### Pneumonia Detection Using Deep Learning Models

*Iraklis Konsoulas*

[it185200@iee.ihu.gr](mailto:it185200@iee.ihu.gr) - [dyka3773@gmail.com](mailto:dyka3773@gmail.com)

Informatics and Electronics Engineer

International Hellenic University, Alexander Campus

57400 Sindos, Thessaloniki, Greece

January 2021

#### **ABSTRACT**

The abstract should appear at the top of the left-hand column of text, about 0.5 inch (12 mm) below the title area and no more than 3.125 inches (80 mm) in length. Leave a 0.5 inch (12 mm) space between the end of the abstract and the beginning of the main text. The abstract should contain about 100 to 150 words, and should be identical to the abstract text submitted electronically along with the paper cover sheet. All manuscripts must be in English, printed in black ink.

***Index Terms—*** One, two, three, four, five

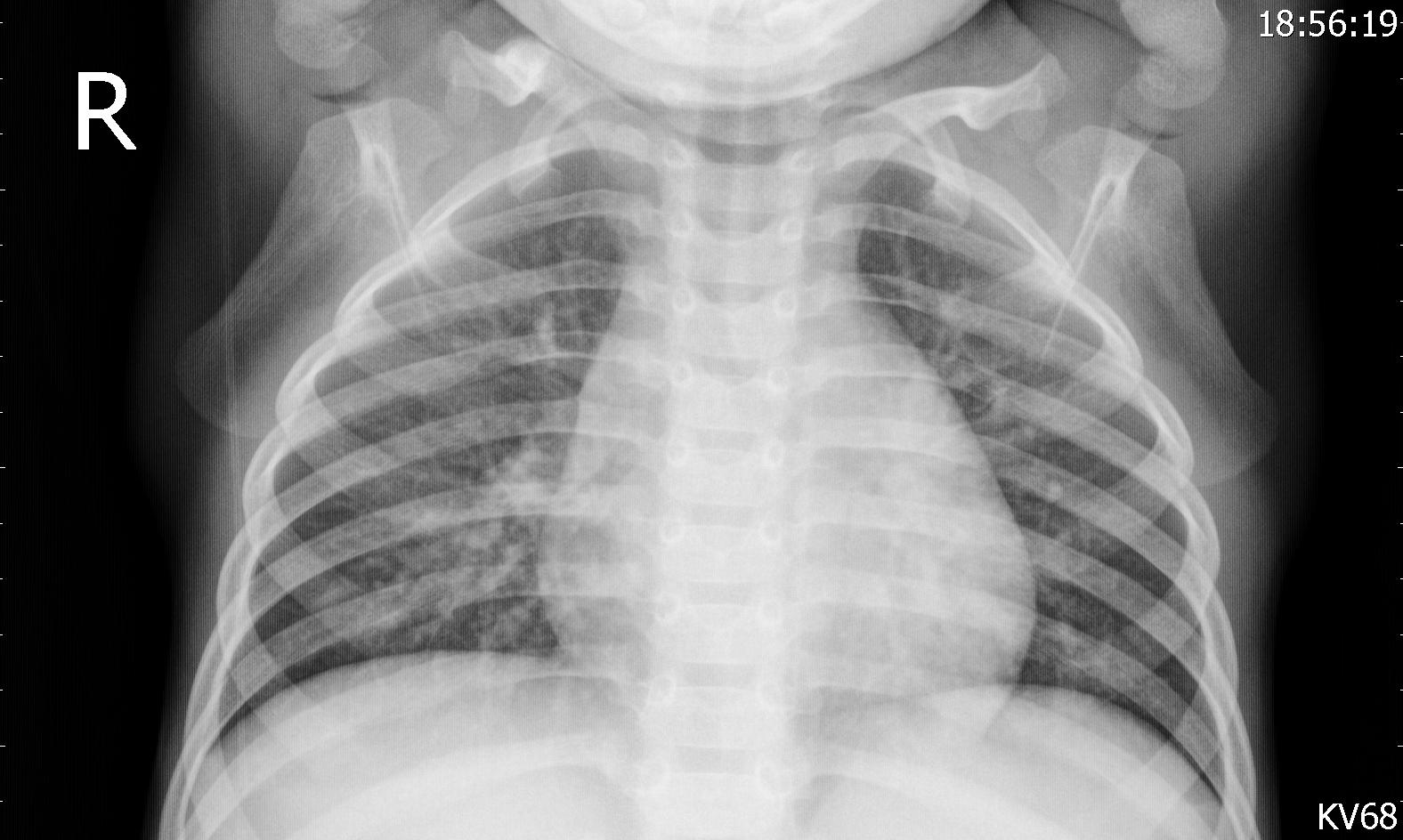
**1. INTRODUCTION**

These guidelines include complete descriptions of the fonts, spacing, and related information for producing your proceedings manuscripts. Please follow them and if you have any questions, direct them to Conference Management Services: Phone +1-979-846-6800 or email: papers@icassp2013.org.

**2. THE DATASET**

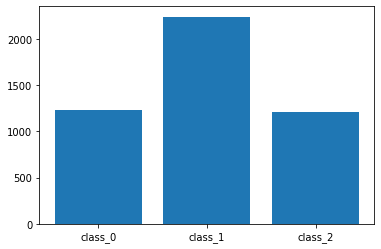
Our dataset consists of 4672 X-ray images classified according to patient’s condition :

* 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, I had to pre-process the images (More on paragraph [4.1] ).

As you can see in the image [2.2] below the distribution of the images is really imbalanced. So I had to use a data augmentation (See paragraph [4.4]) so my model will properly fit and avoid 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, I used Google Colab to write some Python notebooks one for each model I trained. I, also, used Keras, TensorFlow and SciKit-Learn libraries to help me write less but more efficient code. The complete list of the Python libraries that I 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

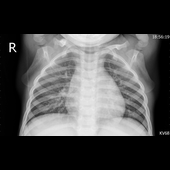
Both of the models trained, used an implementation of ResNet and DenseNet code in Keras and TF.

**4. PROGRAM WORKFLOW**

**4.1. Image Preprocessing**

After importing the images and their labels, all the images have to be resized and preprocessed to fit in the model (The processed images are in PIL format). After many experiments, 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 and it doesn’t deform the images.

The aspect ratio has changed to 1:1, because, due to the randomness of the ratios in the images, I decided to take the safe route and average them all up to squares. That caused me to fill every image with an offset border rectangle, which is black to match with the X-ray’s background. You can see the result below implemented on the example [2.1] image.



**[4.1.1**] The [2.1] image after its preprocessing

**4.2. Data Normalization**

First of all, the PIL images are turned into numpy arrays and every pixel is divided by 255, so that every value is between 0 and 1. The data are splitted in Train and Validation data using the `train\_test\_split`function to divide to a 70-30 ratio.

The arithmetic mean of the Train array is saved and subtracted from both the arrays. That way, every difference or feature an image has is way more visible in our program.

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

**4.3. Functions and Instantiations**

Two major functions are 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.
* `MyCallback`, which is a callback function that prints different model attributes on the screen.

**4.4. Data Augmentation**

As mentioned before ([2.2]), it is obvious that an `ImageDataGenerator` was needed. After researching other models built for X-ray imaging I decided that these changes on the images didn’t affect the class of the image :

* Rotating up to 30°
* Zoom by 20%
* Shift images horizontally and vertically by 10% of their width or height respectively
* Flipping horizontally

So the data generator was fitted using these parameters

**4.5. Model Fitting and Results**

Both models are trained using the same parameters. The batch size is 32 [[1]](#footnote-0), the epochs are 200 and the callback functions are :

* `lr\_schedule`
* `MyCallback`
* `ReduceOnPlateau`
* `ModelCheckpoint`

Models are evaluated and plots displaying their ‘accuracy’ and ‘validation accuracy’ are shown.

To use the model in actual unknown data the user has to Normalize and Preprocess their data the same way I did and preferably use the same arithmetic mean value to have the best results.

**5. MODEL DESCRIPTIONS**

Two models were trained to classify the given dataset :

* ResNet20
* DenseNet40

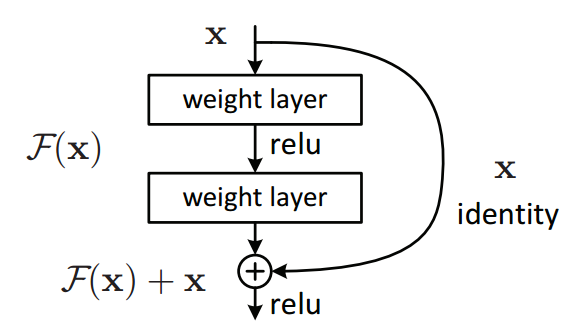
The most successful one being ResNet20 as expected.

I chose these two, because they are some of the best and most well-known Convolutional Neural Nets for image classification.

**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 helps 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[1]

After every Residual block is made,everything passes through an Activation function again, whose inputs are the two parallel, now, block layers.

This happens three times, where each one downsamples the inputs and multiplies the number of filters applied on the convolutional ResNet layer.

In the end, everything passes through a `MaxPooling2D` layer [[2]](#footnote-1), a `Flatten` layer to turn every image array into one long stream of data and finally through a `Dense` layer that is activated using Softmax function to distinguish which class the image is more likely to belong in.

**5.2. ResNet20 Illustration**

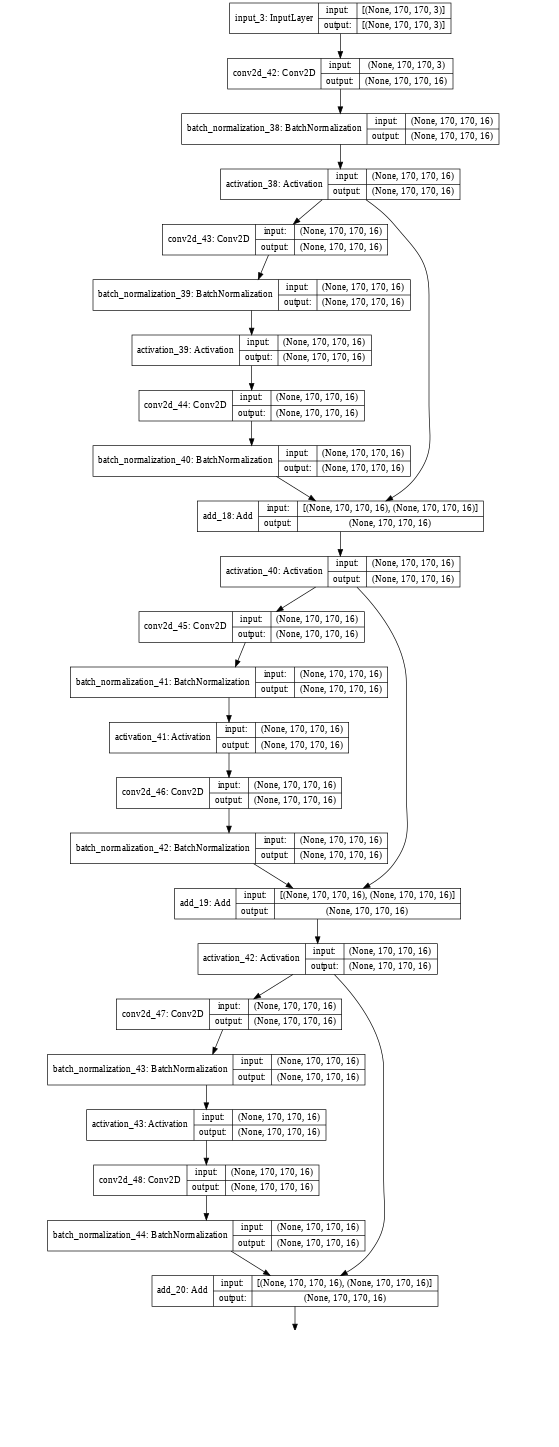
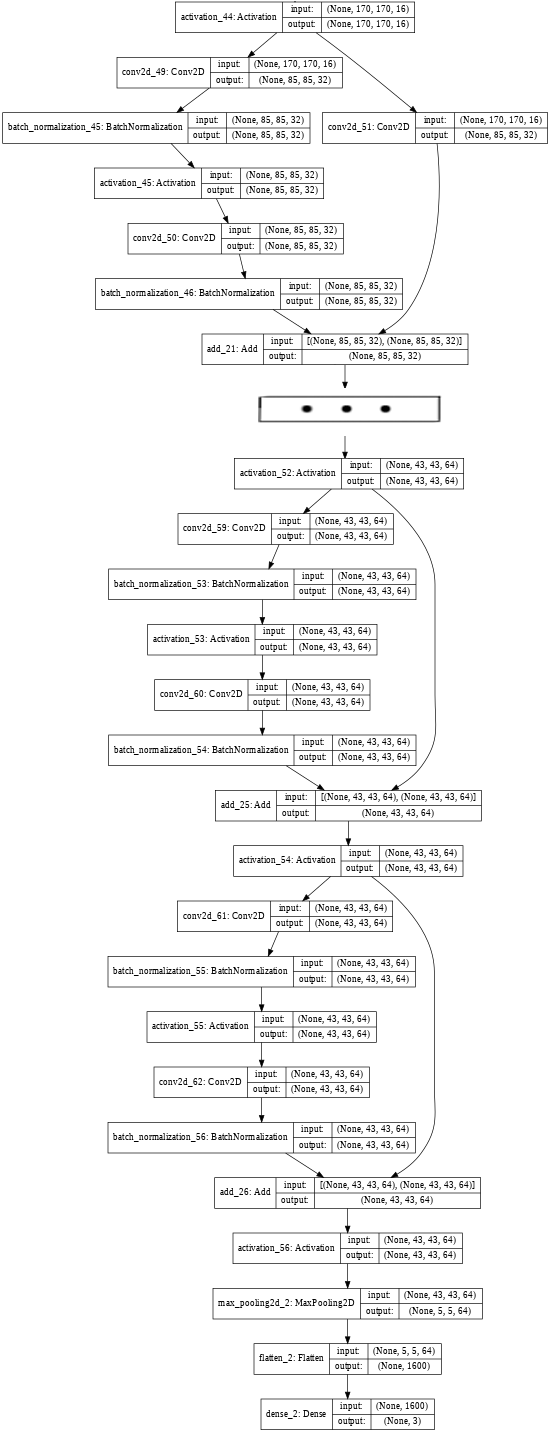
****

Diagram guide :

Top-Left to Bottom-Left then Top-Right to Bottom-Right

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

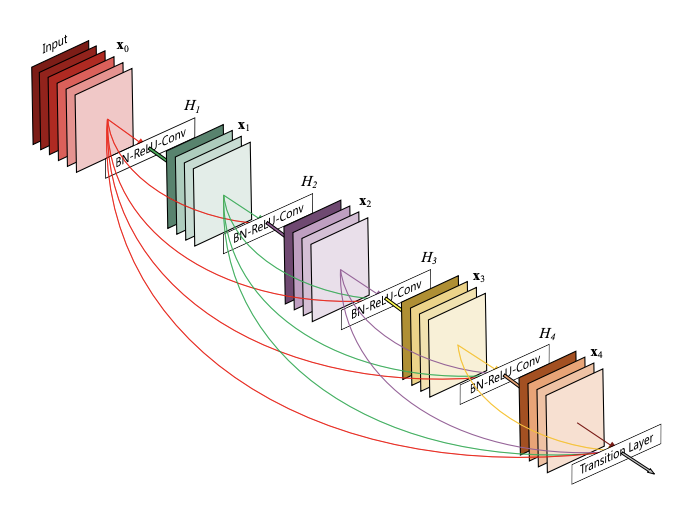
****

**5.3. DenseNet40**

DenseNet40 is a CNN with 3 Dense blocks, consisting of 5 Dense layers each. Every Dense block is followed by a Transition layer. An initial Convolution layer is created.

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 does some downsampling to the images too.

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 [2]

An initial Convolution layer is created, and then the 3 Dense blocks and the Transition layers are made out of 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.

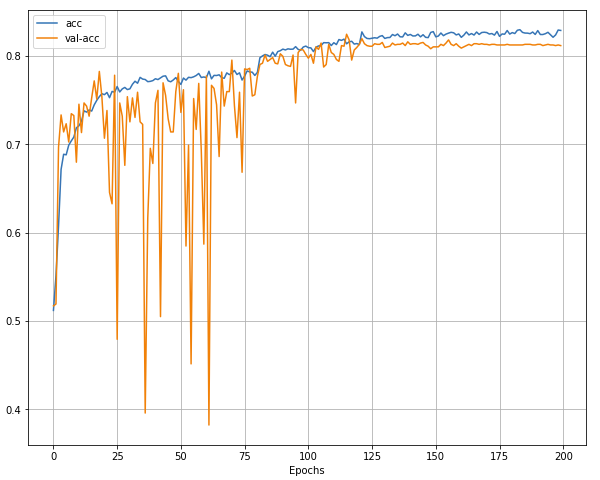
**5.4. DenseNet40 Illustration**

**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 80% 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

**7. REFERENCES**

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

**7.1. In-text Related References**

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun, “[Deep Residual Learning for Image Recognition](https://arxiv.org/pdf/1512.03385.pdf),” *arXiv.org*, 10 Dec 2015.

[2] Gao Huang, Zhuang Liu and Laurens van der Maaten, *“*[*Densely Connected Convolutional Networks*](https://arxiv.org/pdf/1608.06993.pdf)*”*, *arXiv.org*, Sun, 28 Jan 2018.

**7.2. Out-of-text Related References**

* Madz2000, “[Pneumonia Detection using CNN(92.6% Accuracy](https://www.kaggle.com/madz2000/pneumonia-detection-using-cnn-92-6-accuracy))” , *kaggle.com,* [*Chest X-Ray Images (Pneumonia)*](https://www.kaggle.com/paultimothymooney/chest-xray-pneumonia), July 2020

1. Original ResNet paper has a 32 batch size [1] [↑](#footnote-ref-0)
2. Original ResNet paper has an AveragePooling2D layer [1] [↑](#footnote-ref-1)