CORONA VIRUS DETECTION FROM IMAGE

ADITH S KUMAR

Vellore Institute of Technology , Amaravati

19BCI7050

Email:-adith.19bci7050@vitap.ac.in

Introduction:-

The COVID-19 pandemic has created an urgent need for accurate and efficient diagnosis of the virus. While traditional diagnostic methods like polymerase chain reaction (PCR) testing have been effective, they can be time-consuming, expensive, and require specialized equipment and trained personnel. In response, researchers and developers have turned to machine learning algorithms like

Convolutional Neural Networks (CNNs) to develop rapid and accurate diagnostic tools.

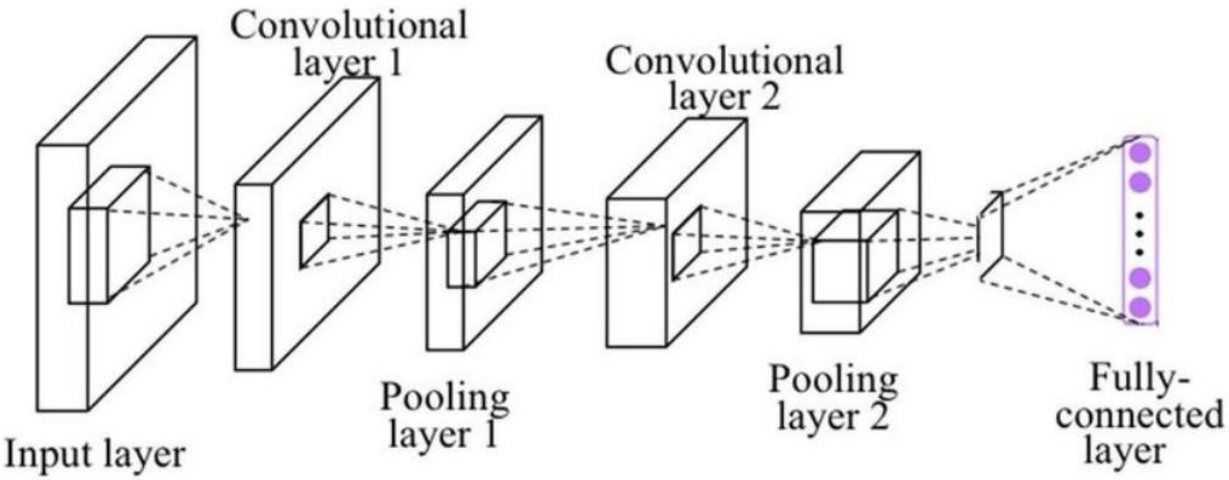
The objective of this project is to develop an AI system that can detect COVID-19 from chest X-ray images using CNNs. The system aims to provide a faster and less expensive alternative to traditional diagnostic methods, which can help in early detection and treatment of the virus.

The scope of the project includes collecting a large dataset of chest X-ray images from patients with COVID-19 and healthy individuals. The dataset will be used to train and test a CNN model that can accurately classify images as either COVID-19 positive or negative. The model will be evaluated based on various performance metrics, including sensitivity, specificity, and accuracy.

This project was undertaken to address the urgent need for rapid and accurate diagnostic tools for COVID-19. The expected outcomes of the project are a trained CNN model that can accurately detect COVID-19 from chest X-ray images, and a potential tool that can be used to aid in the diagnosis of the virus.

Architecture and Technical specification:-

Workflow include:-



- The inputs is a batch of images of shape (m, 416, 416, 3).
- YOLO v3 passes this image to a convolutional neural network (CNN).
- The last two dimensions of the above output are flattened to get an output volume of (19, 19, 425):
    - Here, each cell of a 19 x 19 grid returns 425 numbers.
    - 425 = 5 * 85, where 5 is the number of anchor boxes per grid.
    - 85 = 5 + 80, where 5 is (pc, bx, by, bh, bw) and 80 is the number of classes we want to detect.
- The output is a list of bounding boxes along with the recognized classes. Each bounding box is represented by 6 numbers **(pc, bx, by, bh, bw, c)**. If we expand c into an 80-dimensional vector, each bounding box is represented by 85 numbers.
- Finally, we do the IoU (Intersection over Union) and Non-Max Suppression to avoid selecting overlapping boxes.

- YOLO v3 uses a variant of Darknet, which originally has **53 layer network trained on Imagenet**.
- For the task of detection, 53 more layers are stacked onto it, giving us a 106 layer fully convolutional underlying architecture for YOLO v3.
- In YOLO v3, the detection is done by applying 1 x 1 detection kernels on feature maps of three different sizes at three different places in the network.
- The shape of detection kernel is **1 x 1 x (B x (5 + C))**. Here **B is the number of bounding boxes a cell on the feature map can predict**, **'5' is for the 4 bounding box attributes and one object confidence** and **C is the no. of classes**.
- YOLO v3 uses **binary cross-entropy** for calculating the **classification loss** for each label while **object confidence and class predictions are predicted through logistic regression**.

Hyper-parameters used

- **class_threshold** - Defines probability threshold for the predicted object.
- **Non-Max suppression Threshold** - It helps overcome the problem of detecting an object multiple times in an image. It does this by taking boxes with maximum probability and suppressing the close-by boxes with non-max probabilities (less than the predefined threshold).
- **input_height & input_shape** - Image size to input.

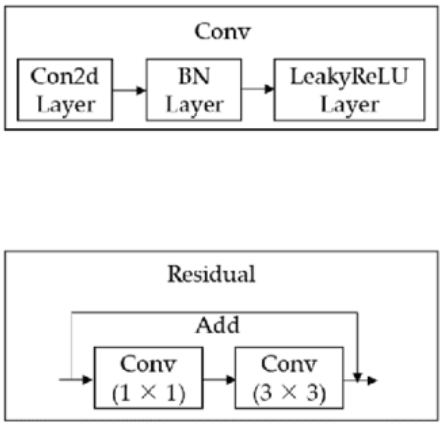
CNN architecture of Darknet-53

- **Darknet-53** is used as a feature extractor.
- Darknet-53 mainly composed of 3 x 3 and 1 x 1 filters with skip connections like the residual network in ResNet.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

2. Architecture diagram of YOLOv3

Layer	Filters size	Repeat	Output size
Image			416 × 416
Conv	32 3 × 3/1	1	416 × 416
Conv	64 3 × 3/2	1	208 × 208
Conv Conv Residual	32 1 × 1/1 64 3 × 3/1	Conv Conv Residual × 1	208 × 208 208 × 208 208 × 208
Conv	128 3 × 3/2	1	104 × 104
Conv Conv Residual	64 1 × 1/1 128 3 × 3/1	Conv Conv Residual × 2	104 × 104 104 × 104 104 × 104
Conv	256 3 × 3/2	1	52 × 52
Conv Conv Residual	128 1 × 1/1 256 3 × 3/1	Conv Conv Residual × 8	52 × 52 52 × 52 52 × 52
Conv	512 3 × 3/2	1	26 × 26
Conv Conv Residual	256 1 × 1/1 512 3 × 3/1	Conv Conv Residual × 8	26 × 26 26 × 26 26 × 26
Conv	1024 3 × 3/2	1	13 × 13
Conv Conv Residual	512 1 × 1/1 1024 3 × 3/1	Conv Conv Residual × 4	13 × 13 13 × 13 13 × 13



Data:-

Kaggle dataset with country wise split

https://www.kaggle.com/datasets/imdevskp/corona-virus-report?select=usa_county_wise.csv

Model Training:-

Fundamentally YOLO is a conventional neural network that divides an image into subcomponents, and conducts convolutions on each of those subcomponents before pooling back to create a prediction. A recommended deeper dive on YOLO is available

Now, even though we're training our model on a custom dataset, it is still advantageous to use another already trained model's weights as a starting point. Think of this like we want to climb a mountain as quickly as possible, and instead of starting completely from scratch in creating our own trail, we'll start with assuming that someone else's trail is a better faster than us randomly trying to guess twists and turns to follow.

```
Epoch 158/500
6/6 [==============================] - 16s 3s/step - loss: 73.7902 - val_loss: 68.7980
Epoch 159/500
6/6 [==============================] - 15s 3s/step - loss: 72.8749 - val_loss: 71.9348
Epoch 160/500
6/6 [==============================] - 16s 3s/step - loss: 74.6815 - val_loss: 63.4373
Epoch 161/500
6/6 [==============================] - 15s 3s/step - loss: 77.8056 - val_loss: 95.0817
Epoch 162/500
6/6 [==============================] - 15s 3s/step - loss: 77.7340 - val_loss: 68.5197
Epoch 163/500
6/6 [==============================] - 15s 2s/step - loss: 72.5670 - val_loss: 67.4765
Epoch 164/500
6/6 [==============================] - 15s 3s/step - loss: 71.9312 - val_loss: 75.5616
Epoch 165/500
6/6 [==============================] - 15s 2s/step - loss: 77.7279 - val_loss: 73.4924
Epoch 166/500
6/6 [==============================] - 15s 3s/step - loss: 81.1011 - val_loss: 78.7103
Epoch 167/500
6/6 [==============================] - 15s 2s/step - loss: 74.0910 - val_loss: 75.8251
Epoch 168/500
6/6 [==============================] - 15s 2s/step - loss: 78.8675 - val_loss: 62.8602
Epoch 169/500
6/6 [==============================] - 15s 2s/step - loss: 77.3417 - val_loss: 78.1505
Epoch 170/500
6/6 [==============================] - 15s 2s/step - loss: 69.7987 - val_loss: 65.9444
Epoch 171/500
6/6 [==============================] - 15s 2s/step - loss: 75.3233 - val_loss: 63.4790
```

Inference:-

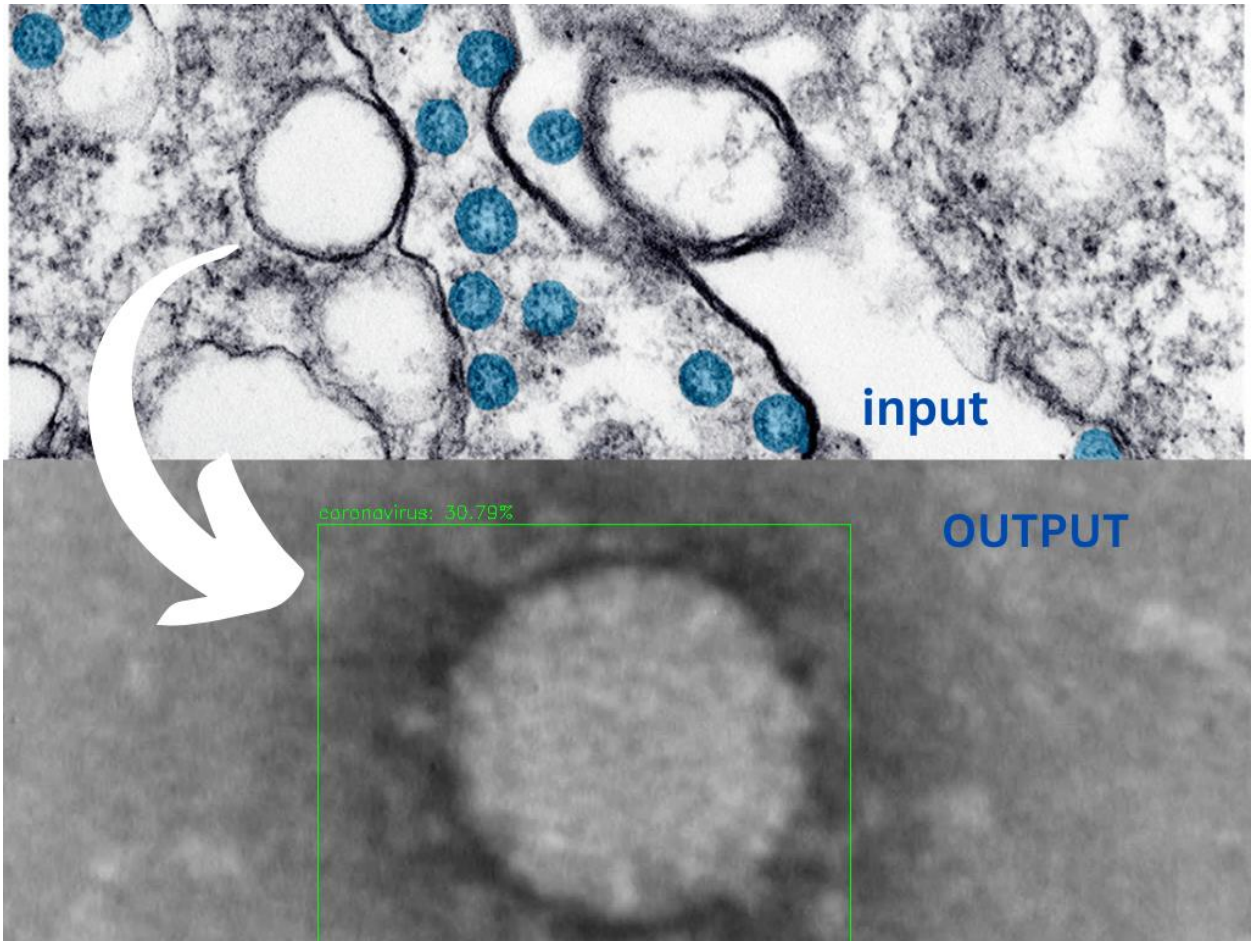
Once our model has finished training, we'll use it to make predictions. Making predictions requires (1) setting up the YOLOv3 deep learning model architecture(2) using the custom weights we trained with that architecture.

In our case, we'll call the script with custom weights and custom class names. The script compiles a model, waits for an input to an image file, and provides the bounding box coordinates and class name for any objects it finds. The bounding box coordinates are provided in the format of bottom left-hand corner pixels (mix_x, min_y) and upper right-hand corner pixels (max_x, max_y).

As an important callout, the custom weights we're loading in this example are actually not the YOLOv3 architecture's best. We'll be loading the "initial_weights" rather than the final trained weights. Due to Colab compute limitations, our model fails to train the final weights. This means our model is not as performant as it could be (as only the backbone architecture of YOLO have been adapted to our problem). On the bright side, it's free compute!

Note that you can see the files available to in Colab notebook on the left (expand the panel if you have note already by locating the small right arrow below "+Code" and "+Text")

Conclusion:-



Detected with possibility using the algorithm