VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

SOFT COMPUTING (SWE1011)

LUNG CANCER DETECTION USING TRANSFER LEARNING

SUBMITTED TO

Prof. JAGADEESH KANAN R

SUBMITTED BY

ANEESH MUTHYALA -16MIS1057

WINTER SEMESTER 2019-20

VELLORE INSTITUTE OF TECHNOLOGY
MTech Integrated (software engineering)
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING
CHENNAI, (600127)
TAMIL NADU

ABSTRACT

Each year, pneumonia claims about one million victims globally. In general, the risk is most severe for children, though there are regional disparities as well. According to UNICEF, pneumonia causes more deaths in children than HIV/AIDS and malaria combined. One of the most urgent needs in combatting pneumonia is a cost-effective manner of diagnosing X-ray images. Current methods require highly skilled personnel and are impractical for many regions of the world. The doctor, when interpreting the chest x-ray, will look for white patches in the lungs to detect pneumonia. However, such hazy patterns would also be observed tuberculosis and severe cases of bronchitis. For the purpose of conclusive diagnosis, the purpose of this study is to identify pneumonia, through analysis of chest X-ray images, to recognize patterns of the disease using a neural network.

INTRODUCTION

Pneumonia is a significant cause of mortality in children across the world. According to the World Health Organization (WHO), around 2 million pneumonia-related deaths are reported every year in children under 5 years of age, making it the most significant cause of pediatric death. Pneumonia sourced from bacterial and viral pathogens are the two leading causes and require different forms of management. Bacterial pneumonia is immediately treated with antibiotics while viral pneumonia requires supportive care, making timely and accurate diagnosis important. Chest X-ray (CXR) analysis is the most commonly performed radiographic examination for diagnosing and differentiating the types of pneumonia. However, rapid radiographic diagnoses and treatment are adversely impacted by the lack of expert radiologists in resource-constrained regions where pediatric pneumonia is highly endemic with alarming mortality rates.

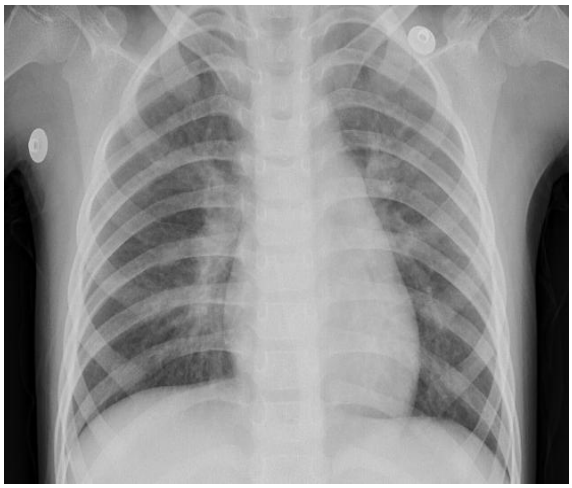
With numerous lung diseases people can grow, here is just any specimen of diseases people can protect if they discover it out prior. With the expertise computer and machine power, the prior credentials of diseases, mainly lung disease, this paper can be assisted to detect prior besides

more correctly, which can protect numerous people in addition to decrease the diseases on the method. The health scheme has not established in time through the growth of people

With the control of computers along with the huge volume of records being unrestricted to the widespread, this is a high time to put up resolving this complication. Wishing to put up additional to the society, serving those who are not capable to suffer for medical expenditures, in this solution can put up decreasing medical costs with the enlargement of computer science for health and medical science projects. For implementation, the big lung X-ray data is culled from Kaggle

PROBLEM STATEMENT

Take a look at those two photos. Which one do you think is infected with pneumonia?



If you are after medical training you have probably guessed right. For us — we couldn't really spot any significant difference between the pictures. The problem can be in fact tricky even for professionals and might require additional tests.

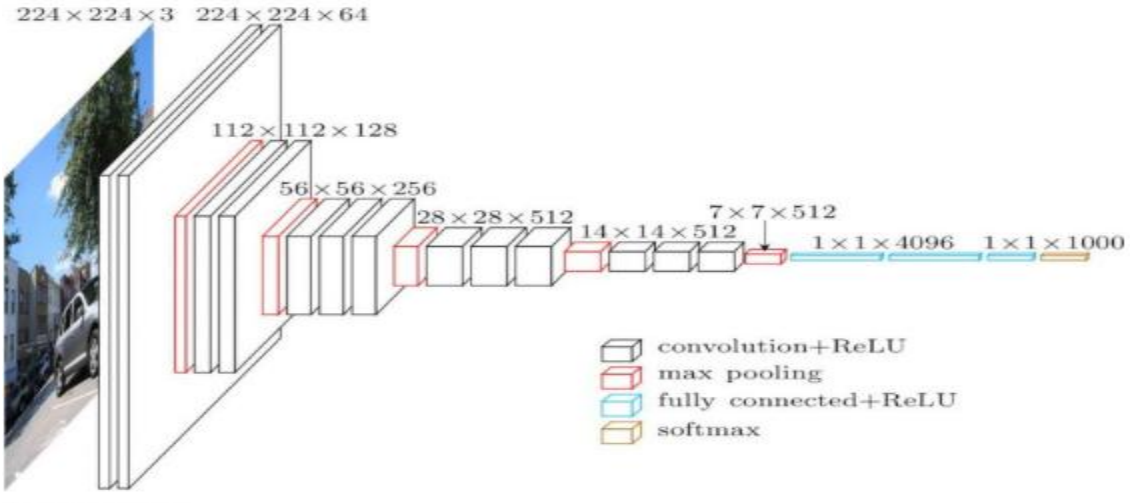
We therefore were super-curious to see if neural network, provided with properly labelled data would be able to see patterns that could split patients into 2 classes — normal/pneumonia.

DATASET

Our journey started with Kaggle dataset. It consists of 5'863 X-ray images of lungs taken on a group of paediatric patients that are 1–5 years old. All of images have been labeled by 2 specialists to minimize labeling error risk. The data was already split for us into training, validation and test datasets.

ARCHITECTURAL DESIGN OF VGG16

- **Input.** VGG takes in a 224x224 pixel RGB image. For the ImageNet competition, the authors cropped out the center 224x224 patch in each image to keep the input image size consistent.
- **Convolutional Layers.** The convolutional layers in VGG use a very small receptive field (3x3, the smallest possible size that still captures left/right and up/down). There are also 1x1 convolution filters which act as a linear transformation of the input, which is followed by a ReLU unit. The convolution stride is fixed to 1 pixel so that the spatial resolution is preserved after convolution.
- **Fully-Connected Layers.** VGG has three fully-connected layers: the first two have 4096 channels each and the third has 1000 channels, 1 for each class.
- **Hidden Layers.** All of VGG's hidden layers use ReLU (a huge innovation from AlexNet that cut training time). VGG does not generally use Local Response Normalization (LRN), as LRN increases memory consumption and training time with no particular increase in accuracy.



VGG16 Architecture

METHODOLOGY

Our main principle is to predict whether the child is having Pneumonia or not. For this prediction we have to train our model with most accuracy so that it can study each and every pixel of the image clearly. Transfer Learning has many features to process the images and predict the output.

Data Preprocessing

In this preprocessing of data for mutually full dataset as well as sample dataset in "Data preprocessing" contains of next steps:

- At first rescale all images for the purpose of reducing size with the feature in which creates training faster.
- The all images are transformed to rgb and gray. Mutually are conducted for various models.
- In this next step the numpy array uses for read the images at that time normalize by separating the images matrix using 255.

Detailed Architecture

During training, the input to our ConvNets is a fixed-size 224 × 224 RGB image. The only preprocessing we do is subtracting the mean RGB value, computed on the training set, from each pixel. The image is passed through a stack of convolutional (conv.) layers, where we use filters with a very small receptive field: 3 × 3 (which is the smallest size to capture the notion of left/right, up/down, center). In one of the configurations we also utilize 1 × 1 convolution filters, which can be seen as a linear transformation of the input channels (followed by non-linearity). The convolution stride is fixed to 1 pixel; the spatial padding of conv. layer input is such that the spatial resolution is preserved after convolution, i.e. the padding is 1 pixel for 3 × 3 conv. layers. Spatial pooling is carried out by five max-pooling layers, which follow some of the conv. layers (not all the conv. layers are followed by max-pooling). Max-pooling is performed over a 2 × 2 pixel window, with stride 2.

A stack of convolutional layers (which has a different depth in different architectures) is followed by three Fully-Connected (FC) layers: the first two have 4096 channels each, the third performs 1000- way ILSVRC classification and thus contains 1000 channels (one for each

class). The final layer is the soft-max layer. The configuration of the fully connected layers is the same in all networks

IMPLEMENTATION

- We have Google Colaboratory to run the code. So hence first we have to mount the data to the drive. Next step is to import all the required libraries from keras library.
- Then we will call the VGG16 model with parameter as "include_top=false" because we need to design our own hidden layers.
- We will then execute a for loop saying that al the existing training values should be erased and new values must be recorded as per the current runtime.
- We will declare the last hidden layer as 'Flatten' layer to convert the dimensions of the image. The output layer is defined as 'Dense' layer with parameter as '2' because we have only two prediction classes- Normal, Pneumonia.
- We will create the model using the inputs and the vgg16 model declared above and the print the model summary to check the layers with their dimensions.
- The next step is to use the image data generator to import the images from the dataset and to divide them into training and testing dataset.
- Then use the model.fit_generator to send all the values into the model and fit the data to train it efficiently. We can as many epochs as we like. A standard count of epochs will give higher efficiency. If the size of the dataset is high then this step might take some time (1-1 ½ hour).
- After the model is trained we will plot the accuracy and loss of the prediction done by model while training. The main purpose is to visualize that how accurately did our model is trained.
- If satisfied with the model we will import it to a .h file.
- Now al the predictions will be done using the model in .h file. We will pass each and every image into the model. The output will be 1*2 matrix where 1st column represents Normal and 2nd column reprents Pneumonia. The output will be a Boolean output with 1's and 0'2 at respective places. If 1 is at the 1st column it means the passed image does not have Pneumonia if in the output 1 is at the second column it means that the image has Pneumonia Bacteria.

TRAINING OF VGG16

The architecture of the VGG-16 model [1]. It has 13 conv layers and 3 fc layers. The column "complexity" is the theoretical time complexity, shown as relative numbers to the total convolutional complexity. The column "# of zeros" is the relative portion of zero responses, which shows the "sparsity" of the layer.

layer	filter size	# channels	# filters	stride	output size	complexity (%)	# of zeros
$Conv1_1$	3×3	3	64	1	224×224	0.6	0.48
$Conv1_2$	3×3	64	64	1	224×224	12.0	0.32
Pool1	3×3			2	112×112		
$Conv2_1$	3×3	64	128	1	112×112	6.0	0.35
$Conv2_2$	3×3	128	128	1	112×112	12.0	0.52
Pool2	2×2			2	56×56		
$Conv3_1$	3×3	128	256	1	56×56	6.0	0.48
$Conv3_2$	3×3	256	256	1	56×56	12.1	0.48
$Conv3_3$	3×3	256	256	1	56×56	12.1	0.70
Pool3	2×2			2	28×28		
$Conv4_1$	3×3	256	512	1	28×28	6.0	0.65
$Conv4_2$	3×3	512	512	1	28×28	12.1	0.70
$Conv4_3$	3×3	512	512	1	28×28	12.1	0.87
Pool4	2×2			2	14×14		
$Conv5_1$	3×3	512	512	1	14×14	3.0	0.76
$Conv5_2$	3×3	512	512	1	14×14	3.0	0.80
$Conv5_3$	3×3	512	512	1	14×14	3.0	0.93

Whole-model acceleration with/without rank selection for VGG-16. The solver is the asymmetric version. The speedup ratios shown here involve all convolutional layers. We do not accelerate Conv11. In the case of no rank selection, the speedup ratio of each other layer is the same. Each column of C12-C53 shows the rank d0 used, which is the number of filters after approximation. The error rates are top-5 single-view, and shown as the increase of error rates compared with no approximation.

speedup	rank sel.	$C1_1$	$C1_2$	$C2_1$	$C2_2$	$C3_1$	$C3_2$	$C3_3$	$C4_1$	$C4_2$	$C4_3$	$C5_1$	$C5_2$	$C5_3$	err. ↑ %
2×	no	64	28	52	57	104	115	115	209	230	230	230	230	230	0.99
2×	yes	64	18	41	50	94	96	116	207	213	260	467	455	442	**0.28**
3×	no	64	19	34	38	69	76	76	139	153	153	153	153	153	3.25
3×	yes	64	15	31	34	68	64	75	134	126	146	312	307	294	**1.66**
4×	no	64	14	26	28	52	57	57	104	115	115	115	115	115	6.38
4×	yes	64	11	25	28	52	46	56	104	92	100	232	224	214	**3.84**

EXPERIMENT AND CALCULATIONS WITH VGG16

The very deep VGG models have substantially improved a wide range of visual recognition tasks, including object detection semantic segmentation image captioning, video/action recognition, image question answering, texture recognition etc. Considering the big impact yet slow speed of this model, we believe it is of practical significance to accelerate this model.

Accelerating VGG-16 for ImageNet Classification

Firstly we discover that our whole-model rank selection is particularly important for accelerating VGG-16. In theBelow figure, we show the results without/with rank selection. No 3d decomposition is used in this comparison. For a 4 speedup, the rank selection reduces the increased error from 6.38% to 3.84%. This is because of the greater diversity of layers in VGG

increase of top-5 error (1-view)			
speedup ratio	3×	4×	5×
Jaderberg *et al.* [17] (our impl.)	2.3	9.7	29.7
our asym. (3d)	0.4	0.9	2.0
our asym. (3d) FT	**0.0**	**0.3**	**1.0**

Accelerating the VGG-16 model using a speedup ratio of 3, 4, or 5. The top-5 error rate (1-view) of the VGG-16 model is 10.1%. This table shows the increase of error on this baseline.

model	speedup solution	top-5 error (1-view)	CPU (ms)	GPU (ms)
VGG-16 [1]	-	10.1	3287	18.60
VGG-16 (4×)	Jaderberg *et al*. [17] (our impl.)	19.8	875 (3.8×)	6.40 (2.9×)
	our asym.	13.9	875 (3.8×)	7.97 (2.3×)
	our asym. (3d)	11.0	860 (3.8×)	6.30 (3.0×)
	our asym. (3d) FT	**10.4**	858 (3.8×)	6.39 (2.9×)

The top-5 error is the absolute value.The running time is a single view on a CPU (single thread, with SSE) or a GPU. The accelerated models are those of 4 theoretical speedup. On the brackets are the actual speedup ratios.

```
Model: "model_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_2 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_2 (Dense)              (None, 2)                 50178
=================================================================
Total params: 14,764,866
Trainable params: 50,178
Non-trainable params: 14,714,688
_________________________________________________________________
```

CONCLUSION

We have demonstrated how to classify positive and negative pneumonia data from a collection of X-ray images. We build our model from scratch, which separates it from other methods that rely heavily on transfer learning approach. The Convolutional Neural Network was used to train the neural network and, for the validation of the model, Cross validation was used. The classification model presented was efficient in the classification, obtaining an average accuracy of 95.30 % in the tests.

REFERENCES

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

https://towardsdatascience.com/step-by-step-vgg16-implementation-in-keras-for-beginners-a833c686ae6c

https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33

https://www.youtube.com/watch?v=apkOsw27Cng

https://www.youtube.com/watch?v=zBOavqh3kWU&t=772s