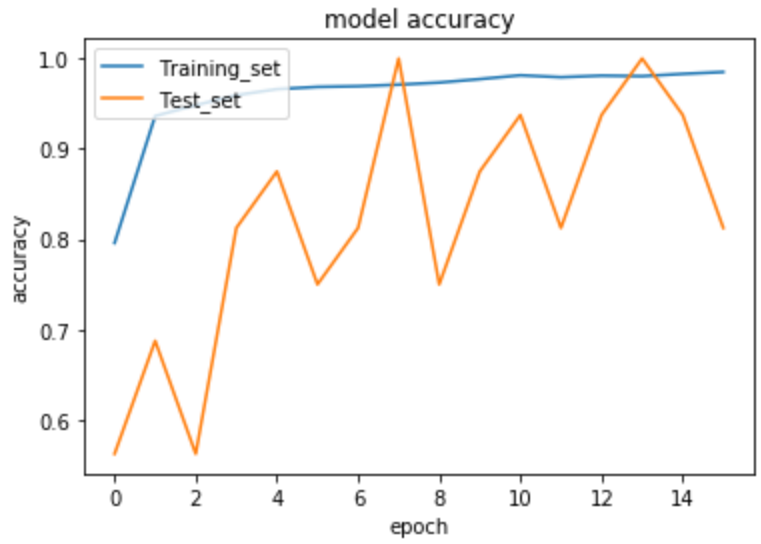
Model-3:

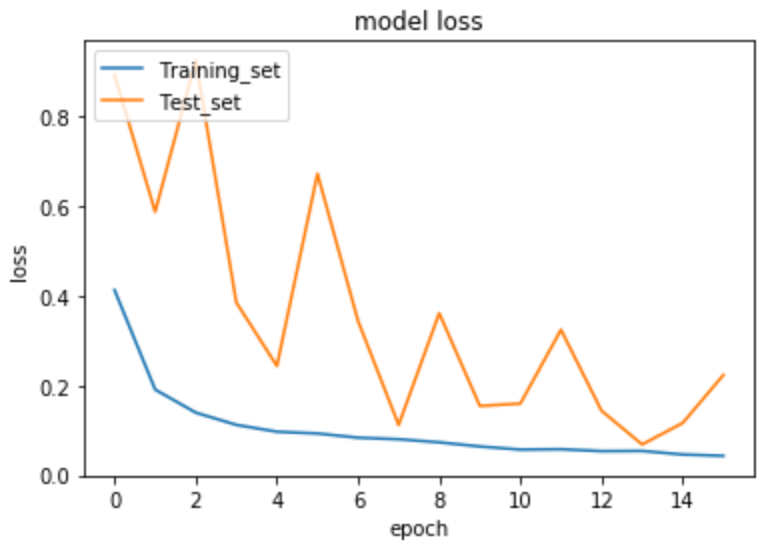
```
Epoch 1/16
326/326 [==============================] - 176s 540ms/step - loss: 0.4121 - acc: 0.7958 - val_loss: 0.8918 - val_acc: 0.5625
Epoch 2/16
326/326 [==============================] - 131s 401ms/step - loss: 0.1906 - acc: 0.9363 - val_loss: 0.5869 - val_acc: 0.6875
Epoch 3/16
326/326 [==============================] - 131s 402ms/step - loss: 0.1392 - acc: 0.9482 - val_loss: 0.9239 - val_acc: 0.5625
Epoch 4/16
326/326 [==============================] - 131s 403ms/step - loss: 0.1119 - acc: 0.9594 - val_loss: 0.3848 - val_acc: 0.8125
Epoch 5/16
326/326 [==============================] - 131s 403ms/step - loss: 0.0967 - acc: 0.9661 - val_loss: 0.2432 - val_acc: 0.8750
Epoch 6/16
326/326 [==============================] - 130s 400ms/step - loss: 0.0927 - acc: 0.9686 - val_loss: 0.6716 - val_acc: 0.7500
Epoch 7/16
326/326 [==============================] - 129s 397ms/step - loss: 0.0833 - acc: 0.9693 - val_loss: 0.3433 - val_acc: 0.8125
Epoch 8/16
326/326 [==============================] - 130s 398ms/step - loss: 0.0799 - acc: 0.9711 - val_loss: 0.1118 - val_acc: 1.0000
Epoch 9/16
326/326 [==============================] - 129s 397ms/step - loss: 0.0730 - acc: 0.9734 - val_loss: 0.3607 - val_acc: 0.7500
Epoch 10/16
326/326 [==============================] - 130s 399ms/step - loss: 0.0638 - acc: 0.9772 - val_loss: 0.1541 - val_acc: 0.8750
Epoch 11/16
326/326 [==============================] - 130s 398ms/step - loss: 0.0568 - acc: 0.9814 - val_loss: 0.1593 - val_acc: 0.9375
Epoch 12/16
326/326 [==============================] - 130s 398ms/step - loss: 0.0577 - acc: 0.9793 - val_loss: 0.3239 - val_acc: 0.8125
Epoch 13/16
326/326 [==============================] - 130s 398ms/step - loss: 0.0534 - acc: 0.9810 - val_loss: 0.1434 - val_acc: 0.9375
Epoch 14/16
326/326 [==============================] - 129s 395ms/step - loss: 0.0538 - acc: 0.9804 - val_loss: 0.0683 - val_acc: 1.0000
Epoch 15/16
326/326 [==============================] - 128s 394ms/step - loss: 0.0459 - acc: 0.9829 - val_loss: 0.1161 - val_acc: 0.9375
Epoch 16/16
326/326 [==============================] - 127s 391ms/step - loss: 0.0429 - acc: 0.9850 - val_loss: 0.2229 - val_acc: 0.8125
```

Best Validation accuracy epoch-

Epoch 15/16

train_loss: 0.0459 - train_acc: 0.9829 - val_loss: 0.1161 - val_acc: 0.9375





We see some amount of overfitting which is possible since the model is huge and the dataset is relatively small.

Tested on a blind test set of 624 images-

[10]:
```
scores = inception_transfer.evaluate_generator(test_generator)
print('acc =',scores[1]*100)
```

```
acc = 89.74358974358975
```

[11]:
```
print(scores[0],scores[1])
```

```
0.2770901068698806 0.8974358974358975
```

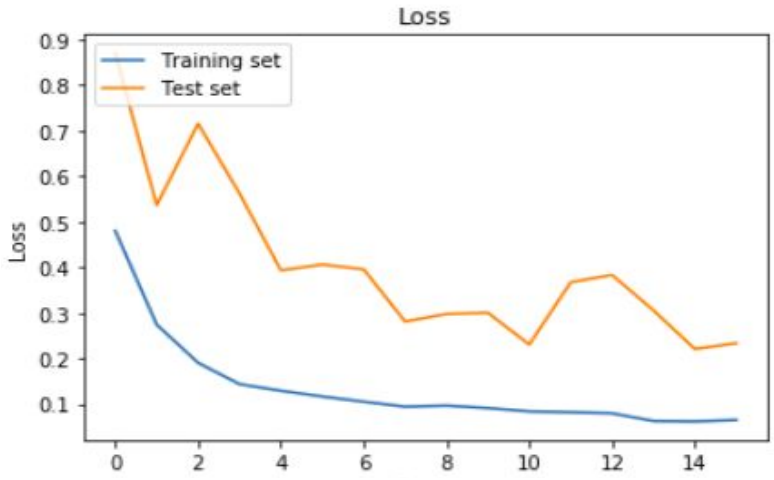
Model-4:

```
Epoch 1/16
326/326 [==============================] - 193s 592ms/step - loss: 0.4803 - acc: 0.7433 - val_loss: 0.8706 - val_acc: 0.5000
Epoch 2/16
326/326 [==============================] - 145s 445ms/step - loss: 0.2744 - acc: 0.8982 - val_loss: 0.5366 - val_acc: 0.6875
Epoch 3/16
326/326 [==============================] - 147s 450ms/step - loss: 0.1906 - acc: 0.9356 - val_loss: 0.7157 - val_acc: 0.6250
Epoch 4/16
326/326 [==============================] - 148s 453ms/step - loss: 0.1437 - acc: 0.9465 - val_loss: 0.5624 - val_acc: 0.6875
Epoch 5/16
326/326 [==============================] - 146s 448ms/step - loss: 0.1292 - acc: 0.9534 - val_loss: 0.3936 - val_acc: 0.8125
Epoch 6/16
326/326 [==============================] - 148s 454ms/step - loss: 0.1165 - acc: 0.9584 - val_loss: 0.4063 - val_acc: 0.8750
Epoch 7/16
326/326 [==============================] - 146s 446ms/step - loss: 0.1052 - acc: 0.9624 - val_loss: 0.3958 - val_acc: 0.8750
Epoch 8/16
326/326 [==============================] - 146s 447ms/step - loss: 0.0942 - acc: 0.9634 - val_loss: 0.2811 - val_acc: 0.9375
Epoch 9/16
326/326 [==============================] - 145s 444ms/step - loss: 0.0966 - acc: 0.9661 - val_loss: 0.2978 - val_acc: 0.8125
Epoch 10/16
326/326 [==============================] - 147s 452ms/step - loss: 0.0910 - acc: 0.9672 - val_loss: 0.3005 - val_acc: 0.8750
Epoch 11/16
326/326 [==============================] - 146s 448ms/step - loss: 0.0837 - acc: 0.9718 - val_loss: 0.2306 - val_acc: 0.9375
Epoch 12/16
326/326 [==============================] - 151s 465ms/step - loss: 0.0819 - acc: 0.9691 - val_loss: 0.3674 - val_acc: 0.8125
Epoch 13/16
326/326 [==============================] - 146s 449ms/step - loss: 0.0796 - acc: 0.9728 - val_loss: 0.3835 - val_acc: 0.7500
Epoch 14/16
326/326 [==============================] - 145s 445ms/step - loss: 0.0628 - acc: 0.9780 - val_loss: 0.3055 - val_acc: 0.8125
Epoch 15/16
326/326 [==============================] - 147s 452ms/step - loss: 0.0619 - acc: 0.9799 - val_loss: 0.2208 - val_acc: 0.9375
Epoch 16/16
326/326 [==============================] - 145s 446ms/step - loss: 0.0649 - acc: 0.9768 - val_loss: 0.2337 - val_acc: 0.8750
```

Best Validation accuracy epoch-

Epoch 15/16

train_loss: 0.0619 - train_acc: 0.9799 - val_loss: 0.2208 - val_acc: 0.9375



Tested on a blind test set of 624 images-

```
acc = 91.66666666666666
```

```
print(scores[0],scores[1])
```

```
0.22040158300063548 0.9166666666666666
```

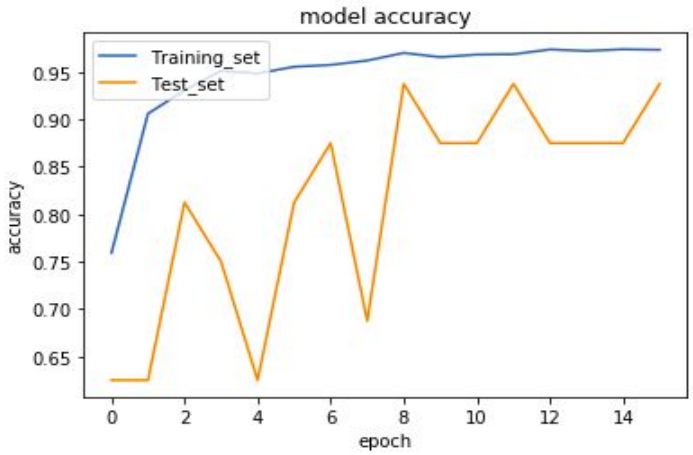
Model-5:

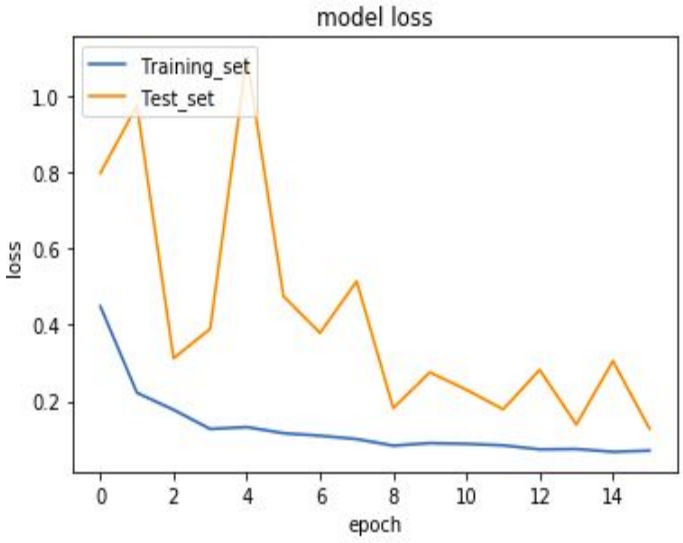
```
Epoch 1/16
326/326 [==============================] - 160s 492ms/step - loss: 0.4487 - acc: 0.7592 - val_loss: 0.7972 - val_acc: 0.6250
Epoch 2/16
326/326 [==============================] - 129s 395ms/step - loss: 0.2222 - acc: 0.9061 - val_loss: 0.9737 - val_acc: 0.6250
Epoch 3/16
326/326 [==============================] - 127s 391ms/step - loss: 0.1777 - acc: 0.9306 - val_loss: 0.3125 - val_acc: 0.8125
Epoch 4/16
326/326 [==============================] - 129s 395ms/step - loss: 0.1275 - acc: 0.9513 - val_loss: 0.3895 - val_acc: 0.7500
Epoch 5/16
326/326 [==============================] - 131s 401ms/step - loss: 0.1325 - acc: 0.9484 - val_loss: 1.1025 - val_acc: 0.6250
Epoch 6/16
326/326 [==============================] - 128s 394ms/step - loss: 0.1164 - acc: 0.9555 - val_loss: 0.4752 - val_acc: 0.8125
Epoch 7/16
326/326 [==============================] - 126s 386ms/step - loss: 0.1099 - acc: 0.9574 - val_loss: 0.3789 - val_acc: 0.8750
Epoch 8/16
326/326 [==============================] - 127s 391ms/step - loss: 0.1011 - acc: 0.9620 - val_loss: 0.5139 - val_acc: 0.6875
Epoch 9/16
326/326 [==============================] - 128s 393ms/step - loss: 0.0838 - acc: 0.9701 - val_loss: 0.1822 - val_acc: 0.9375
Epoch 10/16
326/326 [==============================] - 126s 386ms/step - loss: 0.0905 - acc: 0.9657 - val_loss: 0.2759 - val_acc: 0.8750
Epoch 11/16
326/326 [==============================] - 126s 386ms/step - loss: 0.0886 - acc: 0.9684 - val_loss: 0.2301 - val_acc: 0.8750
Epoch 12/16
326/326 [==============================] - 127s 390ms/step - loss: 0.0846 - acc: 0.9688 - val_loss: 0.1789 - val_acc: 0.9375
Epoch 13/16
326/326 [==============================] - 129s 395ms/step - loss: 0.0740 - acc: 0.9737 - val_loss: 0.2823 - val_acc: 0.8750
Epoch 14/16
326/326 [==============================] - 126s 387ms/step - loss: 0.0750 - acc: 0.9722 - val_loss: 0.1389 - val_acc: 0.8750
Epoch 15/16
326/326 [==============================] - 127s 390ms/step - loss: 0.0677 - acc: 0.9739 - val_loss: 0.3055 - val_acc: 0.8750
Epoch 16/16
326/326 [==============================] - 129s 395ms/step - loss: 0.0709 - acc: 0.9734 - val_loss: 0.1278 - val_acc: 0.9375
```

Best epoch-

Epoch 16/16

training loss: 0.0709 -training acc: 0.9734 - val_loss: 0.1278 - **val_acc: 0.9375**





When tested on a blind test set of 624 images:-

[loss, accuracy] - `0.16663386445874587 0.9375`

Test Set loss: 0.16663386445874857

Test Set Accuracy:- 93.75%

Tabulation of the Results-

Model	**Training Accuracy**	**Validation Accuracy**	**Test Accuracy**
1. Simple CNN	94.86	87.50	83.20
2. Modified CNN	95.25	75.00	88.94
3. ResNet50	98.50	93.75	89.74
4. InceptionV3	97.99	93.75	91.66
5. VGG16	97.39	93.75	93.75

4. Conclusion and Future work

In this work, we studied the different classifiers currently popular in the medical field along with modifying and pre training them. We have shown an analysis and comparison of the classifiers on augmented and preprocessed data. We have implemented transfer learning using the ImageNet dataset and the SGD optimizer for fine tuning to overcome overfitting on a very limited dataset.

Deep Learning is however limited by the amount of data available in the domain. Training models requiring more than a million images on a few thousand can cause overfitting. The accuracy achieved in this work could be improved further perhaps by modifying the models, using other models or by introducing a larger dataset. For future research we will try different ensemble techniques to try and combine the most unique and effective characteristics of individual classifiers.