Index

Information.

School of Emerging Science & Technology

Name: Jay.M. Soni

Roll No.: 31

Course: M.Sc. Data Science

Sem: 6

Email: Jaymsoni.jsm@gmail.com

Project Name:

Detection of Pneumonia from chest X-ray using Neural Networks.

Abstract:

In this project we created convolutional neural network models to accurately detect pneumonic lungs from chest X-rays, which can be utilized in the real world by medical practitioners to treat pneumonia. Experimentation was conducted on Chest X-Ray Images (Pneumonia) dataset available on Kaggle. The model first consists of five convolutional layers. The second model consists of seven convolutional layers. The first model achieves an accuracy of 79%, the second one reaches an accuracy of 92%, Dropout regularization is employed in the second model to minimize overfitting in the fully connected layers. Furthermore, recall and F1 scores are calculated from the confusion matrix of the second model for better evaluation.

Introduction:

- In the world of healthcare, one of the serious issues that medical professionals face is the correct diagnosis of conditions and diseases of patients. Not being able to correctly diagnose a condition is a problem for both the patient and the doctor. The doctor is not benefitting the patient in the appropriate way if the doctor misdiagnoses the patient. This could lead to malpractice lawsuits and overall hurt the doctor's business. The patient suffers by not receiving the proper treatment and risking greater harm to health by the condition that goes undetected; further, the patient undergoes unnecessary treatment and takes unnecessary medications, costing the patient time and money.
- If we can correctly diagnose a patient's condition, we have the potential to solve the above-mentioned problems. If we can produce deep learning models that can classify whether a patient has a condition or not, that can determine which condition the patient has, and that can determine the severity of the condition, then medical professionals will be able to use these models to better diagnose their patients. Accurate diagnosis can also be useful by allowing for timely treatment of a patient; being misdiagnosed can cause a delay in receiving the proper treatment.

Methodology:

- CNN model has been created from scratch and trained on Chest X-Ray Images (Pneumonia) dataset on Kaggle. Keras neural network library with TensorFlow backend has been

used to implement the models. Dataset consists of 4099 training images, 879 testing images and 879 validation images. The models have been trained on the training dataset, each with a different number of convolutional layers. The model was trained for 10 epochs, with training and testing batch sizes of 128.

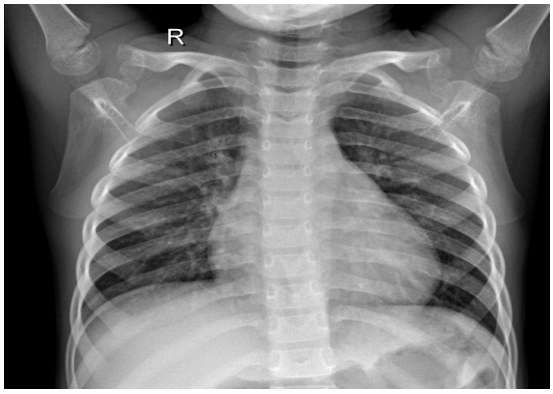
- There is, however, some preparation of the images that is necessary before applying an artificial neural network. The images need to be prepared using convolutional layers in a process called convolution. There are several stages in this process, convolution operation, 'ReLU' operation, pooling, and flattening; the result is a vector that we can feed into an artificial neural network.
- During the convolution operation, various feature detectors are applied to the image, creating a stack of feature maps — this stack of feature maps is called a convolutional layer. ReLU is applied to each feature map to enhance non-linearity. During the pooling stage, also known as subsampling, we apply max pooling to each feature map, creating smaller feature maps that preserve the relevant features of the image. The resulting stack of pooled featured maps forms the pooling layer.
- Once we get to the pooling layer, consisting of pooled feature maps, each pooled feature map is flattened into a vector and the resulting vectors are combined sequentially into one vector. The entries of this vector are fed into the input units of the artificial neural network. The artificial neural network is then trained on the training set and tested on the test set.
- Function of Convolution
- The calculation equation of the convolutional neural network is shown in N = (W – F + 2p)/s + 1 where is the output size, Wis the input size, F is the convolution kernel size, P is the padding value, and S is the stride value. For example, we input an RGB image, the size of our input image is 227 X 227 X 3, that is, it has three channels, and the size of each channel is 227 * 227. We set the padding as 0 and the stride as 4, and then, the convolution kernel size is 3 * 3. Through Formula, we can calculate the output size as N = (227 – 3 + 2 x 0)/4 + 1 = 57. The larger the convolution kernel, the larger the receptive field, the more image information you can see, and the better the features you can obtain. A large convolution kernel will cause a surge in calculations, which is not conducive to the increase of model depth, and calculation performance will also decrease. Therefore, better

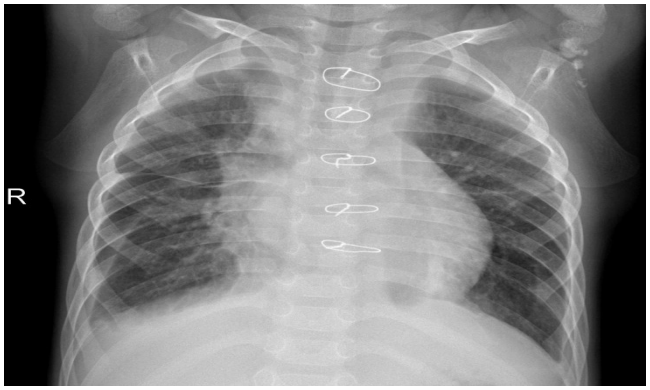
features can be obtained by mixing convolution kernels of different sizes.

- The convolutional Neural Network is nothing but a combination of Artificical Neural Network and Convolution.

Dataset:

Chest X-Ray Images (Pneumonia) dataset of 1.16 GB size has been imported from Kaggle, with total of 5856 jpeg images split into Train, Test and Val folders each divided into category Pneumonia and Normal. Chest X-ray images (front and back) were selected from pediatric patients Guangzhou Women and Children’s Medical Center.





In the above images the first one is Normal lungs, and the second images is of pneumonia person's lungs.

CNN Architecture

CNN models are feed-forward networks with convolutional layers, pooling layers, flattening layers and fully connected layers employing suitable activation functions.

Convolutional layer.

It is the building block of the CNNs. Convolution operation is done in mathematics to merge two functions. In the CNN models, the input image is first converted into matrix form. Convolution filter is applied to the input matrix which slides over it, performing element-wise multiplication and storing the sum. This creates a feature map. A 3 ×3 filter is generally employed to create 2D

feature maps when images are black and white. Convolutions are performed in 3D when the input image is represented as a 3D matrix where the RGB color represents the third dimension. Several feature detectors are operated with the input matrix to generate a layer of feature maps which thus forms the convolutional layer.

Activation functions.

All four models presented in this paper use two different activation functions, namely ReLU activation function and activation function. The ReLU activation function stands for rectified linear function. It is a nonlinear function that outputs zero when the input is negative and outputs one when the input is positive. The ReLU function is given by the following formula: This type of activation function is broadly used in CNNs as it deals with the problem of vanishing gradients and is useful for increasing the nonlinearity of layers. ReLU activation function has many variants such as Noisy ReLUs, Leaky ReLUs and Parametric ReLUs. Advantages of ReLU over other activation functions are computational simplicity and representational sparsity. Softmax activation function is used in all four models presented in this paper. This broadly used activation function is employed in the last dense layer of all four models. This activation function normalizes inputs into a probability distribution. Categorical cross-entropy cost function is mostly used with this type of activation function.

Pooling layer.

Convolutional layers are followed by pooling layers. The type of pooling layer used in all four models is max-pooling layers. The max-pooling layer having a dimension of 2 ×2 selects the maximum pixel intensity values from the window of the image currently covered by the kernel. Max-pooling is used to down sample images, hence reducing the dimensionality and complexity of the image. Two other types of pooling layers can also be used which are general pooling and overlapping pooling.

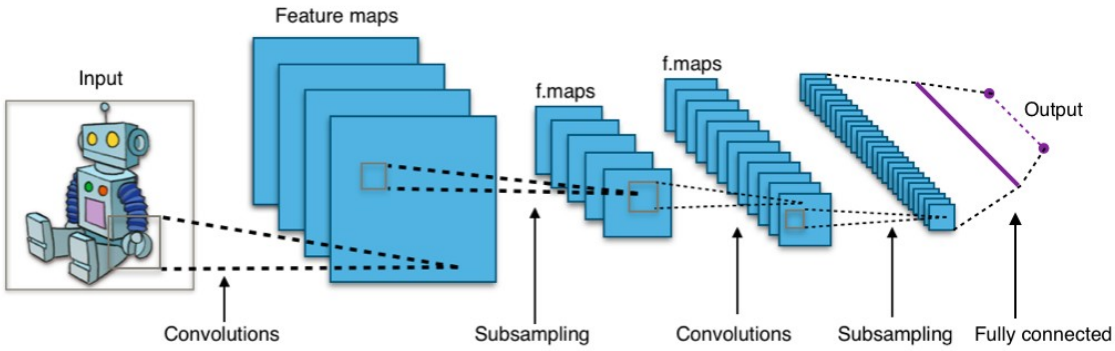
The models presented in this paper use max-pooling technique as it helps recognize salient features in the image.

Flattening layer and fully connected layers.

After the input image passes through the convolutional layer and the pooling layer, it is fed into the flattening layer. This layer flattens out the input image into a column, further reducing its computational complexity. This is then fed into the fully connected layer/dense layer. The fully connected layer has multiple layers, and every node in the first layer is connected to every node in the second layer. Each layer in the fully connected layer extracts features, and on this basis, the network makes a prediction. This process is known as forward propagation. After forward propagation, a cost function is calculated. It is a measure of performance of a neural network model. The cost function used in all four models is categorical cross-entropy. After the cost function is calculated, back propagation takes place. This process is repeated until the network achieves optimum performance. Adam optimization algorithm has been used in all four models.

Reducing overfitting.

The first model exhibits substantial overfitting; hence, dropout technique was employed in the later models. Dropout technique helps to reduce overfitting and tackles the problem of vanishing gradients. Dropout technique encourages each neuron to form its own individual representation of the input data. This technique on a random basis cuts connections between neurons in successive layers during the training process. The learning rate of models was also modified, to reduce overfitting. Data augmentation techniques can also be employed to reduce overfitting.



Algorithm of CNN classifiers:

The algorithms used in the convolutional neural network classifiers have been explained in the figure below. Classifier models trained for a greater number of epochs have shown overfitting. Several optimizer functions were also trained and studied. Adam optimizer function was finalized to be used for classifiers after it gave the best results. Initially, a simple classifier model with convolutional layer of image size set to 512* 512, 64 feature maps and employing ReLU activation function was trained. A fully connected dense layer with 128 perceptron was utilized. Dense layer, dropout layer and learning rate were there. There are seven layers for this model for best results.

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 512, 512, 1)]     0
_________________________________________________________________
conv2d (Conv2D)              (None, 512, 512, 64)      640
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 256, 256, 64)      0
_________________________________________________________________
dropout (Dropout)            (None, 256, 256, 64)      0
```

conv2d_1 (Conv2D)	(None, 256, 256, 96)	55392
max_pooling2d_1 (MaxPooling2	(None, 128, 128, 96)	0
dropout_1 (Dropout)	(None, 128, 128, 96)	0
conv2d_2 (Conv2D)	(None, 128, 128, 128)	110720
max_pooling2d_2 (MaxPooling2	(None, 64, 64, 128)	0
dropout_2 (Dropout)	(None, 64, 64, 128)	0
conv2d_3 (Conv2D)	(None, 62, 62, 160)	184480
max_pooling2d_3 (MaxPooling2	(None, 31, 31, 160)	0
dropout_3 (Dropout)	(None, 31, 31, 160)	0
conv2d_4 (Conv2D)	(None, 31, 31, 192)	276672
max_pooling2d_4 (MaxPooling2	(None, 15, 15, 192)	0
dropout_4 (Dropout)	(None, 15, 15, 192)	0
conv2d_5 (Conv2D)	(None, 15, 15, 224)	387296
max_pooling2d_5 (MaxPooling2	(None, 7, 7, 224)	0
dropout_5 (Dropout)	(None, 7, 7, 224)	0
conv2d_6 (Conv2D)	(None, 7, 7, 256)	516352
max_pooling2d_6 (MaxPooling2	(None, 3, 3, 256)	0
dropout_6 (Dropout)	(None, 3, 3, 256)	0
global_average_pooling2d (Gl	(None, 256)	0
dense (Dense)	(None, 1)	257

Total params: 1,531,809
Trainable params: 1,531,809
Non-trainable params: 0

After creating model we fit model model with data which is generated from keras function ImageDataGenrator

```
datagen = ImageDataGenerator(rescale = 1/255,
                             zoom_range=0.1,
                             height_shift_range=0.05,
                             width_shift_range=0.05,
                             rotation_range=5)
```

This will create an augmented image for our model to fit that will be helpful in accuracy, also reduce computation time sice we have `Total params: 1,531,809` , which is quite high, after that we will fit model.

Code:

```
history = model.fit(
    train_gen,
    epochs = 10,
    validation_data = val_gen
)
```

Output :

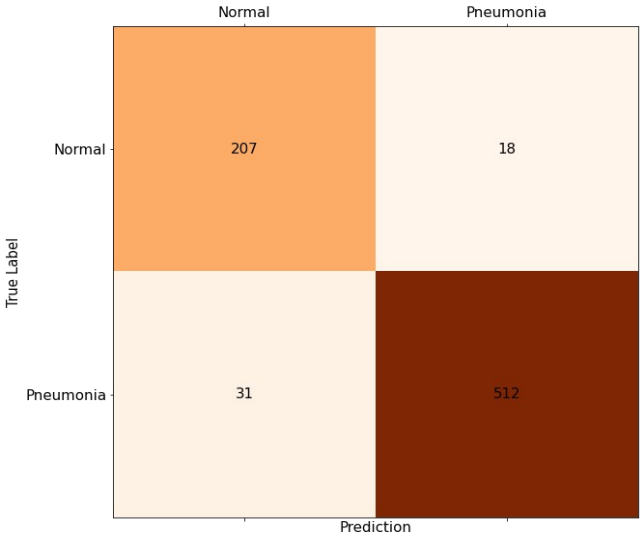
```
Epoch 1/10


2022-04-30 12:23:46.927899: I
tensorflow/stream_executor/cuda/cuda_dnn.cc:369] Loaded cuDNN version
8005


33/33 [==============================] - 116s 3s/step - loss: 0.6508 -
accuracy: 0.7329 - val_loss: 0.5856 - val_accuracy: 0.7369
Epoch 2/10
33/33 [==============================] - 87s 3s/step - loss: 0.5819 -
accuracy: 0.7329 - val_loss: 0.5868 - val_accuracy: 0.7369
Epoch 3/10
33/33 [==============================] - 87s 3s/step - loss: 0.5865 -
accuracy: 0.7329 - val_loss: 0.6216 - val_accuracy: 0.7369
Epoch 4/10
```

```
33/33 [==============================] - 86s 3s/step - loss: 0.5114 -
accuracy: 0.7597 - val_loss: 0.5121 - val_accuracy: 0.7608
Epoch 5/10
33/33 [==============================] - 87s 3s/step - loss: 0.4239 -
accuracy: 0.8068 - val_loss: 0.3695 - val_accuracy: 0.8462
Epoch 6/10
33/33 [==============================] - 87s 3s/step - loss: 0.3484 -
accuracy: 0.8522 - val_loss: 0.3146 - val_accuracy: 0.8759
Epoch 7/10
33/33 [==============================] - 87s 3s/step - loss: 0.3067 -
accuracy: 0.8683 - val_loss: 0.2308 - val_accuracy: 0.9226
Epoch 8/10
33/33 [==============================] - 87s 3s/step - loss: 0.2762 -
accuracy: 0.8892 - val_loss: 0.2061 - val_accuracy: 0.9248
Epoch 9/10
33/33 [==============================] - 86s 3s/step - loss: 0.2111 -
accuracy: 0.9200 - val_loss: 0.1883 - val_accuracy: 0.9157
Epoch 10/10
33/33 [==============================] - 87s 3s/step - loss: 0.2053 -
accuracy: 0.9185 - val_loss: 0.1320 - val_accuracy: 0.9522
```

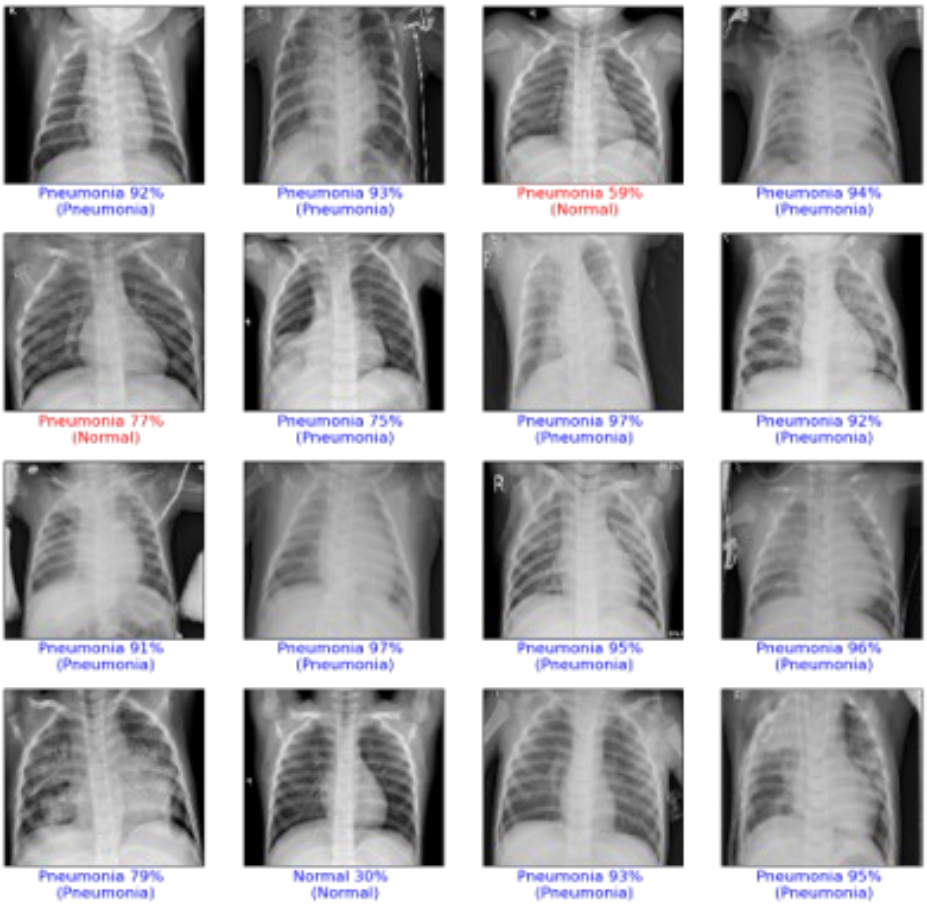
Accuracy :



- Precision with base test data: 0.9847
- Recall with base test data: 0.9748

- Loss rate on evaluation data : 0.1899
- Accuracy rate on evaluation data : 0.9294

Evaluation on Unseen Data with 92% accuracy is not bad but not good enough for real life cases



Conclusion

The validation accuracy, recall and F1 score of CNN classifier model with seven convolutional layers are 92.31%, which are quite high. The models created by us at best could achieve 92.31% accuracy which is lower, but 98% recall has been achieved. High recall values will ensure that the number of false-negative instances is lower, hence lowers the risk to the patient's life. Diagnostic results, thus helping healthcare systems provide efficient patient care services and reduce mortality rates. These convolutional neural networks' models were successfully achieved by employing various methods of parameter tuning like adding dropout, changing learning rates, changing the batch size, number of epochs, adding more complex fully connected layers and changing various stochastic gradient Optimizers.

Acknowledgement:

Reference:
https://www.researchgate.net/publication/340961287_Pneumonia_Detection_Using_Convolutional_Neural_Networks_CNNs

https://www.hindawi.com/journals/cmmm/2021/8854892/#introduction

Dataset Link :

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

Code Link on kaggle :

https://www.kaggle.com/code/jaysoni7/project-6/edit/run/94410553