Covid-19 X-ray Classifier

Andre Williams   
*FAU – Computer Science*Boca Raton, FL, USA  
andrewilliam2018@fau.edu

*Abstract*— Computer Vision is a red-hot field in today’s world of machine learning. Image sentiment can be very useful as it gives machine learning methods a means of determining sentiment and emotions. Computers can now determine if a person is happy, if a person could potentially be a criminal, if a neighborhood is safe or not, if a woman is in love, how people feel about the next president, and a vast number of other conclusions [5]. There are situations where data scientist can input an x-ray photo into a machine learning algorithm and detect if a person has cancer better than Physicians. Since the invention of social media networks such as Twitter, Facebook, and Flickr. The machine learning field now has over hundreds of millions of images to use to train models on.

# Introduction

There are many different machine learning methods to use to solve image classification. The fundamental building block of computer vision is the convolutional neural network machine learning method. Also, there are more advanced methods where researchers will use a pretrained network such as VGGNET, ResNet, Inception, and Xception. ImageNet is a dataset of over 15 million labeled high-resolution images belonging to roughly 22,000 categories. A team of researchers from Qatar University, Doha, Qatar, and the University of Dhaka, Bangladesh along with their collaborators from Pakistan and Malaysia in collaboration with medical doctors have created a database of chest X-ray images for COVID-19 positive cases along with Normal and Viral Pneumonia images. This database is continually updated in stages.

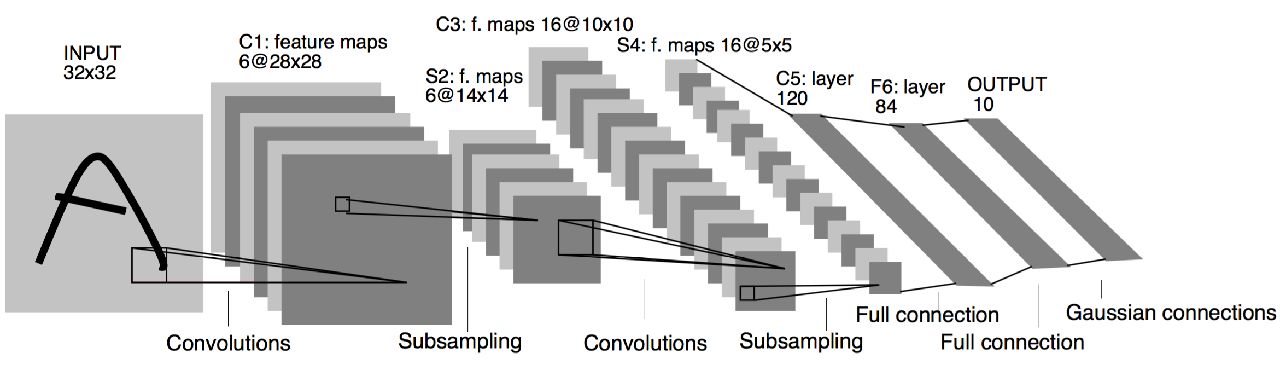
Some researchers have used VGG just to obtain the feature representation of the dataset and then use that representation as an input to a traditional Deep Learning Neural Network. This method is known as transfer learning and the goal is to experiment with this machine learning method.

A simple convolutional neural network will be employed as the baseline model and then experiment with batch normalization and different size nodes to improve the accuracy of the model. First, task will be to mine labeled images from kaggle. The next objective is to train and test the convolutional neural network model to classify the X-ray images. Finally, summarize the findings and possible future betterments researchers can experiment with to improve the model.

# Related Works

Image sentiment works boomed in the late 90’s. Earlier research extrapolated meta-data associated to images to assign sentiment scores and used correlations between images to classify the sentiment in terms of positive or negative. Before convolution neural networks, researchers used a single artificial neural network; however, they were not as accurate as single artificial neural networks only process global patterns, so they do not perform well with image classification as they are more fitted to detect global patterns and do not perform well if images are shifted, inverted, or distorted in some form or fashion.

In 2014 French computer scientist LeChun worked on the fine-tuned Convolution Neural Network which extract high-level features by finding local patterns in images.



First, an input image is fed to the network. Filters of a given size scan the image and perform convolutions. The obtained features then go through an activation function. Then, the output goes through a succession of pooling and other convolution operations. As you can see, features are reduced in dimension as the network goes on. At the end, high-level features are flattened and fed to fully connected layers, which will eventually yield class probabilities through a SoftMax layer. During training time, the networklearns how to recognize the features that make a sample belong to a given class throughbackpropagation [1]. This is the basic framework of convolutional network that this research will use with the addition of computational tricks such as dropouts.

# Baseline Work

This section will outline the framework of a simple convolutional neural network using the covid-19 dataset. Python will be the language of choice with a few commonly used packages: Pandas, NumPy’s, TensorFlow, Kera’s, and SKLearn.

## Dataset

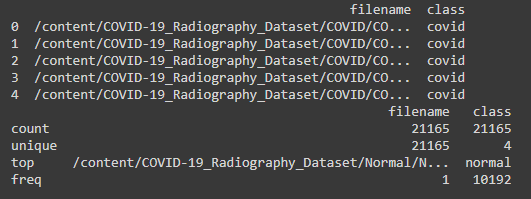
First thing is first, the dataset from Kaggle is a zip file with four folders for each class of image (Covid-19, Viral Pneumonia, Lung Opacity, and Normal). After unzipping the file there will be 3616 images in the Covid Folder, 6012 images in the Lung Opacify folder, 10192 in the Normal folder, and 1345 in our Viral Pneumonia folder. From first glance we can see that the classes are imbalanced which will reduce the accuracy of the CNN as the model will have a high bias. Due to the imbalance, we will employ cross validation to help our model see more images from the under-represented classes and see what other computational tricks we could use to improve accuracy. Unfortunately, there are not any packages to perform cross validation when images are flowing from a directory, so it is necessary to use flow from data frame by creating a data frame with the file path as one column and the class as another data frame. See code below:

Label\_names = [‘covid,’lung\_opacity’,’normal’,’viral’]

All\_filepaths = covid\_filenames + lung\_opacity\_filenames + normal\_filenames + viral\_filenames

All\_labels = [label\_names[0]]\*len(covid\_filenames) + [label\_names[1]]\*len(lung\_opacify\_filenames) + [label\_names[2]]\*len(normal\_filenames + [label\_names[3]]\*len(viral\_pneumonia\_filenames)

Df = pd.DataFrame({‘filename’:all\_filepaths, ‘class’:all\_labels})



After pre-processing data, it’s necessary to convert images to 3 dimensional NumPy’s arrays (one extra dimension for the channel as a 2-dimensional array is greyscale) then load valid images and sentiment labels into memory into two array list data structures based. The two list will be in the following (pseudo, the image list shapes are 32, 32, 3) format:

Image\_list = [0101101, 1001010, 00100101, 10010101, 0100101, …..]

After all the data is preprocessed into NumPy arrays, the next step is to do the standard machine learning practice of splitting the data into a training set (80 percent of raw data) and a testing set (20 percent of raw data). Finally, define the K-fold stratification for the train and validation set splits for cross validation. Since the data is now in dataframe format, it’s possible to get the indexes from the dataframe to perform the splits. See code below:

Train\_val\_df, test\_df = train\_test\_split(df)

Num\_cv\_splits = 5

Skf = StratifiedKFold(n\_splits=num\_cv\_splits, shuffle=True)

List\_train\_idxs = []

List\_val\_idxs = []

For train\_idxs, val\_idx, in skf.split(train\_val\_df[‘class’])

List\_train\_idxs.append(train\_idxs)

List\_val\_idxs.append(val\_idxs)

After splitting the data, the next step is to build a base model and then attempt to improve the results from there. For the base model, the TensorFlow package is used to build a convolutional neural network which will consist of two convolutional layers, two max pool layers, a flatten layers, two dropout layers, and three dense layers. The network will have the following architecture.

Before building the model architecture, it’s necessary to clearly explain the difference between batch size and epochs as we want to measure all of the experimented models with the same epochs and batch size. Essentially, an epoch is the amount of times train on the entire dataset. So, with an epoch of size 25 mean we would touch all samples in the dataset 25 times; need less to say, with the Covid-19 dataset of size 15000+ it could take a model a lengthy amount of time to get through 25 epochs. For research purposes 10 Epochs will suffice. Batch Size is the number of “buckets” to put the dataset into. Empirically speaking, if the batch size is 500 and the dataset is 15000 then it will take 30 iterations to get through 1 epoch (len(dataset)/batch size).

The labels have 4 categories (there are a very few Viral Pneumonia labels in the dataset that to take into consideration) therefore another option to make up for the imbalance is to use different loss functions. It is possible to consider compiling the model with the ‘sparse\_categorical\_crossentropy’ loss function which performs cross-entropy calculation of error without requiring that the target variables be one hot encoded prior to training; however, we will use categorical\_crossentropy. To adjust the weights, this research will use the Adam optimization algorithm for the baseline model and SGD for the following models. Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iterative based on training data. Adam combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. The dataset has 4 classes, so it is critical that our last Dense layer has 4 nodes, one for each class. Lastly, the best approach is to use Accuracy as the performance metrics. So, after using the architecture learned in class here is the first model:

model=Sequential()

model.add(Conv2D(32, (3,3),activation="relu",input\_shape=(32,32,3)))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Conv2D(32, (3,3),activation="relu"))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Conv2D(32, (3, 3),activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Flatten())

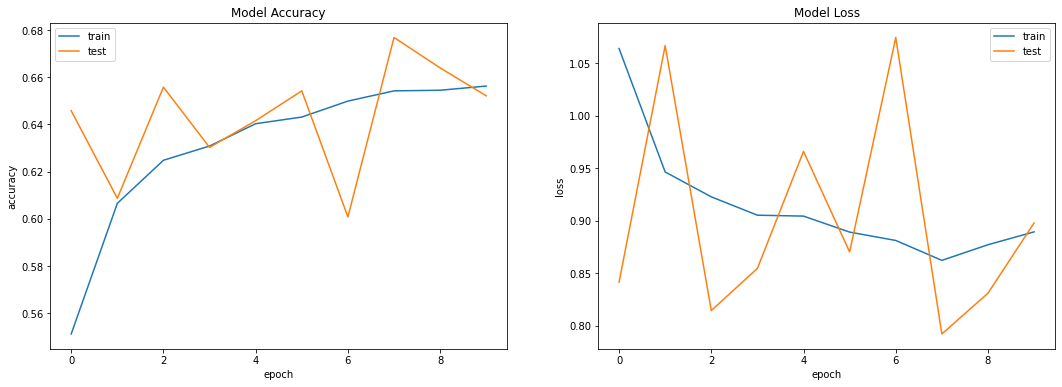
model.add(Dense(16, activation='relu'))

model.add(Dropout(0.1))

model.add(Dense(4, activation='sigmoids'))

model.compile(optimizer=’adam’, loss=’categorical\_crossentropy’,metrics=[‘accuracy’])

Here are the results of the baseline model.



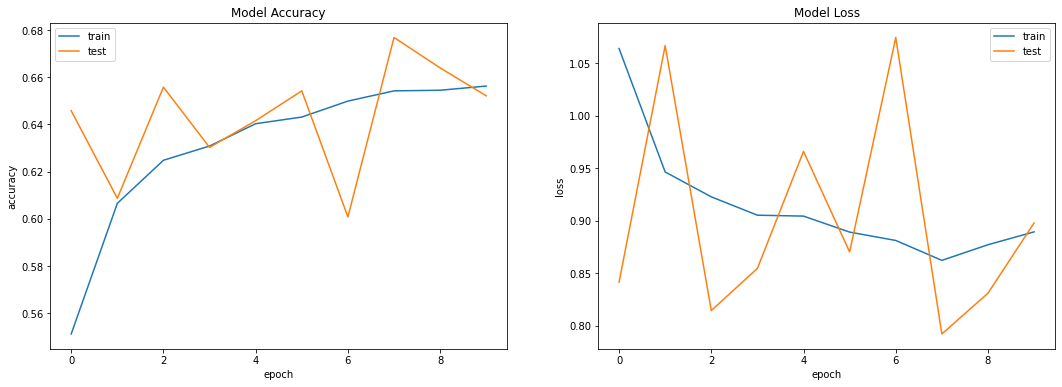


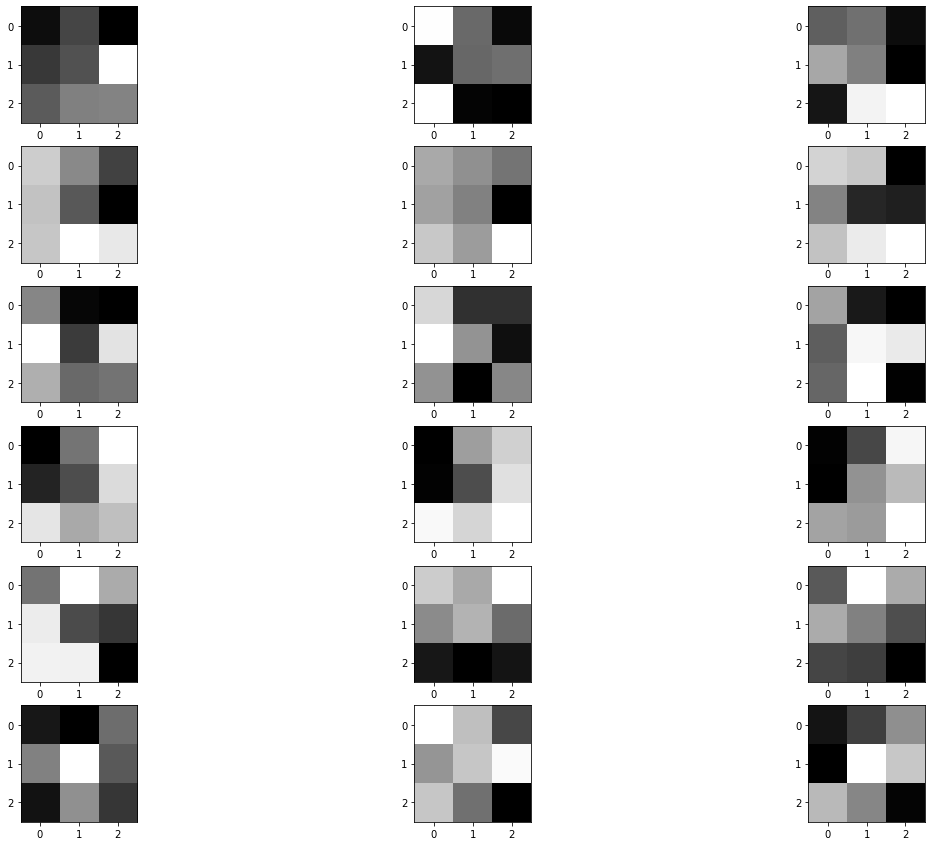
Figure 1: Accuracy of Class taught CNN

Though the accuracy and loss graphs look clean, the baseline model did not perform to well. After 10 epochs the model achieved an accuracy of 65%. Also, the accuracy struggles to stabilize, and we can see from the plot of the model loss that the model is failing to converge. “How can we improve these results”, one may ask? There are many techniques to us to reduce overfitting like regularization techniques and adding more layers to improve model accuracy. Considering this is a simpler machine learning method, this research will use ~65% as the baseline accuracy. Now review the summary (count of parameters) of the baseline model to later compare with the summary of the experimental models.



Figure 2: Parameters for Baseline Model

For this research, we will retrieve the weights from the second hidden layer and normalize the 3x3 filter values between 0 and 1 to visualize the filter. It is very hard to make sense of these filter with the human eye; however, we can feed a random picture from the dataset to see the output.



# Experiment:

Now the objective is to attempt experimental models that will improve the results of the CNN used for the baseline model. Below are two techniques to improve accuracy while continuing to use accuracy as the performance measure.

1. Changing the Loss Function
2. Add More layers with batch normalization.

There are no new additional packages needed to address the three experimental improvements as the baseline model itself (convolutional neural network) has not changed.

## Stochastic Gradient Descent

Stochastic gradient descent (SGD) in contrast performs a parameter update for each training example x and label y. Batch gradient descent performs redundant computations for large datasets, as it recomputes gradients for similar examples before each parameter update. SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster. While batch gradient descent converges to the minimum of the basin the parameters are placed in, SGD’s fluctuation, on the one hand, enable it to jump to new and potentially better local minima. [2]

## Batch Normalization

Batch normalization is a technique designed to automatically standardize the input to a layer in a deep learning neural network. Once implemented, batch normalization has the effect of dramatically accelerating the training process of neural network, and in some cases improves the performance of the model via a modest regularization effect [4]. Batch normalization allows each layer of network to learn by itself a little bit more independently of other layers. Batch normalization reduced overfitting because it has a slight regularization effect. Like dropouts, it adds some noise to each hidden layer’s activations. Therefore, using batch normalization requires less dropouts, which is a good thing because the model is not going to lose a lot of information. However, it’s not recommended to depend only on batch normalization for regularization; it’s better to use batch normalization together with dropout. To increase the stability of a neural network, batch normalization normalizes the output of a pervious activation layer by subtracting the batch mean and dividing by the batch standard deviation [4].

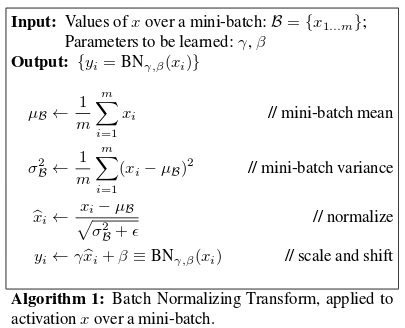


Figure 5: Batch Normalization Algorithm

## Modified CNN Experiment

Now after adding batch normalization into our toolset and changing the optimizer, the next step is to change the structure of the architecture to add more layers and more nodes after flattening into a dense layer. In this new architecture employ batch normalization node to provide some regularization. The architecture used in a mini version of VGGnet (a very commonly used CNN network) with less layers. Below is a glance at the new architecture.

# Defining the CNN architecture using keras Sequential API

model = Sequential()

model.add(Conv2D(32, kernel\_size=(3,3), padding="same", activation="relu", input\_shape=(128,128,3)))

model.add(BatchNormalization())

model.add(Conv2D(32, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2)))

model.add(Dropout(0.25))

model.add(Conv2D(64, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(BatchNormalization())

model.add(Conv2D(64, kernel\_size=(3,3), padding="same", activation="relu"))

model.add(BatchNormalization())

model.add(MaxPooling2D(pool\_size=(2,2), strides=(2,2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(512, activation="relu"))

model.add(BatchNormalization())

model.add(Dropout(0.5))

model.add(Dense(4, activation="softmax"))

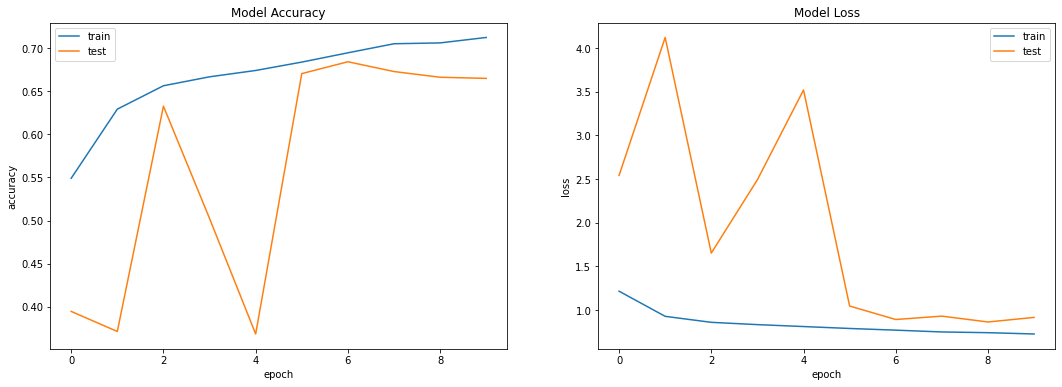
# printing model summary

model.summary()

# Compiling the model

model.compile(loss="categorical\_crossentropy", optimizer="sgd", metrics=["accuracy"])

Here are the results for this architecture:



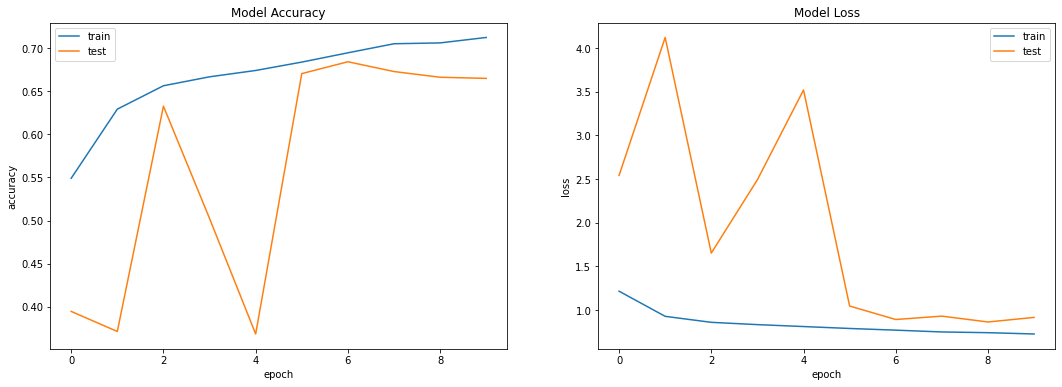


Figure 6: Modified CNN results

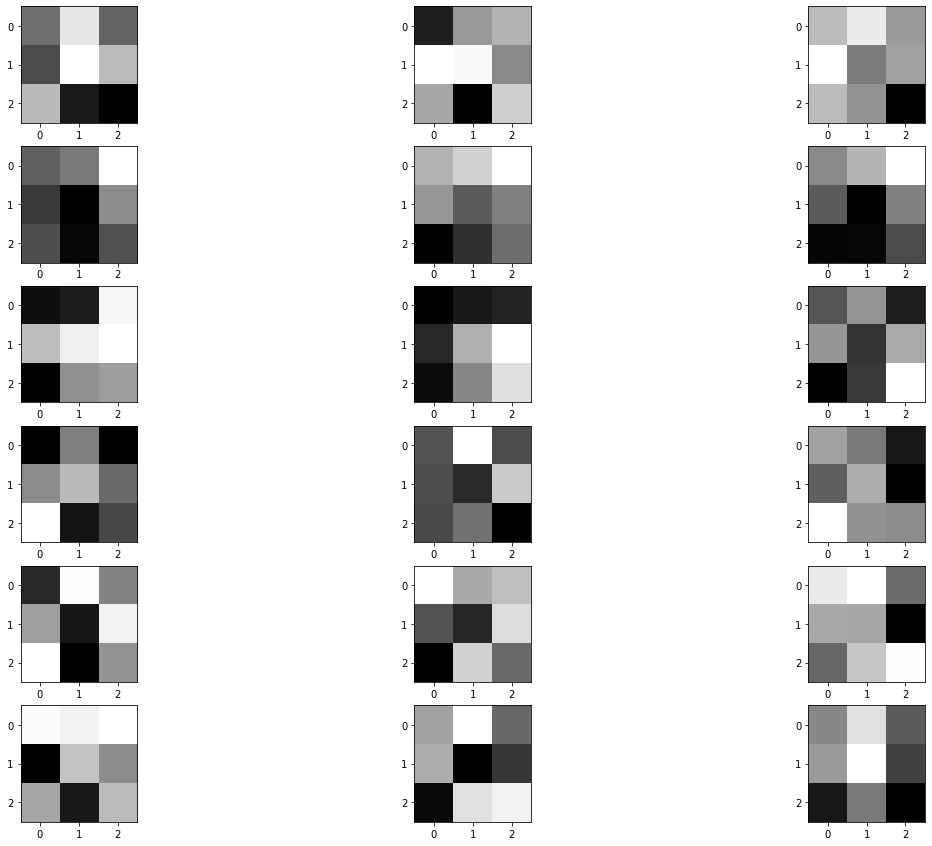
It clear to see this is a major improvement from the earlier architecture as this model has been able to achieve a high of ~71% accuracy on the test set which is a major improvement from the baseline model. The modified CNN is also more stable than the baseline model and from the loss graph the model appears to converge at the 5th epoch. In future works it is possible to change the optimizer to see if this could stabilize the accuracy and model loss even more to give a smoother result. Also, the runtime for the modified CNN will be the slowest method as the model will have many weights that need to be updated and optimized during training. The trainable parameters will be the highlight as a performance metric for the following models as these parameters are congruent to computation complexity.



Figure 7: Parameters for Modified CNN

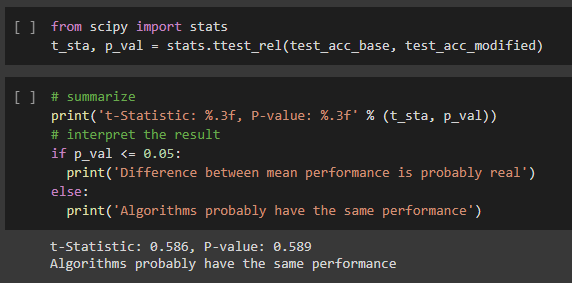
It is safe to say that the Modified CNN is a testament to the reason why researchers should experiment with different optimizers and batch normalization to help with convergence. Next experiment will be the transfer learning method, maybe a transfer learning model can achieve higher accuracy with this commonly used method.

When retrieving the second layer for the modified CNN, we can compare the normalized filters of the modified CNN to the normalized filters of the baseline CNN and it a clear difference between the two. It’s also safe to say that the filters shown below are more accurate than the filters of the baseline model. In future works it would be compelling to select a random sample from the database and feed it to the two separate filters to see the results.



## Comparison of Models

Its clearly obvious that the modified CNN slightly outperforms the based model CNN; however, this observation could be by chance. One could assume if the model were to be re-fit hundreds of times, then it’s possible that the base model could occasionally outperform the modified CNN. Standard practice to compare two models is to do statistical test to see if there is a significant difference between the two or if they both perform similar. See below for the statistical test of the base CNN model vs modified CNN model.



## Transfer Learning Overview

The basic premise: of transfer leaning is simple: take a model trained on a large dataset and transfer its knowledge to a smaller dataset. For object recognition with a CNN, best practice is to freeze the early convolutional layers of the network and only train the last few layers which make a prediction. The idea is the convolutional layers extract general, low-level features that are applicable across images – such as edges, patterns, gradients – and the later layers identify specific features within an image such as eyes or wheels (Koehrsen).

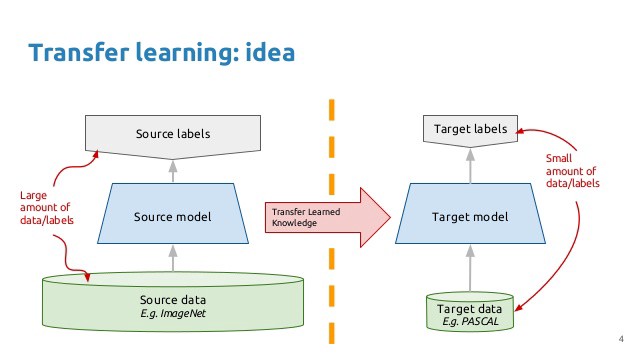


Figure 8: Flowchart of Transfer Learning [7]

Following is the general outline for transfer learning for object recognition:

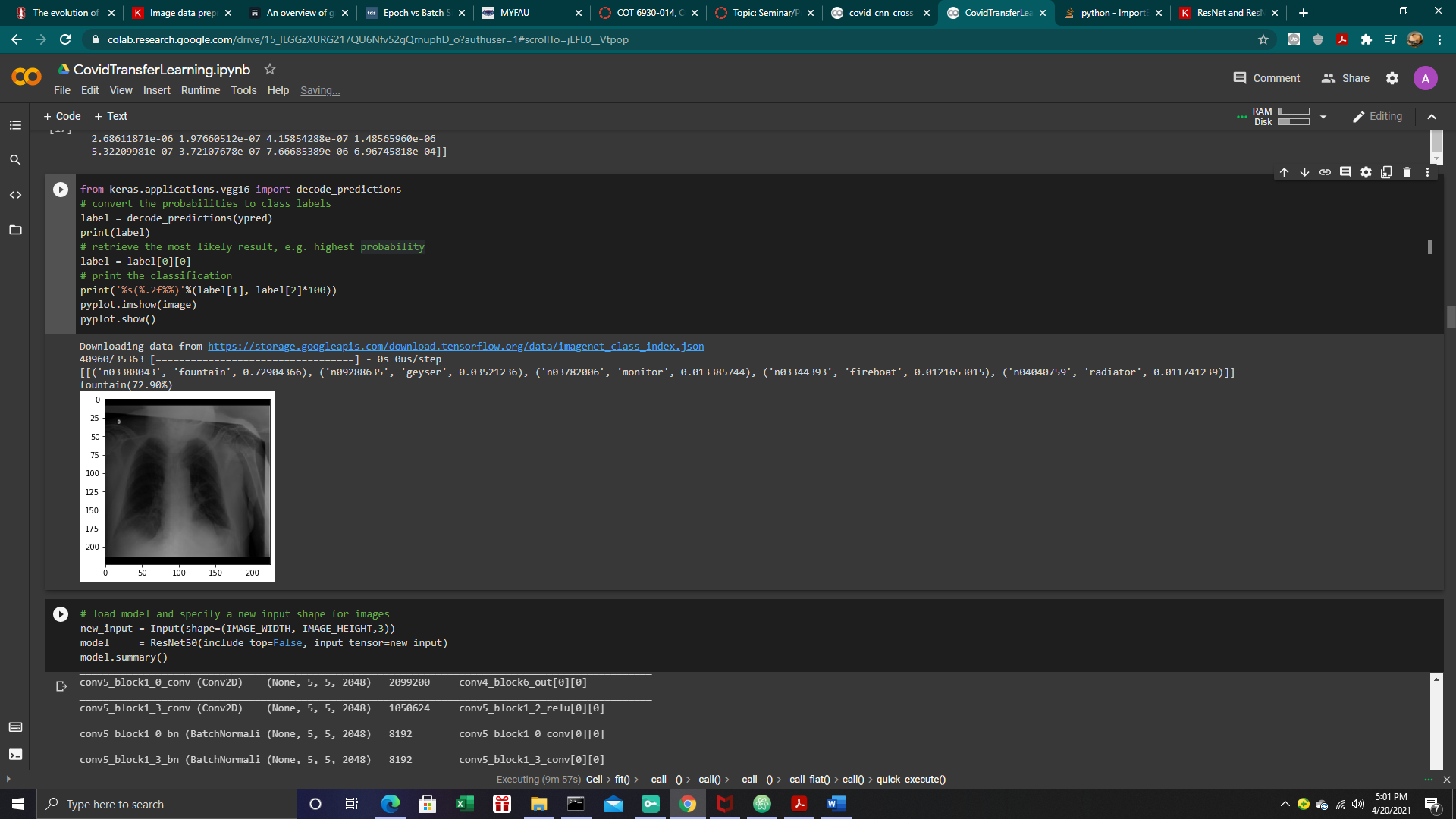
1. Load in a pre-trained CNN model trained on a large dataset.
2. Freeze parameters (weights) in model’s lower convolutional layers
3. Add custom classifier with several layers of trainable parameters to model.
4. Train classifier layers on training data available for task
5. Fine-tune hyperparameters and unfreeze more layers as needed.

(Koehrsen)

One important discussion to be had about transfer learning. The final layers of the pretrained model do not include a flatten layer or a dense layer so just using the pretrained model alone can give you a wide range of results not related to the classes assigned to the dataset fed to the model. Here is an example of a bare bones ResNet50 output. Say we feed a ResNet50 model the following random photo from the Covid-19 dataset.

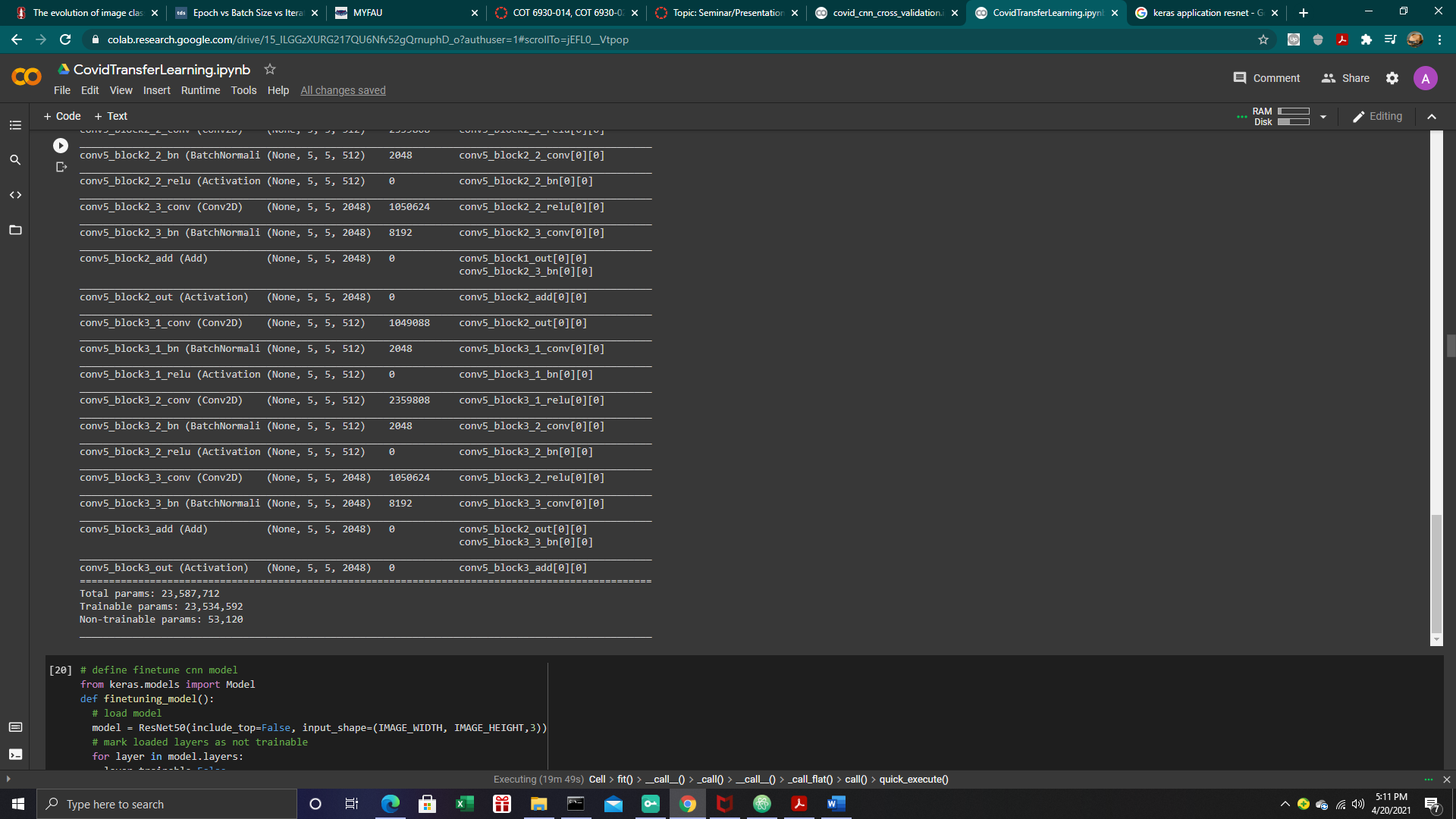


Hopefully the model would output Covid, Normal, Lung Opacity, or Viral Pneumonia; however, given that the model has free range to use any of the millions of classes used in the pretrained model the prediction is a fountain with 72.90% accuracy. Hence is why it’s necessary to add layers on top of the pretrained model to fine tune the pretrained model.



## Transfer Learning Experiment

First experiment will be a ResNet50 model initialized with weights trained on the ImageNet dataset with size of 128,128,3. Then freeze all layers so that they will not be updated during the training process. Next, flatten the output of ResNet and add a Dense layer with regularization. The objective here is to experiment with fine-tuning a pretrained ResNet-50 and VGG. Because this network is very deep (has 50 layers) and is trained on both scene-centric data (MS Coco) as well as object-centric data (ImageNet), the expectation this model will better results [5]. Keep in mind, if resnet.trainable = True is used, it will increase the number of trainable parameters; however, by freezing the network and setting it to false the model will lower trainable parameters. Only 10 epochs should be enough to yield good performance. The number of trainable parameters (even after freezing the layers) is around 6 million + with the resnet50 model. So modifying the architecture by reducing the number of dense layer neurons or using a global pooling layer on top of the pretrained ResNet50 base model may help reduce the number of parameters resulting in reducing the training time. As of now with the out of the box ResNet50 model it will take about 3-4 hours to train the model. Here are the last layers of the ResNet50 pretrained model and its parameter count:



Here are the last layers of the ResNet50 pretrained model and it’s parameter count after fine tuning the pretrained model with three additional layers. To get our final amount of trainable parameters we would combine the ResNet base plus the additional layers to give us a total of 30 million + weights to be adjusted during training.

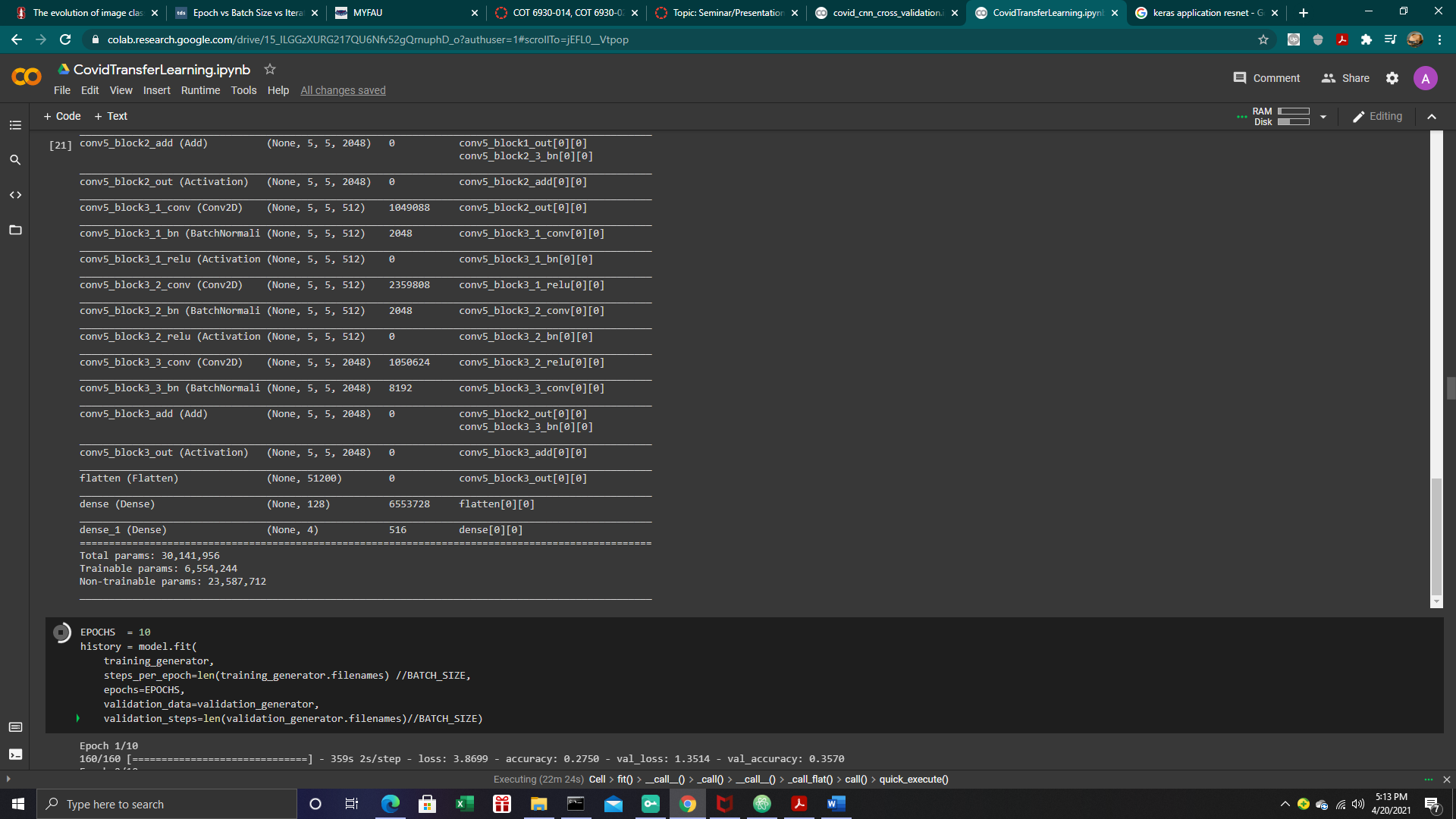


Figure 9: Parameters for Pretrained Dataset

Transfer Learning using ResNet50 provided us with poor results as the accuracy is as high as ~45% accuracy which is worse than a Modified CNN and the training time is impractical. See results below.

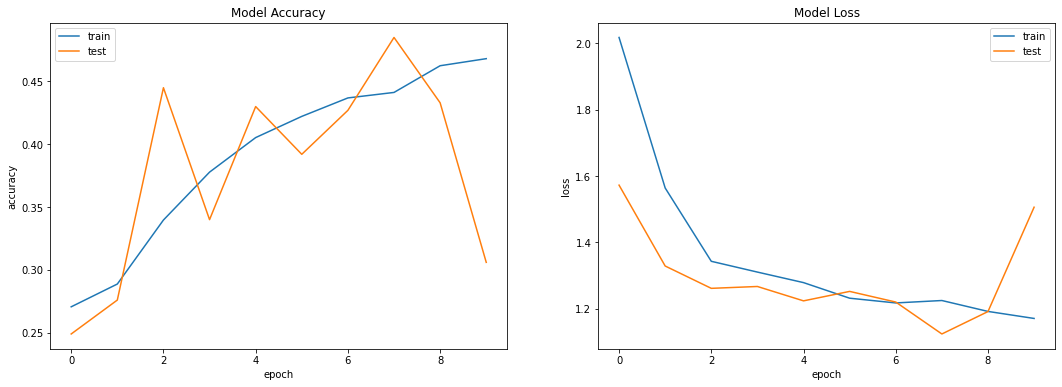


Figure 10: Accuracy of ResNet Transfer Learning Model

For experimental purposes, try another pretrained model to achieve better results. Let us see if it is possible to get better results with a VGG Pretrained network with weights trained on ImageNet. Below is the architecture for VGG:

From keras.application.vgg16 import VGG16

New\_input= Input(shape=(Image\_Width, Image\_height,3)

Model = VGG16(include\_top=False, input\_tensor=new\_input)

Model.summary()

The good news about VGG network is that it is considerably faster because there are less trainable parameters. Below are the parameters of the VGG16 model, which we will fine tune add more layers on top of the default VGG models. See below for the total trainable parameters after fine tuning the model.

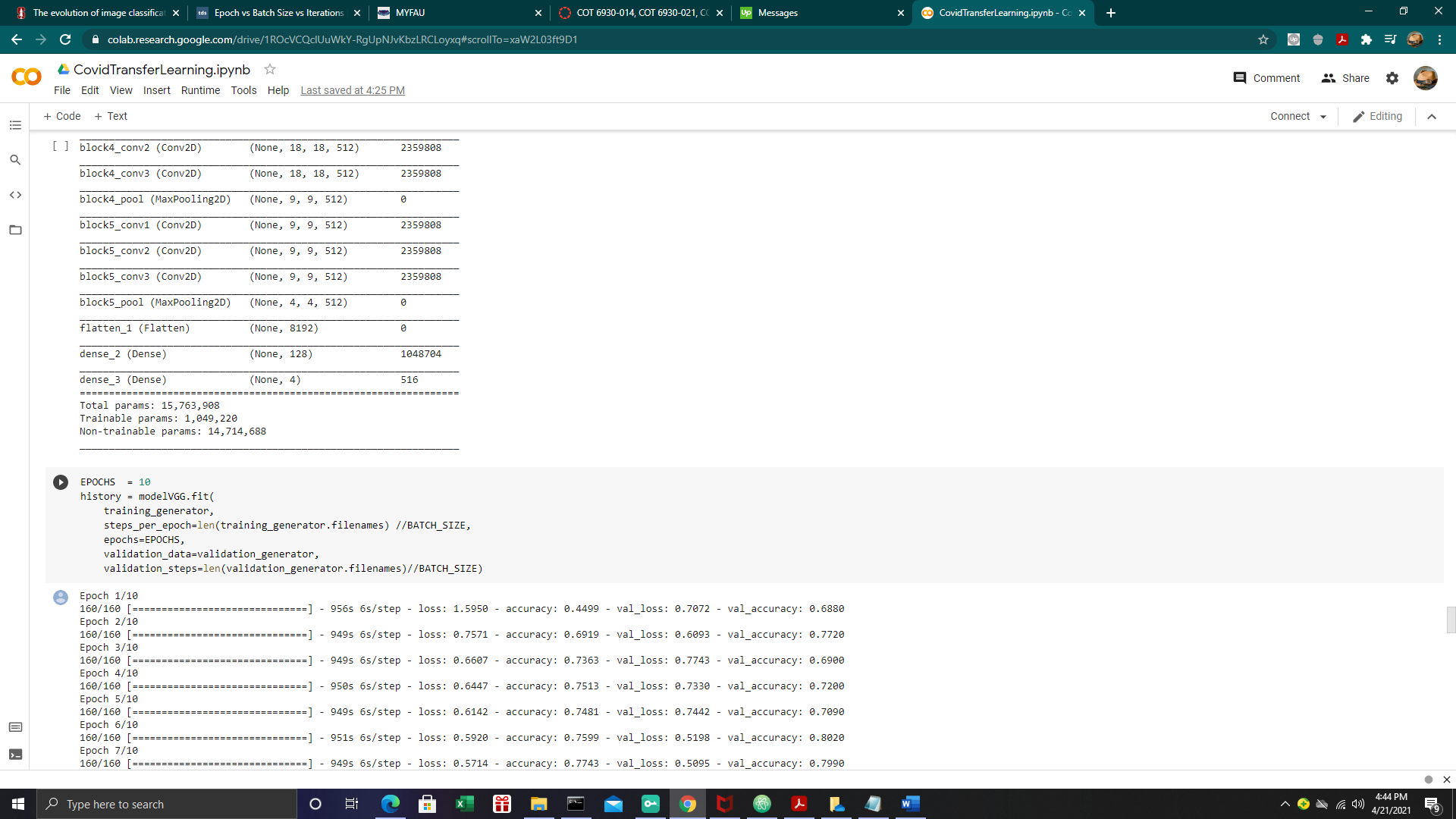


Figure 11: Parameters for VGG Network

Now we will add our own custom layers on top of the VGG model to implement our transfer learning model.

Model = VGG16(include\_top=False, input\_shape = input\_shape)

For layer in model.layers:

Layer.trainable=False

Flat1 = Flatten()(model.layers[-1].output)

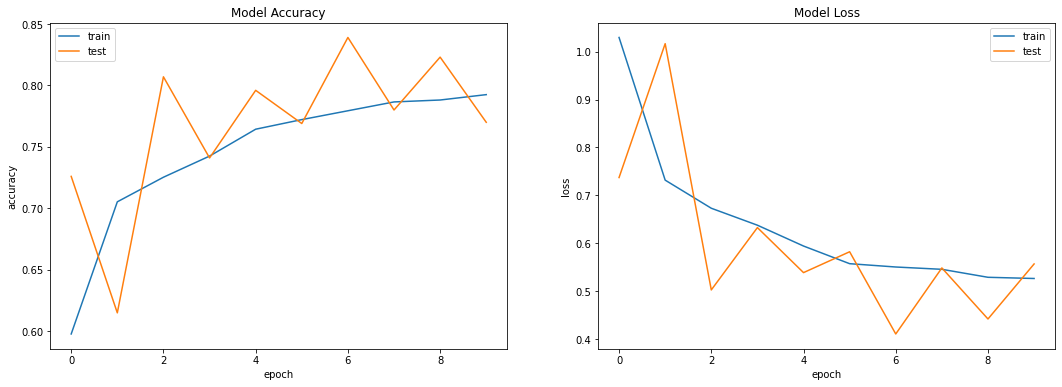
Class1=Dense(128,activation=’relu’)(flat1)

Output=Dense(4,activation=’softmax’)(class1)

Model = Model(inputs=model.inputs,outputs=outputs)

Model.compile(loss=’categorical\_crossentropy)

Transfer Learning using VGG provided us with the best results as the accuracy is as high as ~80% accuracy which is twofold better than ResNet. See results below.



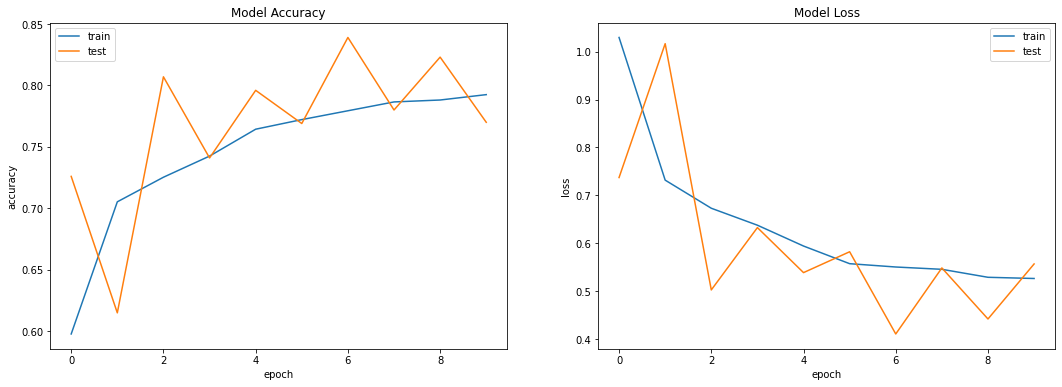


Figure 12: Accuracy of VGG Transfer Learning Method

# Analysis of Results

|  |  |  |
| --- | --- | --- |
|  | Accuracy | Trainable Parameters |
| Baseline CNN | 65% | 673,780 |
| Modified CNN | 71 % | 44,930,340 |
| ResNet | 45% | 30,141,946 |
| VGG | 80% | 15,763,908 |

# Conclusion

In this research, Covid-19 X-ray image classification was overviewed with state-of-the-art techniques in the field [1]. Principles of design of classification systems are presented and discussed under four main points of view: A simple CNN as a baseline learner, a modified CNN, ResNet and VGG Pretrained model. This research also reviewed the components that can affect the classification of an image in different ways are defined and analyzed. One of the main challenges is the dataset itself and how to preprocess the data to implement cross validation. Another challenge was how to compare models against each other to see if they perform the same or if one model is statistically different than another model. Using Transfer learning seem to outperform a baseline CNN which could be due to transfer learning model being pre-trained on millions of instances. Of the two-transfer learning pretrained networks that were used VGG was considerably faster and more accurate than ResNet50 which also helps prove that models with fewer trainable parameters (weights) will operate faster than models with more trainable parameters. Due to the extensive training time of the transfer learning model, it’s plausible to compare the model accuracy of the two and assume that there is a statistical significance between the two models. For future works, it would be good to compare the p values of the two using the traditional cross validation of the two models. Note: for Sklearn models we could use the paired\_ttest\_5x2cv package to implement a simplified version of the statistical comparison; however, unfortunately transfer learning is a deep learning model from TensorFlow so we would have to do a traditional cross validation (a model with 6 million parameters, 3-4 hours per fit, for 5 different sets of validations could be rather timely).

# Future Works

It is possible to generate a numerous number of ideas to incorporate in future works. One option would be to explore adding additional layers on top of the pre-trained networks (What would happen after adding 6 or more layers on top). Researching GAN’s to generate new samples to help overcome the imbalanced dataset is plausible, one of the main challenges for this method is developing a way to label the computer-generated images. It is possible to also work on why the accuracy does not improve as the weights update.

# Acknowledgment

##### The author gives thanks to Dr. Xingquan Zhu for in motivation to complete this project. Florida Atlantic University for providing quality machine learning education. Finally, the author would like to thank the reviewers for their helpful suggestions.

##### References

1. Afshine Amidi, Shervine Amidi. *The evolution of image classification explained.*November2018. <https://stanford.edu/~shervine/blog/evolution-image-classification-explained>
2. Sebastian Ruder. *An overview of gradient descent optimization algorithms.* 19 January 2016 [www.ruder.io](http://www.ruder.io) 18 April 2021
3. ~~Alessandro Ortis, Giovanni Maria Farinella, and Sebastiano battiato.~~ *~~An Overview on Image Sentiment Analysis: Methods, Datasets, and current Challenges~~*~~. Report. Cantania Italy: University of Catania, n.d.~~
4. Boyle, Tara. *Dealing with Imbalanced Data*. 3 February 2019. towardsdatascience.com. 12 December 2020.
5. Brownlee, Jason. *How to accelerate learning of Deep Neural Network with Batch Normalization*. 25 August 2020. www.machinelearning mastery.com. 12 December 2020.
6. D, F. *Batch normalization in Neural Networks*. 20 October 2017. www.towardsdatascience.com. 12 December 2020.
7. Gupta, Vasavi Gararla and Aditi. *Emotion Detection and Sentiment Analysis of Images*. Report. Georgia: Georgia Institute of Technology, n.d.
8. Honbin Zhang, Jinpeng Wu, Haowei Shi, Ziliang Jiang, Donghong Ji, Tian Yuan, and Guangli Li. *Multidimensional Extra Evidence Mining for Image Sentiment Analysis*. Wuhan China: ieee Access, 2020. ieeexplore.ieee.org. 2020 December 2020.
9. Koehrsen, Will. *Transfer Learning with onvolutional Neural Networks in PyTorch*. 26 November 2018. www.towardsdatascience.com. 12 December 2020.
10. Rocca, Baptiste. *Handling imbalanced datasets in machine learning*. 27 January 2019. www.towardsdatascience.com. 14 December 2020.
11. Leon, Juan. "Does the iput image size affect CNN's performance." *blog*. Colombia: www.Quora.com, 10 May 2019. Blog.
12. Popular networks. *VGG16 - Convolutional Network for Classification and Detection*. 2020 November 2018. https://neurohive.io/en/popular-networks/vgg16/. 05 December 2020.