Making a Convolutional Neural Network
COVID-19 CT Image Classifier

a project authored by

Aidan Finkler, Sidney Finkler, Alex H, Kaylee Kelly, Jacob Moesner,

Nathan Pan, Aaban Syed, Imaad Syed

supervised by

Mr. Steven Griffin

29 July, 2020

Abstract

In this project, MISI interns developed a convolutional neural network (CNN) class to classify patient lung CT-scans as either Covid-19 positive or negative. The main goal was to learn about convolutional neural network layers and features, in addition to gain familiarity with developing CNNs. In this project, multiple classes and train methods were created, in addition to data augmentation scripts. After accomplishing the main goals, data was augmented to produce more training data, to produce a more accurate model. Transfer learning was also done, but the accuracy of all models were less accurate than the original COVID-19 image classifier. However, the main goals, in addition to many others, such as transfer learning, data transfer and an executable script for use were all accomplished.

Contents

Acronyms

CNN Convolutional Neural Network

CPU Central Processing Unit

CT Computed Tomography

GPU Graphics Processing Unit

ReLU Rectified Linear Unit

Chapter 1

Introduction

We are members of AI Team 5 for the 2020 Maryland Innovation and Security Institute Virtual Internship. The overall artificial intelligence project revolved around improving the UCSD-AI4H COVID-CT GitHub project. This GitHub project is a COVID-19 image classifier, using lung CT-scans as an input. As a part of Team 5, we were given sub-tasks to complete with the main goal of trying to improve this project.

1.1 Problem Statement

In order to combat the COVID-19 pandemic, robust testing is needed to be able to identify COVID-19 hotspots and respond accordingly. Current COVID-19 tests are short in supply and take too long to identify if the patient is COVID-19 positive or not. To solve this problem, we aim to develop a Convolutional Neural Network that can use patient lung CT scans to classify patients as either COVID-19 positive or negative. By doing so, we hope that this neural network can be used to efficiently and effectively combat the COVID-19 pandemic.

1.2 Objectives

The official objectives given to us were to:

- Learn about the options for layers

- Make a new CNN class model to gain familiarity

However, after we finished these tasks, we took on additional challenges. They were the following:

- Create a train method

- Create a new data loader script

- Create a new testing script

- Expand our dataset through image augmentation

- Implement transfer learning

1.3   Overview

In our project, we constructed and trained a new convolutional neural network (CNN) to classify non-COVID-19 and COVID-19 lung CT scans. The images were given as inputs, and the output tensor gave values for COVID and non COVID. The larger value of the parts of the tensor indicated what the network classified the image as. To make the construction of the CNN easier, we used the Pytorch library. We also selected Pytorch, as it allows for GPU-accelerated training. CNNs were built using the Pytorch framework.

Our task was to create a CNN class and explore layers. We quickly found that more than 7 total layers (5 hidden) posed to be too complex for our dataset, and far too expensive to train (training time rose very quickly with new layers). Additionally, we experimented with the number of nodes in layers, finding out which models were too simple, and which were too complex and resource-inefficient. After completing our given task of creating a new CNN class and experimenting with it, we noticed that our model was very inaccurate. We had only trained the new class on 588 images.

After using the raw COVID-19 lung CT-scan database of 588 images to train our network, we looked towards data engineering. In order to get more data to train our network,

we rotated images to random angles, re-iterating through the same dataset, creating "new" images of COVID and Non-COVID CT scans. Additionally, we tried blurring and shifting images.

After our experimentation with data augmentation, we decided to implement transfer learning. We chose the Resnet-18 model, and modified the last layer of the artificial neural network (ANN) to 2 outputs.

1.4 A Quick Overview of Convolutional Neural Networks

A neural network consists of layers and weights. Layers themselves consist of nodes (or neurons). The first layer is the "input" layer, and the final layer is the "output" layer. The layers in between are "hidden" layers, adding complexity to the model. When a network has 2 or more hidden layers, then it is considered to be a "deep" neural network, or a part of deep learning. The weights can be thought of as the lines connecting nodes of one layer to nodes of the next layer. These weights change, giving a weighted input to activate certain neurons, which give the highest value to the desired output neuron, when trained. There are also biases attached to the weights, adding or subtracting a certain amount. These weights and biases help in identifying patterns within the input images.
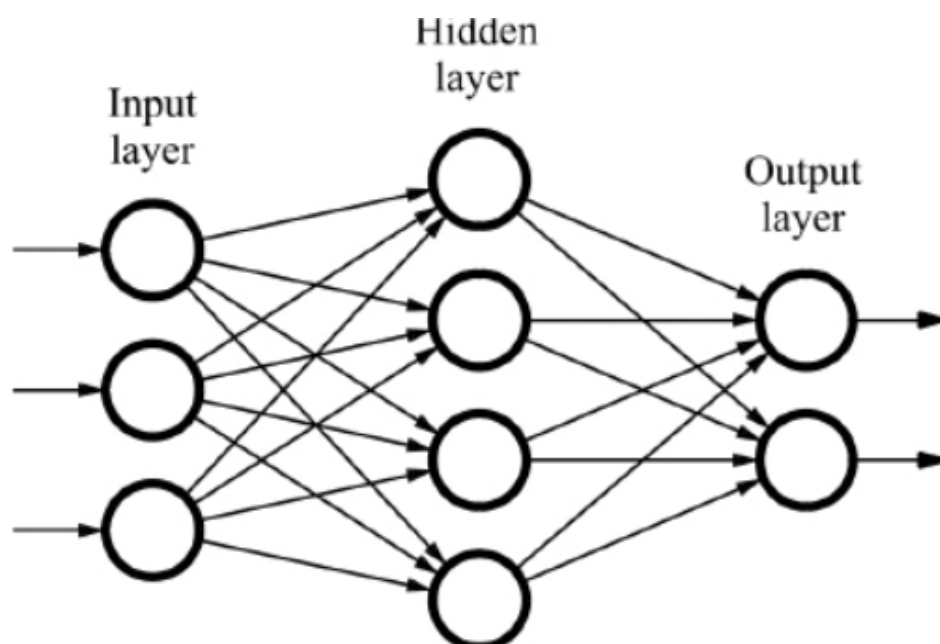
The value of the next neuron is calculated by multiplying the weights by the input values of the previous layers' neurons, inputted into an activation function. Activation functions can be sigmoids, tanh or rectified linear units (ReLU). Sigmoids are logistic functions, outputting a value between 0 and 1, tanh outputs a value between -1 and 1 and ReLU functions are piecewise linear functions, outputting the input when the input is above 1, and either 0 or a negative number when the input is 0 or less. This process of calculating outputs continues to the output layer. The equation follows this form:

$$A(\Sigma(w * i + b)) = Output$$

In order to train the network, weights are adjusted to activate the right neurons. Let us consider the weights to be an n-dimensional vector. Each dimension is a weight connecting from the current node to the previous layers' neurons. To train the network, we calculate a "loss". This is an error value, calculated by the following:

$$Loss = |Desired^2 - Actual^2|$$

The idea is to achieve the lowest possible loss. Zero is ideal, but usually impossible, so a very low value (0.001 or less) is usually desired. Training starts from the output layer, tracing back to the input layer weights. This process is called "back propagation". In order to modify the weights effectively, we want to approach a minimum loss, as a function of the weights. Recall the n-dimensional weight vector. The loss function contains the actual output value, which is a function of this weight vector. To approach a minimum, the gradient of the function is taken. Recall that the gradient vector points towards the direction of steepest ascent (local maximum), so subtracting this vector allows for the weights to approach a minimum loss. This gradient is multiplied by the scalar l, which is the learning rate. This changes the weight of the gradient, affecting the speed and accuracy of training. 0.001 is usually set as the learning rate. Below is the weight adjustment function:

$$\begin{bmatrix} w \\ ... \end{bmatrix} - l * \nabla Loss = \begin{bmatrix} new \\ ... \end{bmatrix}$$
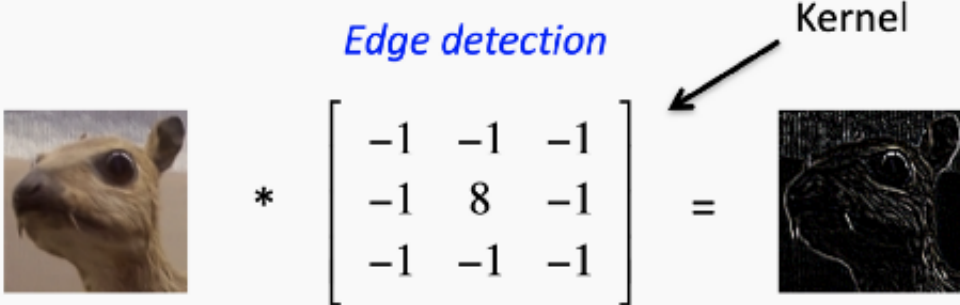
A convolutional neural network is a unique type of artificial neural network that is much better at image classification. This is due to the convolution filter this network applies that makes it easier to recognize shapes and patterns. The convolution filter works

6

by moving (convolving) a feature filter across a grid of input pixels. Each time it covers a select portion of the input grid, the values in the input grid are multiplied with the feature filter. This process is known as matrix multiplication.

Through this process of matrix multiplication, the network can detect a variety of features depending on the filter. For example, in the image shown below two different feature filters are applied to the animal (Note: By "applied", this means that the feature filter was multiplied to the input pixels). The "Edge Detection" feature filter allowed for the edges of the animal to be detected, while the "Sharpen" feature filter sharpened the image.

After going through the convolutional layer, the new output of that layer is fed into an activation function (Described Previously). After going through the activation function, the new output undergoes the pooling process. Pooling helps in highlighting the importance of features, as well as reducing the dimensions of feature maps. In the image below, a 2x2 pooling filter is applied to the 4x4 input matrix. Oftentimes, the Max Pooling layer is used to highlight the features the most.

The architecture for a CNN is as follows: input, padding, convolution and activa-

Rectified feature map

1	4	2	7
2	6	8	5
3	4	0	7
1	2	3	1

max pooling with 2x2 filters
and stride 2

Pooled feature map

6	8
4	7

Max(3, 4, 1, 2) = 4
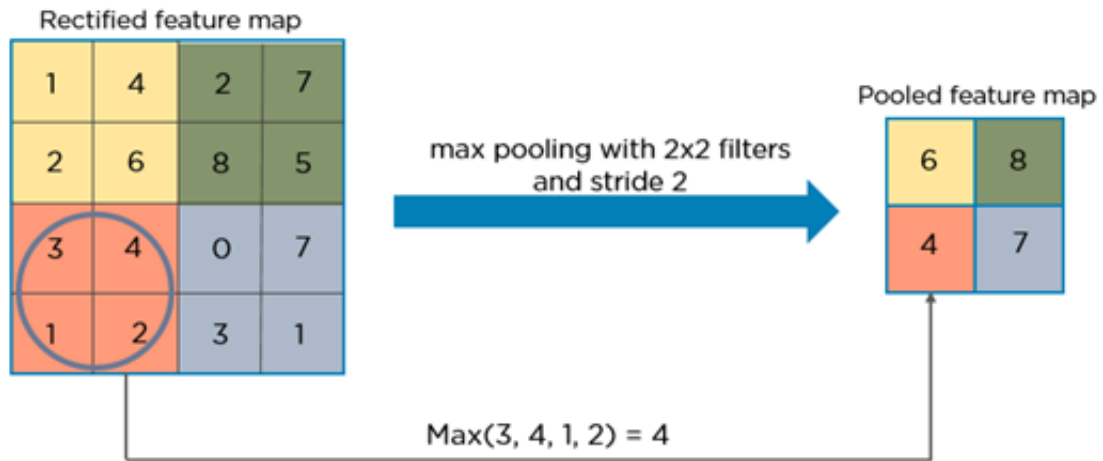
tion/Relu, pooling, flatten/dense, and fully connected/softmax. The convolution and pooling operations happen on the image, outputting a tensor (essentially an array). This tensor is then fed into the fully-connected layers (essentially an artificial neural network), and the output is values corresponding to each output node.

Chapter 2

Methods

Everything in this project was written in Python 3. The main libraries we used were Pytorch, in addition to many of its sub-libraries. We initially started working on GitHub, but quickly transferred our files to a GPU instance server, as it was much more convenient for running our scripts.

2.1 Classifier Model and Training

We created a CNN class module using Pytorch. The CNN was set up so that the input CT scans would go through the following sequence: Convolution, Max Pooling, Convolution, Max Pooling and then three Fully Connected Linear Layers. While creating our CNN class, we experimented with layers, by changing the number of nodes and layers. We settled on 3 layers, as we didn't have much training data, and training a classifier with more than 3 layers was too time-expensive. We settled on a model which took in a 224x224 black and white image.

We also developed our own training methods, as suggested by Pytorch tutorials. The training methods all performed the gradient descent operations to tune the weights, as well as provided statistics on the loss and accuracy in the current training batch. Several hundred epochs (generally 700) were used, with small batch sizes (12 for the small data-set, 65 for the large one).

2.2 Data Handling

To properly train our classifier, we created custom data csv files, which stored the image index, image path, image name (optional) and COVID-19 status (indicated by 1 or 0, 1 being positive, 0 being negative). The dataLoader.py file simply provided data structure classes to use in the main file, CNNv3.py. The data class took in a csv file, searched the appropriate columns in the file and returned the image converted into a tensor, in addition to the COVID-19 status.

We started on a 588 image dataset, and later added image augmentation to produce more "new" data. The EngineeredData class of the dataLoader.py file used a random rotation script (rotated the image from 0-300 degrees, randomly), blurring script and translation script. The different augmentations were selected with an additional parameter. 1 was the argument used for flipping the image, 2 used for rotation, 3 for translation and 5 for blurring. After some training, we saw some slight improvements, so we moved onto an image-set containing over 3000 CT scans to get better results.

2.3 Testing

To test our models after training, a testing script was created. The testing script is essentially the training script, but with different statistics returned, as well as lacking the gradient descent operations. The test script sets the model to "eval" instead of "train", comparing the data from the test data csv file to the output from the model. True Positives, True Negatives, False Positives and False Negatives are tracked, as well as percentages showing accuracy and the rates of the true/false positives and negatives.

A much smaller data-set was used for testing, as we needed as many images as possible to train the classifier. Testing, as well as training were all done on the GPU instance, although only the training method required GPU acceleration. The test script creates a blank model and then loads in a pretrained model. After evaluation, models were stored in the models folder on the GPU instance. We were unable to upload the models to GitHub, as the file sizes were over 700 MB (GitHub supports 25 MB only).

2.4 Transfer Learning and Extra Features

Nearing the end of our project, we engaged in transfer learning, but since this wasn't a priority, we ran it on the smaller data-set. The Resnet18 image classifier was used, with the final layer being modified to have only two outputs. The idea was to take a successful classifier (Resnet18), and apply it to the COVID-19 data-set.

Additionally, we created an "executable" file. It is just a python file, which takes in text input of an image directory, and outputs the trained model's results. This is a user-friendly UI, but not a real app. It is a housing for our model, and a more polished product than the CNN files and test scripts.

Chapter 3

Results

The following sections will detail the results of different training datasets on the CNN performance. We had four different training datasets; the first had 588 images and no augmentations, the second had 588 images and rotations only, the third had 588 images and all augmentations, and the fourth had a large dataset and rotations only.

3.1   Round 1: 588 Images, No Augmentations

For our first round of testing, we used the small dataset with no image augmentation. The model was quite inaccurate, with a very high false negative rate. The testing results are as shown:

Accuracy: 55.79%

Correct: 53

Total: 95

True positives: 2

True negatives: 51

False positives: 16

False negatives: 26

False positive rate: 23.88%

False negative rate: 92.86%

3.2 Round 2: 588 Images, Rotations Only

For our second model, we increased the epochs to 700, and used the rotation script to augment data. The results were "better", however the false negative rate was the exact same as the previous model. The testing results are as shown:

Accuracy: 72.63%

Correct: 69

Total: 95

True positives: 2

True negatives: 67

False positives: 0

False negatives: 26

False positive rate: 0.00%

False negative rate: 92.86%

3.3 Round 3: 588 Images, All Augmentations

For our third model, we included all augmentations. These included rotating the image, translating it, and blurring it. This negatively impacted the results, as the accuracy dropped from 72.63% in the previous model to 37.89% . We believe this is due to the blurring augmentation, as it may have completely erased the signature features in lung CT-scans which determine whether the patient is COVID-19 positive or not. Additional testing results are shown below:

Accuracy: 37.89%

Correct: 36

Total: 95

True positives: 2

True negatives: 34

False positives: 33

False negatives: 26

False positive rate: 49.25%

False negative rate: 92.86%

3.4   Round 4: Large Data-set, Rotations Only

For our fourth model, we used the large dataset with only the rotation augmentation. Although the accuracy is slightly lower than the model described in Section 2.2, the false negative rate has decreased dramatically. Additional testing results are shown below:

Accuracy: 67.37%

Correct: 64

Total: 95

True positives: 13

True negatives: 51

False positives: 16

False negatives: 15

False positive rate: 23.88%

False negative rate: 53.57%

3.5   Transfer Learning

For our transfer learning model, we used the Resnet18 model. The model was trained on the smaller data-set, due to time constraints. As this wasn't a main focus, we finished this step last. The results were extremely poor, as shown below:

Accuracy: 20.00%

correct: 19

total: 95

true positives: 13

true negatives: 6

false positives: 61

false negatives: 15

false positive rate: 91.04%

false negative rate: 53.57%

Chapter 4

Discussion/Conclusion

Overall, we were able to accomplish our mission objectives. We both learned about the different layers within a CNN and developed a CNN class module using Pytorch. We have come to the conclusion that the best model developed/trained was the model described in Section 3.4 (Large Data-set, Rotations Only). Despite having a slightly lower accuracy rate than the model described in Section 3.2 (588 Images, Rotations Only), the false negative rate was much lower in the Section 3.4 model.

Having low false negative rates in the virus testing is extremely crucial in mitigating the impacts of COVID-19. To review, a false negative means that although the model predicted the patient was COVID-19 negative, the patient actually does have the virus. Therefore, if a model has extremely high false negative rates, then it would allow for many individuals to unknowingly spread COVID-19, thinking that they aren't harming anyone.

Chapter 5

References

(n.d.). Retrieved from https://e2eml.school/how_convolutional_neural_networks_work.html

Katyal, R. (2019, November 10). Convolutional Neural Networks: Why, what and How!
Retrieved from https://blog.usejournal.com/convolutional-neural-networks-why-what-and-how-f8f6dbebb2f9

Saving and Loading Models. (n.d.). Retrieved from https://pytorch.org/tutorials/beginner/saving_loading_models.html

Training a Classifier. (n.d.). Retrieved from https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html

Transfer Learning for Computer Vision Tutorial. (n.d.). Retrieved from https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html

Torch.nn. (n.d.). Retrieved from https://pytorch.org/docs/master/nn.htmlnormalization-layers

Images from:

https://blog.usejournal.com/convolutional-neural-networks-why-what-and-how-f8f6dbebb2f9

https://databricks.com/glossary/neural-network

https://misi.tech/assets/img/brand-logo/misi-600.png

Latex Formulas created using:

https://databricks.com/glossary/neural-network