PNEUMONIA DETECTION USING DEEP LEARNING MODELS

Iraklis Konsoulas

it185200@iee.ihu.gr - dyka3773@gmail.com
Informatics and Electronics Engineer
International Hellenic University, Alexander Campus
57400 Sindos, Thessaloniki, Greece

January 2021

ABSTRACT

Technology is constantly playing a bigger role in today's medicine and after an almost full year of quarantine and a pandemic still goin on, technology should help doctors as much as it can to do their jobs faster and easier. The purpose of this study was to determine the effectiveness of Deep Learning models on recognizing pneumonia infections from standard chest X-ray images. Most of the studies and machine learning competitions online[1] are focusing on whether the patient has pneumonia or not[2]. This study is going into more details about which major type of pneumonia, bacterial or viral, the patient has. The models have achieved an accuracy of 81% which confirms that they are effective and can be implemented for the detection of Pneumonia in real life actual patients.

Keywords — ResNet, DenseNet, Pneumonia, Deep Learning, Biomedical Technologies

1. INTRODUCTION

The everyday use of X-ray imaging is crucial for today's medicine as it is considered a standard cheap procedure in most cases of lung diseases. Most of the time, it is a prerequisite for further diagnosis and treatment. Being a standard procedure in cases of lung infections, it greatly applies to all cases of pneumonia, as it may define whether its origin is bacterial, viral, fungal or protozoan[3], and adapt its treatment accordingly.

Investigating the improvement machine learning has to offer on this particular medical field, is indisputably intriguing[4]. Saving time, money and labour for both doctors and medical staff, as well as patients, is highly beneficial for all parts, not to mention the impact on the emergency and the accuracy of the necessary treatment. Bearing all this in mind the following machine learning models and their application might, at least, be used as a double-check to eliminate possible mistakes. Needless to say that nowadays with CoViD-19 pandemic still going on and causing severe cases of pneumonia, such an application is even more vital for obvious reasons[5].

Despite the fact that most of the study cases focus on either solely identifying CoViD-19[6] or, on the other hand, tracing pneumonia[7], this particular study aims at specifying the original cause pneumonia itself, bacterial or viral.

Using some of the most well-known Convolutional Neural Net models [8], ResNet and DenseNet, to be trained on a particular X-ray dataset, led to some quite accurate outcomes, on both models. The first model reached an accuracy of 81.8% using test data and 82.3% in its train data, whereas, the second a slightly lower 80.8% and 81.3% respectively - still allowing space for further improvement.

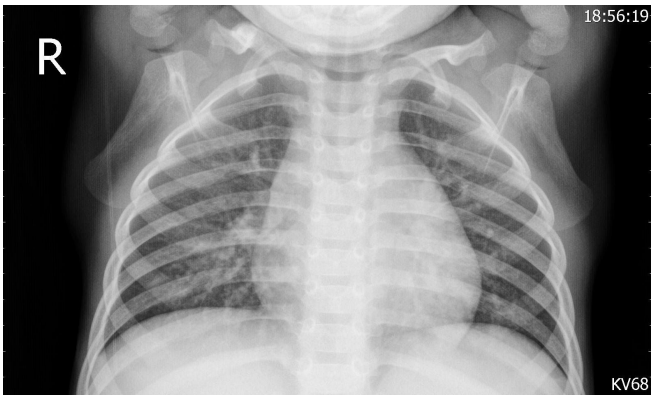
The extensive methodology and procedure, as well as, a more descriptive analysis of the dataset and resources used, is reported below, not flawless, but in the hopes of a future greater impact of Deep Learning on the improvement of human life and activities.

2. THE DATASET

The given dataset consists of 4672 X-ray images classified according to patient's condition :

- Class 0 : Healthy patient
- Class 1 : Bacterial infection causing the pneumonia

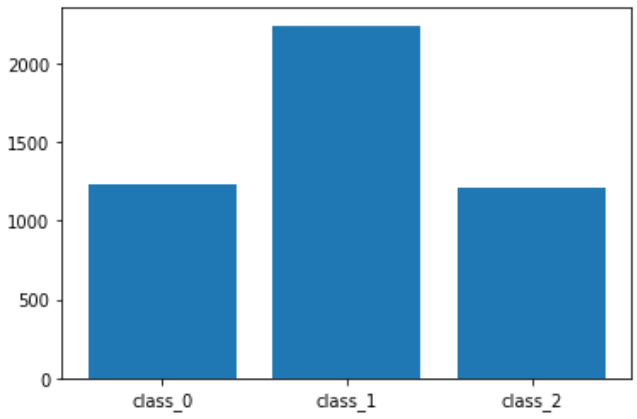
- Class 2 : Viral infection causing the pneumonia



[2.1] Example of a Class 2 patient with Viral infection

The image [2.1] above has a 1576 x 944 resolution, and almost every image in the given dataset has a different aspect ratio and resolution. To fix that problem, the images had to be pre-processed (More on paragraph [4.1]).

As you can see in the image [2.2] below the distribution of the images is really imbalanced. Therefore, data augmentation had to be used (See paragraph [4.4]) in order for the models to properly fit and ensure the avoidance of overfitting in one particular class.



[2.2] Image distribution on different classes

3. RESOURCES USED

To achieve an easier implementation of the Machine Learning algorithms, Google Colab was used to write two Python notebooks, one for each model I trained. Keras[9], TensorFlow[10] and SciKit-Learn[11] libraries were in use to help me write less but more efficient code. The complete list of the Python libraries that were used is :

- matplotlib
  - pyplot
  - image
- numpy
  - linalg.inv
  - linalg.norm
- os
- time
- IPython
  - display
- pandas
- keras
- tensorflow
  - keras.utils.plot_model
  - keras.layers.Dense
  - keras.layers.Conv2D
  - keras.layers.BatchNormalization
  - keras.layers.Activation
  - keras.layers.Concatenate
  - keras.layers.AveragePooling2D
  - keras.layers.Input
  - keras.layers.Flatten
  - keras.layers.MaxPooling2D
  - keras.layers.GlobalAveragePooling2D
  - keras.optimizers.Adam
  - keras.callbacks.ModelCheckpoint
  - keras.callbacks.LearningRateScheduler
  - keras.callbacks.ReduceOnPlateau
  - keras.regularizers.l2
  - keras.backend
  - keras.models.Model
  - keras.preprocessing.image.ImageDataGenerator
- PIL
  - Image
- sklearn
  - model_selection.train_test_split
- csv

Both of the models trained, used an implementation of ResNet and DenseNet code in Keras and TF.

4. PROGRAM WORKFLOW

4.1. Image Preprocessing

After importing the images and their labels, all the images had to be resized and preprocessed to fit in the model (The processed images are in PIL format). After

repetitive experimentation and research on the average image size on CNN models, I found out that the optimal image size was 170 x 170, which is a great downscale from the [2.1] image, but it fits the restrictions of Google Colab's RAM [12] and it doesn't deform the images.

The aspect ratio has changed to 1:1. Due to the randomness of the ratios in the images, it seemed that the best option was to average them all up to squares. This consequently led me into filling every image with an offset border rectangle, which is black in order to match with the X-ray's background. An example of the above mentioned technique implemented on the previous X-ray image [2.1] can be seen below [4.1.1] :

[4.1.1] The [2.1] image after its preprocessing

4.2. Data Normalization

To begin with, the PIL images were turned into numpy arrays and every pixel was divided by 255, so that every value ranges between 0 and 1. The data was split into Train and Validation data using the `train_test_split` function so that were distributed in a 70-30 ratio.

The arithmetic mean of the Train array is saved and subtracted from both arrays. That way, the unique differences and/or features of each image are way more visible to our program.

Every label array, is transformed into a binary vector using `keras.utils.to_categorical` function, which helps our model in its final activation function Softmax (More on that on paragraphs 5.1 and 5.3)

4.3. Functions and Instantiations

Two major functions were defined and used during the training :

- `lr_schedule`, which is a learning rate scheduler that changes the model's learning rate according to the epoch :
  - 0.001 after the 80th epoch
  - 0.0001 after the 120th epoch
  - 0.00001 after the 160th epoch
  - 0.000005 after the 180th epoch
- `MyCallback`, which is a callback function that prints different model attributes on the screen.

After the functions' definition, the models were compiled using Categorical Crossentropy as their loss function and Adaptive Moment Estimation (Adam[13]) as their optimizer which starts from an 0.01 original learning rate.

4.4. Data Augmentation

As mentioned above ([2.2]), it is obvious that an `ImageDataGenerator` was needed. After thorough research, on other models built for X-ray imaging[2], it became evident that the following changes on the images didn't affect the class of the image :

- Rotating it up to 30°
- Zooming by 20% on random parts of the image
- Shifting images horizontally and vertically by 10% of their width or height respectively
- Flipping it horizontally

Taking all this into consideration,the next step was for the data generator to start fitting, using these parameters.

4.5. Model Fitting and Results

Both models were trained using the same parameters. The batch size was 32 [1], the epochs were 200 and the callback functions were :

- `lr_schedule`
- `MyCallback`
- `ReduceOnPlateau`
- `ModelCheckpoint`

Models were evaluated and plots displaying their 'accuracy' and 'validation accuracy' were demonstrated accordingly (See paragraph 6).

To use the model in actual unknown data, the user has to normalize and preprocess their data the former mentioned way, and preferably use the same

[1] Original ResNet paper has a 128 batch size [14]

arithmetic mean value to have the best results (That's why it was saved before).

5. MODEL DESCRIPTIONS

Two models were trained to classify the given dataset :

- ResNet20
- DenseNet20

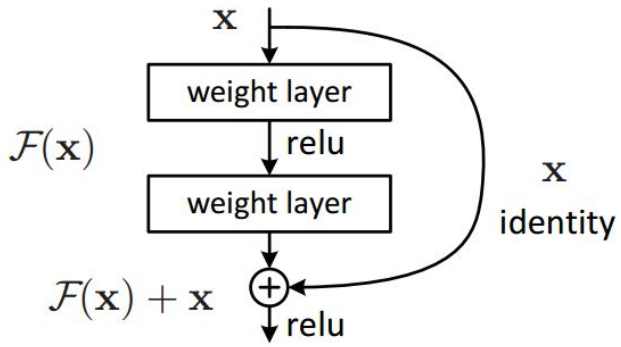
The most successful one being ResNet20 as expected, but only by a small percentage.

Those two were finally chosen after extensive research, since they are some of the best and most well-known Convolutional Neural Nets for image classification. It should be mentioned that the choice was also affected by the limitations placed by Colab's restricted RAM capacity.

5.1. ResNet20

ResNet20 is a CNN with 20 ResNet layers consisting of one Convolution (`Conv2D`) layer, one `BatchNormalization` function and one Activation function, usually 'ReLU'.

ResNet uses Residual blocks of layers which help minimize the problem of vanishing gradients. That means that some streams of data skip some layers in order to remain intact. An example of a Residual block would look something like :



[5.1.1] Residual block as illustrated in the original ResNet paper[14]

Each Residual block consists of a Convolutional (`Conv2D`) layer, a `BatchNormalization` layer (which helps the optimizer smooth out the gradients [15]), an Activation function (`ReLU`) and then again a Convolutional and another `BatchNormalization` layer.

After every Residual block is made, everything passes through an Activation function once more, which leads to two parallel streams that consequently form block layers.

This happens three times, and each time the inputs are downsampled and the number of filters, applied on the convolutional ResNet layer, is multiplied.

In the end, everything passes through a `MaxPooling2D` layer [2], then a `Flatten` layer to turn every image array into one long stream of data, and finally through a `Dense` layer which is activated using Softmax function in order to distinguish which class the image is more likely to belong in.

[2] Original ResNet paper has an AveragePooling2D layer [14]

5.2. ResNet20 Illustration

(Some blocks are omitted due to repetition and readability issues)

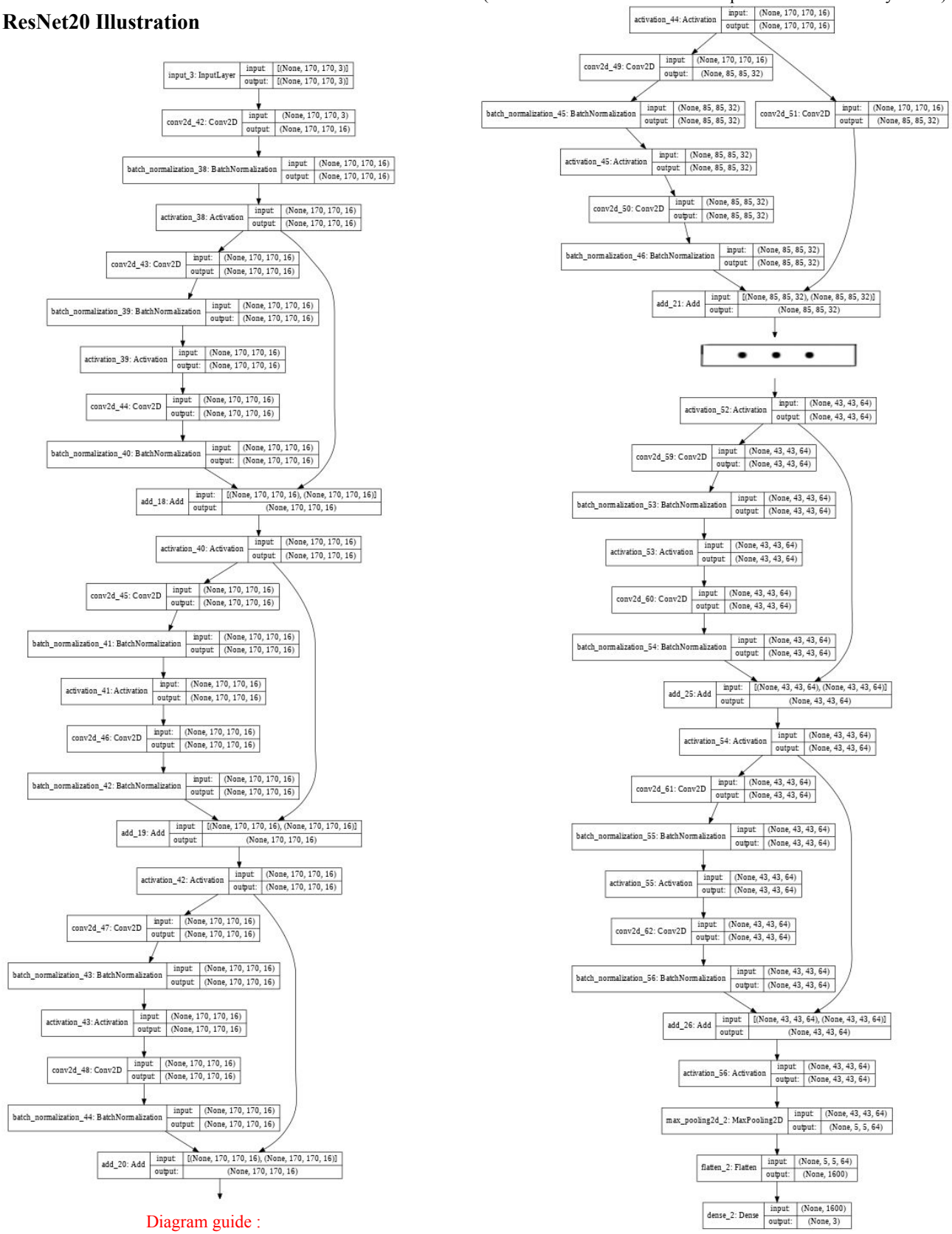


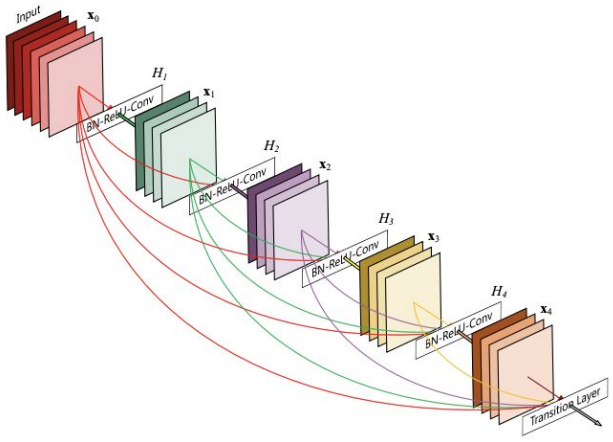
Diagram guide :
Top-Left to Bottom-Left then Top-Right to Bottom-Right

5.3. DenseNet20

DenseNet20 is a CNN with 3 Dense blocks, consisting of 5 Dense layers each. Every Dense block layer is followed by a Transition layer.

Each Transition layer consists of a `BatchNormalization` layer, an Activation function (`ReLU`), a Convolutional (`Conv2D`) layer, a Dropout layer and an `AveragePooling2D` layer. This also applies some downsampling to the images.

The same way ResNet uses Residual Blocks to minimize vanishing gradient, DenseNet concatenates each Convolution block with every other Convolutional block in the next Dense block .



[5.3.1] The DenseNet connectivity illustration as displayed in the original DenseNet paper [16]

An initial Convolution layer is created, and then the 3 Dense blocks and the Transition layers are made by it. In each iteration, the number of channels is multiplied with the compression rate to reduce the number of the feature maps in the next layer.

Everything gets normalized using `BatchNormalization`, then goes through an Activation function, (`ReLU`) and gets average pooled, using `GlobalAveragePooling2D`. Finally, the data goes through the last Dense layer with the Softmax function to distinguish which class the image is more likely to belong in.

5.4. DenseNet20 Illustration

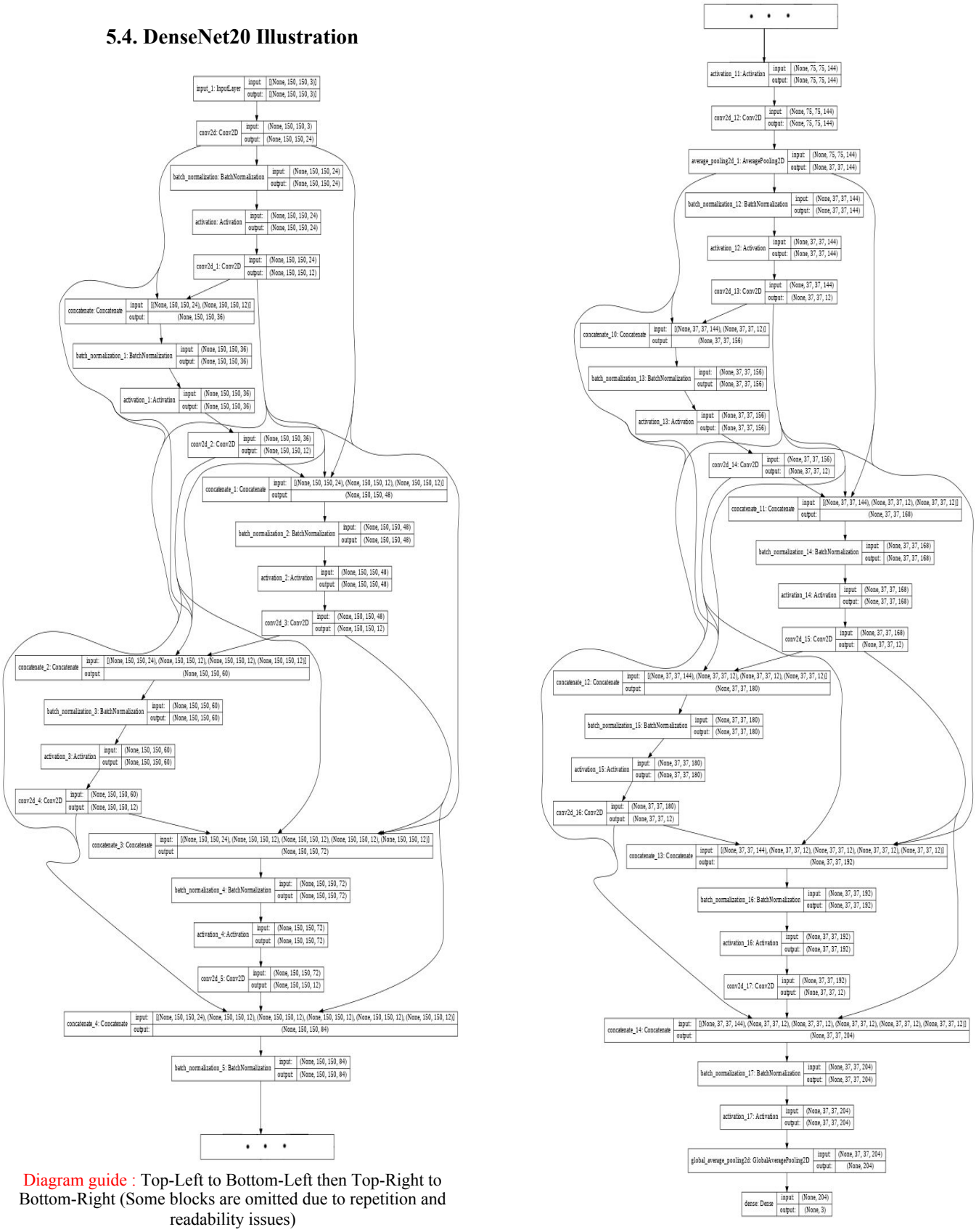


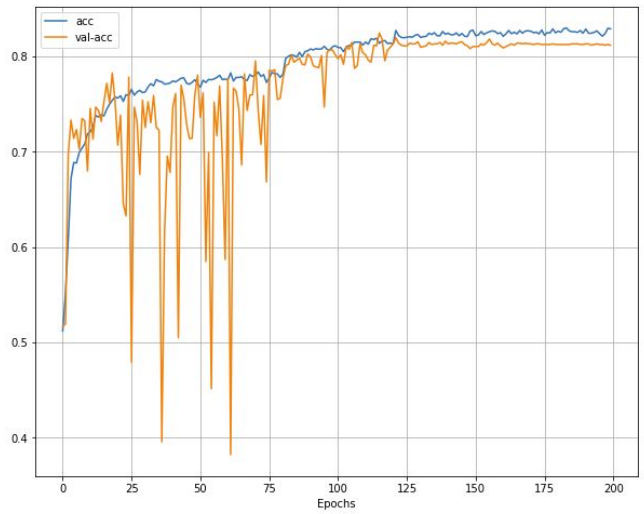
Diagram guide : Top-Left to Bottom-Left then Top-Right to Bottom-Right (Some blocks are omitted due to repetition and readability issues)

6. RESULTS

The results of the two models were really interesting. Both of the models managed to predict the classes of the images fairly well with an average 81% amount of predictions being correct.

6.1. ResNet20's Results

From what we can see on the image [6.1.1] below, the model did not overfit, almost at all, and managed to get an accuracy of 82.7% in the training set and a 81.8% accuracy in the validation set.

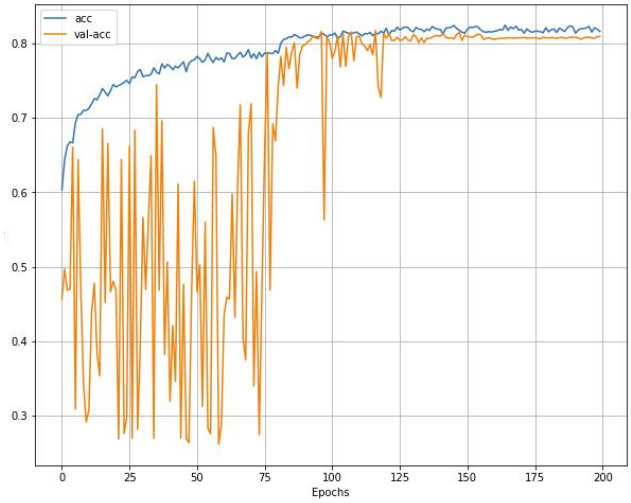


[6.1.1] ResNet's Accuracy and Validation Accuracy results

Although, in the first 75 epochs, validation accuracy is slightly fluttering up and down, it still improves on average and after that, both accuracies move towards the same direction almost parallelly, without any overfitting. At the end, our model stabilizes itself with no major changes. That happens because of the decreasing learning rate value being changed with the `lr_schedule`.

6.2. DenseNet20's Results

In the following graph [6.2.1], it is obvious that the same optimizers and other features don't work quite as well as on different models[3], but still the model managed to reach 81.3% Train accuracy and 80.8% Validation accuracy, which results in just a 0.5% difference.



[6.2.1] DenseNet's Accuracy and Validation Accuracy results

The `lr_schedule` changes its first value in the 80th epoch, which becomes evident as the validation accuracy starts to stabilize itself. This late stabilization might also be attributed to the use of the model's `Adam` optimizer, since Stochastic Gradient Descent might possibly produce better results if advised as used.[17]

[3] At first I thought my model was going to overfit, mainly because DenseNet's depth was just 20 but also because of all that fluttering of the Validation accuracy in the first epochs, even though the Train accuracy was increasing steadily.

7. REFERENCES

References that have been taken from academic papers or websites related to Machine Learning and Deep Learning

[1] Paul Mooney, Chest X-ray Images(Pneumonia), *kaggle.com*, 22 Mar 2018

[2] Madz2000, "Pneumonia Detection using CNN(92.6% Accuracy)" , *kaggle.com, Chest X-Ray Images (Pneumonia)* , July 2020.

[3] Konstantin S Sharov, "SARS-CoV-2-related pneumonia cases in pneumonia picture in Russia in March-May 2020: Secondary bacterial pneumonia and viral co-infections", *J Glob Health*, 2020 Dec; 10(2): 020504, Published online 2020 Aug 18. doi: 10.7189/jogh.10.-020504

[4] Sheikh Md Hanif Hossain, S M Raju and Amelia Ritahani Ismail, "Predicting Pneumonia and Region Detection from X-Ray Images using Deep Neural Network", *arXiv.org*, 19 Jan 2021

[5] Elaziz MA, Hosny KM, Salah A, Darwish MM, Lu S, Sahlol AT "New machine learning method for image-based diagnosis of COVID-19". *PLoS ONE 15(6): e0235187*, 2020.

[6] Weihan Zhang, Bryan Pogorelsky, Mark Loveland and Trevor Wolf, "*Classification of COVID-19 X-ray Images Using a Combination of Deep and Handcrafted Features", arXiv:2101.07866,* 21 Jan 2021

[7] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Robyn L. Ball, Curtis Langlotz, Katie Shpanskaya, Matthew P. Lungre and, Andrew Y. Ng, "CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning", *arXiv:1711.05225*, 25 Dec 2017

[8] dshahid380, "Convolutional Neural Network", *towardsdatascience.com,* Feb 24, 2019.

[9] Chollet, Francois and others, Keras, 2015.

[10] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015

[11] Scikit-learn: Machine Learning in Python, Pedregosa *et al.*, JMLR 12, pp. 2825-2830, 2011.

[12] T. Carneiro, R. V. Medeiros Da NóBrega, T. Nepomuceno, G. Bian, V. H. C. De Albuquerque and P. P. R. Filho, "Performance Analysis of Google Colaboratory as a Tool for Accelerating Deep Learning Applications," in *IEEE Access, vol. 6, pp. 61677-61685*, 2018, doi: 10.1109/ACCESS.2018.2874767.

[13] Sebastian Bock, Josef Goppold and Martin Weiß, "An improvement of the convergence proof of the ADAM-Optimizer ", *arXiv:1804.10587*, 27 Apr 2018

[14] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, "Deep Residual Learning for Image Recognition", *arXiv.org*, 10 Dec 2015.

[15] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, Aleksander Madry, "How Does Batch Normalization Help Optimization?", *Advances in Neural Information Processing Systems 31* (NeurIPS 2018).

[16] Gao Huang, Zhuang Liu and Laurens van der Maaten, *"*Densely Connected Convolutional Networks*"*, *arXiv.org*, Sun, 28 Jan 2018.

[17] Sanket Doshi , "Various Optimization Algorithms For Training Neural Network", *towardsdatascience.com* , 13 Jan 2019.