Deep Learning Competition

COVID-19 and Pneumonia Diagnosis

Team 19

Mirna Muhammad
Computer Science
2016170450

Menna Mohie El-deen
Computer Science
2016170438

Eman Alaa Farag
Computer Science
2016170113

Esraa Yasser
Computer Science
2016170080

Menna-Allah Mostafa Mostafa Awad
Computer Science
2016170437

ABSTRACT

This research aims to come up with an artificial neural network that accurately classifies images of patients who have COVID-19. In the recent events the world is dealing with the coronavirus pandemic, hence, it's crucial to be able to diagnose people who show symptoms of the virus.

In our research, we tried to apply most of the deep-learning techniques that we acquired throughout the course of Artificial Neural Networks. The best-achieved results of our trials scored 86.5% on Kaggle's public score for the competition and 84% for the private score.

1. INTRODUCTION

COVID-19 and Pneumonia present several unique features. While the diagnosis of COVID-19 is confirmed using polymerase chain reaction (PCR), infected patients with pneumonia may present on chest X-ray and computed tomography (CT) images with a pattern that is only moderately characteristic for the human eye.

COVID-19's rate of transmission depends on our capacity to reliably identify infected patients with a low rate of false negatives. In addition, a low rate of false positives is required to avoid further increasing the burden on the healthcare system by unnecessarily exposing patients to quarantine if that is not required. Along with proper infection control, it is evident that timely detection of the disease would enable the implementation of all the supportive care required by patients affected by COVID-19 and Pneumonia.

2. DATASET

The dataset consists of 5,266 X-Ray training images, 1,341 Normal X-Ray and 3,925 (COVID-19 and Pneumonia) images.

The test set contains 690 of unlabeled images.

3. PREPROCESSING

Since one of the classes had less data than the other, we decided that the best option was to augment this class with noise. Also, some of the images were duplicated, so the best option was to delete these as well before augmenting the data.

3.1 Data Augmentation:

1. We created a python jupyter notebook and used these libraries: cv2, NumPy and glob.
2. Then we read the dataset into 2 lists, training set: normal and sick images using cv2 library and concatenated them into a list.
3. We created the y_train list which contained the labels for each image. The list contained zeroes corresponding to normal images and then ones equal for the sick images.
4. For each image in the list, the noise was added to it as explained in Section 3.2 and appended into augmented_x_train alongside its label in the augmented_y_train list.
5. Finally, these 2 lists were extended with the original dataset lists.

3.2 Noise addition:

The noise type was gaussian noise. Gaussian noise variables are: mean = 0, sigma = $100^{0.5}$.

Gaussian noise was created using the NumPy library random method where its parameters were mean, sigma and the size of the image. The augmented image was created by adding the noise to the original image. Then the image was normalized to be within (0,255) using the cv2 library.

3.3 Dataset Splitting

The training data were split into train/dev sets. The training set contained 2600 of the images (1300 of each class). The dev set contained the remaining 2666 images, which were mainly COVID-19 images.

3.4 Color Scale and Spatial Dimensions

For the models mentioned in Section 4.1, images were converted to grey-scale and resized to be of spatial dimensions (112x112).

For the models mentioned in Section 4.2, images were converted to grey-scale and resized to be of spatial dimensions (224x224).

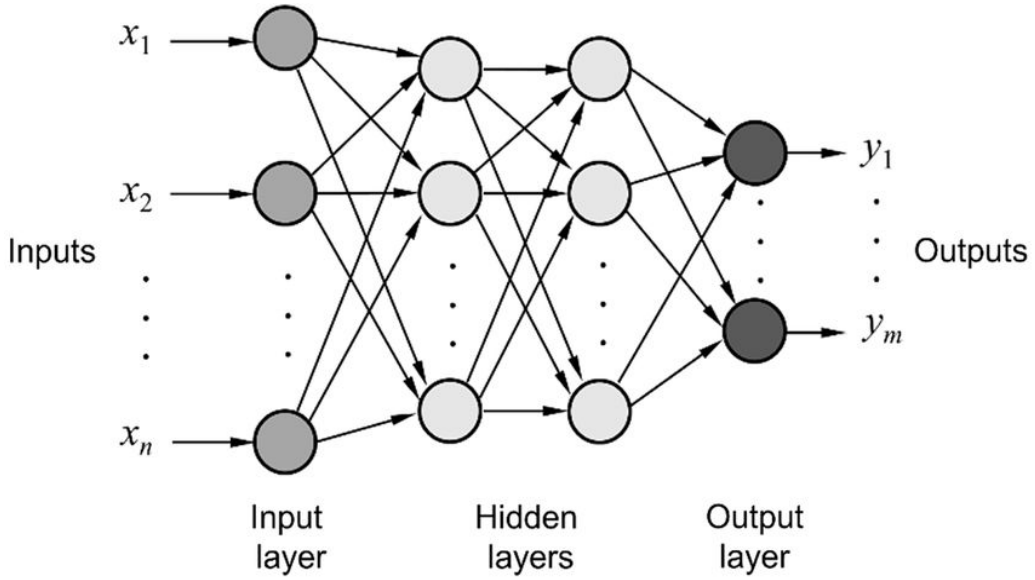
For the models mentioned in Section 4.3, images were converted to RGB-scale and resized to be of spatial dimensions (224x224).

4. METHODS

Google collaboratory platform was used for its support of GPU resources and 25GB ram was claimed. All the training and testing was done using google colab. Keras and Tensorflow were used for the implementation of some of the models mentioned in the subsections of Section 4.

4.1 Multilayer Perceptron Model(s)

Sequential models or multilayer perceptrons were implemented with different hyperparameters for the aim of improving the achieved accuracies.



The hyperparameters that were subject to change were the number of hidden layers, number of neurons in the hidden layers, optimization algorithm, number of iterations used for training, normalization technique in weight initialization[1], batch size if applicable. The activation functions were ReLU for the hidden layers and Sigmoid for the output layer. The default setting for the learning rate was 0.0075. Table 1 summarizes the different MLP implementations.

Table 1
Hyperparameters summary for the MLP trials

Model Number	Layers Dimensions	Number of Iterations	Batch Size	Optimization Algorithm	Initialization with HE
1	[?, 7, 7, 5, 2, 1]	3000	NA	Gradient Descent	Yes
2	[?, 20, 5, 1]	4900	NA	Gradient Descent	Yes
3	[?, 7, 7, 5, 2, 1]	3000	NA	Gradient Descent	No
4	[?, 10, 5, 1]	3000	NA	Gradient Descent	Yes
5	[?, 20, 5, 1]	3000	NA	Gradient Descent	Yes
6	[?, 5, 2, 1]	1000	512	Adam	Yes
7	[?, 10, 5, 1]	1000	256	Adam	Yes

[1] He Normal (He-et-al) Initialization was implemented and used for some of the MLP models instead of initializing with randomly distributed data. (https://medium.com/@prateekvishnu/xavier-and-he-normal-he-et-al-initialization-8e3d7a087528)

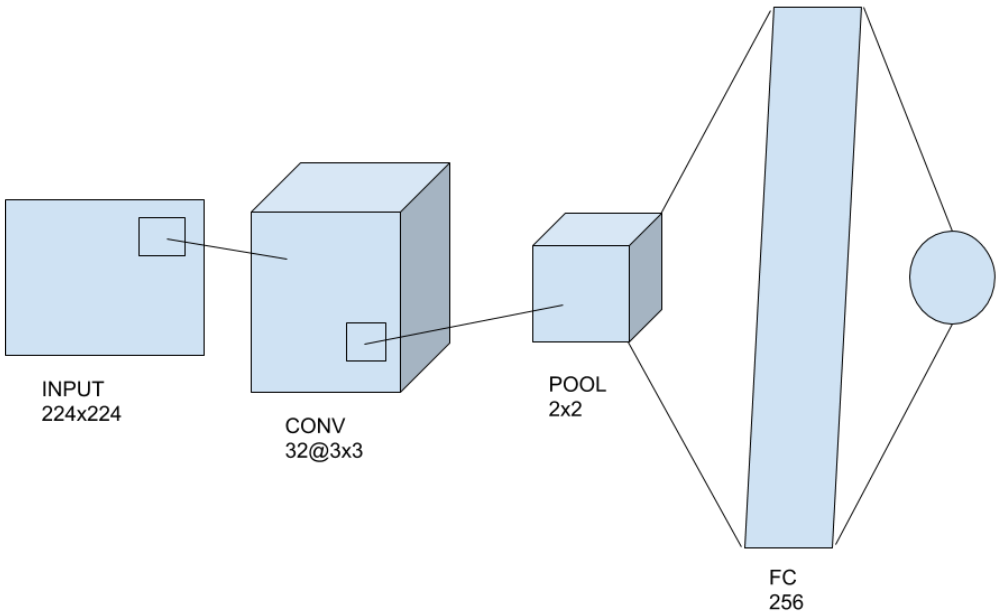
8	[?, 5, 10, 10, 5, 1]	1000	512	Adam	Yes
9	[?, 36, 6, 1]	800	256	Adam	Yes
10	[?, 36, 6, 1]	40	512	Adam	NA

In Table 1, Model Number 10, the architecture also differed from the other models as a dropout layer was added after the first hidden layer with dropout probability of 20%.

4.2 Convolutional Neural Network Model(s)

Since the task at hand is to classify images, a 2D CNN architecture was convenient to implement. The implemented 2D CNN have the following summary:

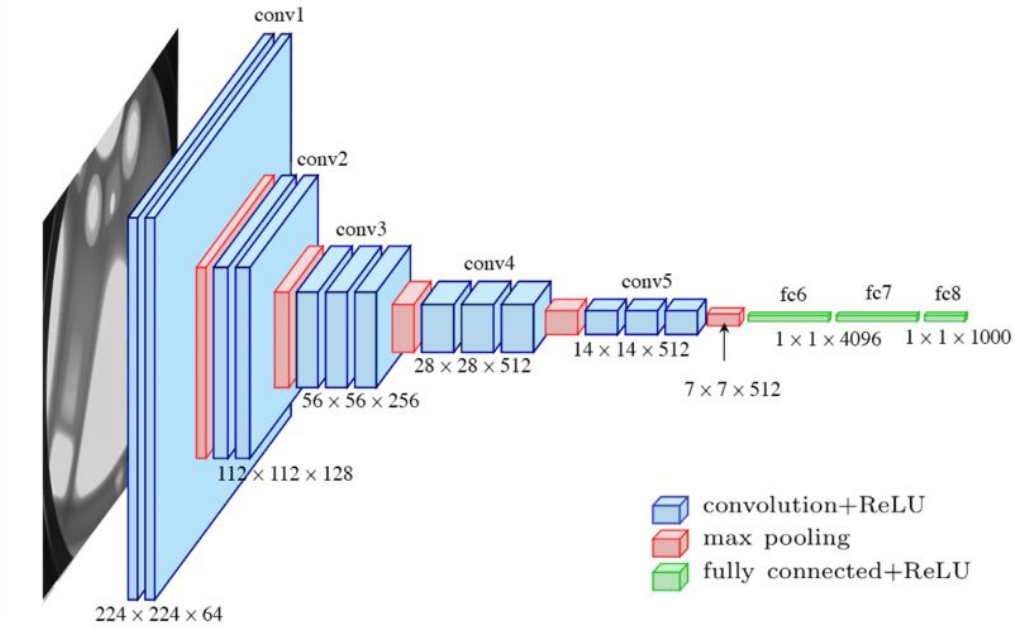
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_2 (Conv2D)            (None, 222, 222, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 111, 111, 32)      0
_________________________________________________________________
flatten (Flatten)            (None, 394272)            0
_________________________________________________________________
dense (Dense)                (None, 256)               100933888
_________________________________________________________________
dropout (Dropout)            (None, 256)               0
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 257
=================================================================
Total params: 100,934,465
Trainable params: 100,934,465
Non-trainable params: 0
```



This model was compiled using Adam optimization algorithm, binary cross-entropy as its loss function. The dropout layer used has a drop-out probability of 30%. The output layer has Sigmoid as its activation function. The rest of the network uses ReLU.

4.3 Transfer Learning

The pre-trained architecture used was VGG-16. It is a simpler architecture model since it's not using many hyperparameters. It always uses 3 x 3 filters with a stride of 1 in convolution layer and uses SAME padding in pooling layers 2 x 2 with a stride of 2.



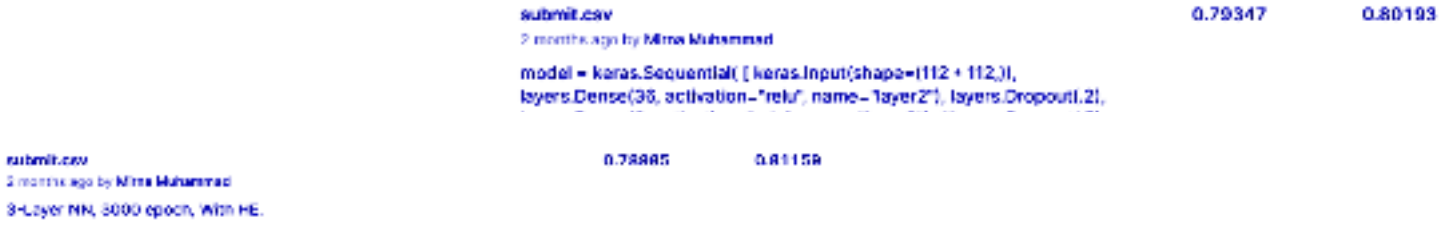
5. RESULTS[2]

As mentioned in Section 4, multiple trials were made with different architectures and sometimes changes to the hyperparameters of the same architecture. Listed in the sections below the results of each subsection exhibited in Section 4.

The metric used for judging the performance of the models is the accuracy metric. The accuracies used for insight are of four categories, the model accuracy on the train set, the dev set if applicable and the Public and Privates scores from Kaggle.

5.1 Multilayer Perceptron Model(s)

Table 2 contains a summary of each model and the corresponding accuracies in the four mentioned categories. As seen in Table 2, if we were to judge the model's performance based on the public score then Model Number 4 outperforms the rest by accuracy 81.15%, but if we judge using the private score then Model Number 10 scores the best performance which is 79.35%.



[2] Some of the results were not recorded since multiple runs were sometimes conducted at once, hence the existence of empty fields.

Table 2
MLP models accuracies results

Model Number	Train Set Accuracy	Dev Set Accuracy	Public Score	Private Score
1	97.80%	95.49%	78.50%	75.72%
2	97.70%	96.30%	76.80%	73.10%
3	86.57%	98.64%	71.98%	68.11%
4	95.60%	92.08%	**81.15%**	78.98%
5	96.50%	97.07%	76.81%	71.74%
6	99.65%	95.00%	76.80%	74.28%
7	100%	94.48%	77.78%	75.72%
8	99.60%	94.60%	78.26%	74.63%
9	100%	95.12%	-	-
10	-	-	80.01%	**79.35%**

5.2 Convolutional Neural Network Model(s)

The proposed architecture was trained multiple times with different batch sizes and in some trials using the same batch size but only the initial randomized weights being different. Only some of the trials were submitted to the Kaggle competition, those that scored the best dev set accuracy. The Kaggle public and private scores can be seen in Table 3.

Table 3
2D CNN model accuracies results

Submitted Trial Number	Public Score	Private Score
1	82.85%	**83.33%**
2	82.37%	82.97%
3	81.40%	79.71%
4	**84.54%**	**83.33%**

During the trials, the overall layers architecture was kept the same but changes were made to the number of units existing in the Fully-connected layer.

The best-submitted trial has a public score of 84.54% and a private score of 83.33%.

5.3 Transfer Learning

Different trials were conducted by changing the batch size of the data. Also, the model was fitted once without splitting the dataset into train/dev sets. The batch sizes used were 8, 32 and 64. The summary of the trials is found in Table 4.

Table 4

VGG-16 trials accuracies results

Submitted Trial Number	Train Set Accuracy	Dev Set Accuracy	Public Score	Private Score
1*	-	-	78.99%	79.71%
2	-	-	84.29%	82.97%
3	99.4%	95.6%	**86.47%**	**84.06%**
4	98.9%	94.7%	81.64%	81.88%
5	97.62%	96.06%	80.19%	77.17%

1*: in this trial, the model was trained without splitting the training dataset.

From Table 4, the best performance was acquired in trial number 3, the settings of this trial were as follows: the batch size was 32, and the number of epochs was 10.

Trials 4 and 5 used the same settings of batch size equal to 64. But in trial 5 the weights were from a checkpoint made on the model. The checkpoint kept the weight of the epoch in which the validation loss was minimum. As seen from the results, the best weights do not always entirely guarantee the best performance.

6. CONCLUSION

In the beginning of the project from the limitations that we faced was when we tried to augment the data as described in Sections 3.1 and 3.2. We processed the dataset locally, but when we tried uploading the new dataset it was too large and we weren't able to upload it, so in the end, we had to skip the augmented data and work with the original one.

In order to balance the samples, we made the train/dev sets splitting. As seen in the performance of the VGG-16, the splitting outperformed working with the train-set as a whole.

The hypothesis that the convolutional networks perform better than MLP networks were proven to be true according to our trials. As we've seen from the result in Tables 2 and 3. A simple convolutional neural network achieved accuracies of 84.54% and 83.33% instead of the highest achieved accuracy in the MLP architectures of 81.15%.

Transfer learning for image classification surpassed the previously acquired results. VGG-16 is a powerful and simple model and we've only applied one of the most famous architectures published.