## **CI7520 – Machine Learning and Artificial Intelligence**

# Topic: Neural Network / Artificial Intelligence System

Name: NILENDU SAHA

# **Stream: Mechatronic Systems**

KU Number: K1943079

Date of Submission: 29/03/2020

1. **INTRODUCTION**

Deep Neural Networks have created a revolution in each and every field. Either its in a manufacturing industry or in a normal football game, the concepts can fit in any region. The motive of this assignment is to make a Machine Learning Application with Deep Neural Networks. The aims and objectives was to be fulfilled from a given set of themes.

For this particular assignment the Image Data theme was selected. The main objective of the entire task is to detect the object of a given image from a set of class labels. The VisDrone-2019 image dataset was used. The assignment contains several objectives, they are as follows,

* The initial task is to use an existing DNN and implement the Object Detection using an image from the drone dataset.
* Then train the dataset on existing Object Detection DNN models and analyse the results.
* Create a model for Image Classification and plotting the results.

1. **ANALYSIS OF THE EXISTING DNNs**

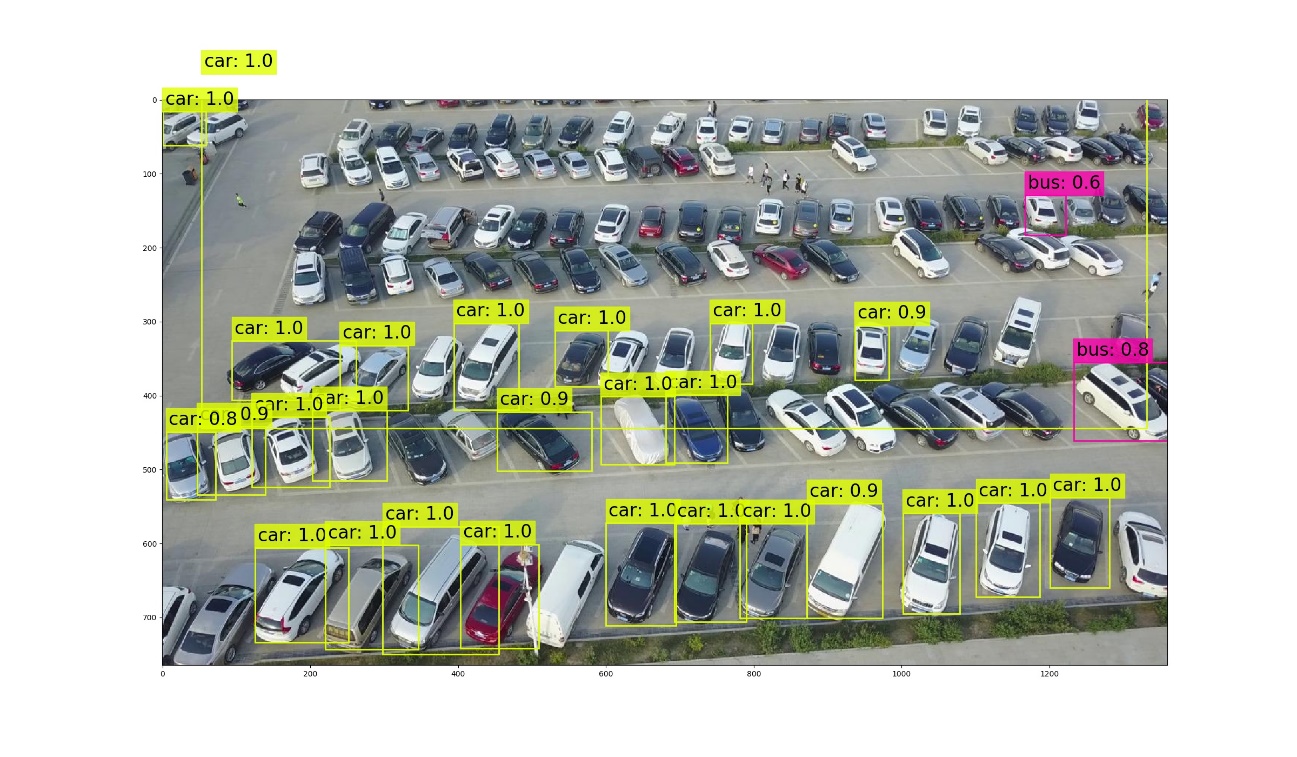
Three types of DNNs were utilised to meet the objectives of this assignment. They are as follows:

* YOLOv3
* MASK-RCNN
* ResNet

All the mentioned models have been trained on the well-known COCO dataset. The COCO dataset has 80 classes.

**YOLOv3:**

This model was used in both Part 2 and Part 3 of the assignment. It has a total of 106 layers which consists of 75 CNN layers and 31 other layers to generate a perfect output. The model predicts the class of the objects, bounding boxes and the average precision score.



**Fig. 1**

The above figure 1 represents the yolov3 output image of a randomly selected image. The model is able to detect almost majority number of cars in the picture. Some car labels have an AP score of 0.9 and 1.0. Some objects are misclassified as bus.

**Mask-RCNN:**

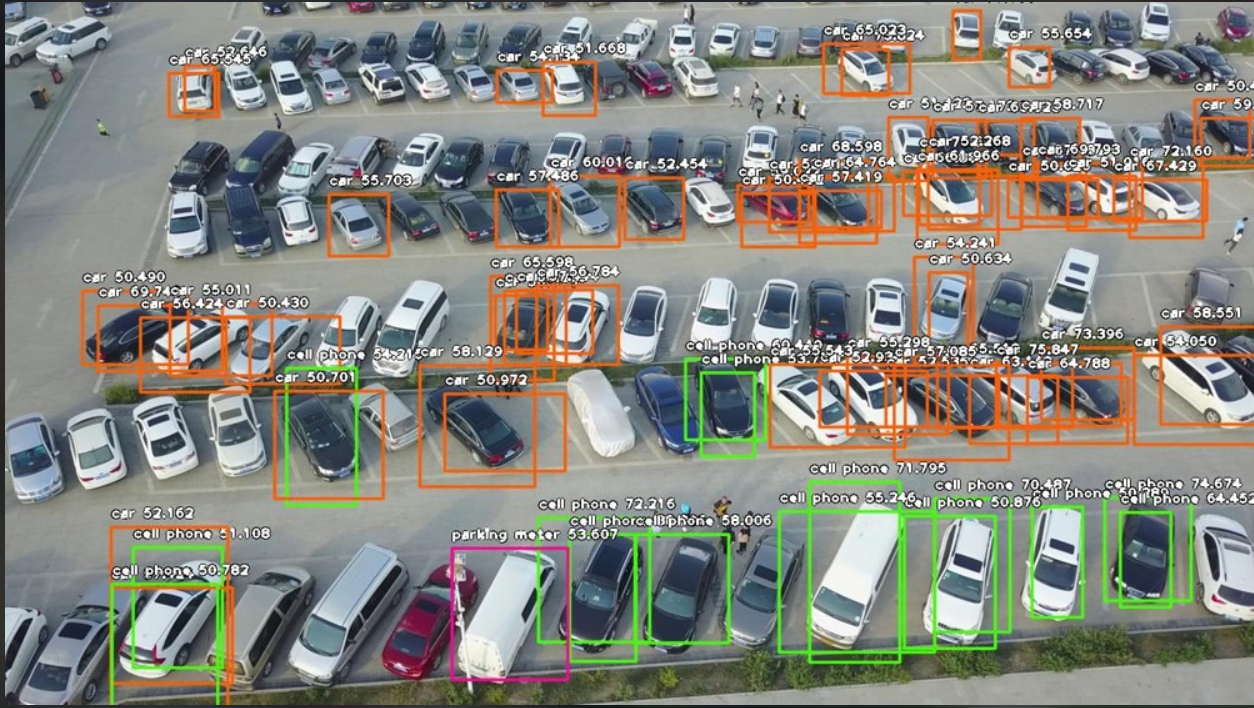
It is a DNN which is used to solve image segmentation problems at instances. Simply it can separate different objects in an image. It not only predicts the class and the bounding boxes of the object but also masks the object along its edges with the help of a particular faded colour of a certain class. There are two stages while prediction. First the model tries to approximate the chances of a known object in the random uploaded image. Then it predicts the class, the bounding box, average precision of the prediction and finally masks the object from the image. Both of these stages are connected to a backbone structure.

**Fig. 2**

The above given figure 2 forms the output image of the mask-rcnn object detection method. Here the model is able to make more bounding boxes than the yolov3 model. It is also able to provide AP scores and also mask all the edges of the object. But most of the objects are misclassified as either cell phone or chair.

**ResNet:**

ResNet 50 forms a backbone for almost every object detection challenges till date. The model consists of five stages. Each stage has a convolution and an Identity block. Each of the convolution block has three convolution layers and each identity block also has three convolution layers. This model simply predicts the class of the objects, its bounding boxes and the average precision score. Its output is as same as the output of the Yolov3.

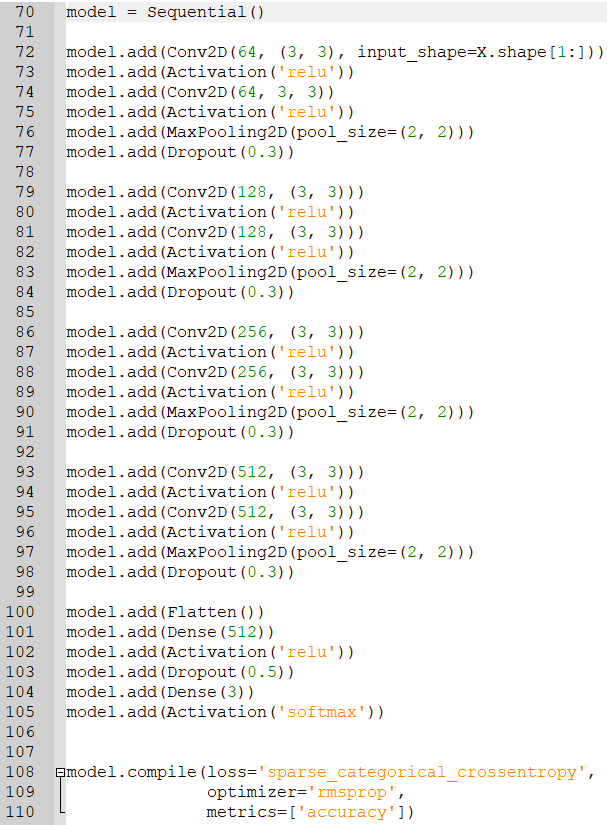


**Fig. 3**

The above figure 3 shows the output image for ResNet50 model. The model is also able to detect objects much more than the yolov3 image. It is also showing the bounding boxes and the AP scores. The scores are pretty lower than the other two models. There are many misclassified objects too.

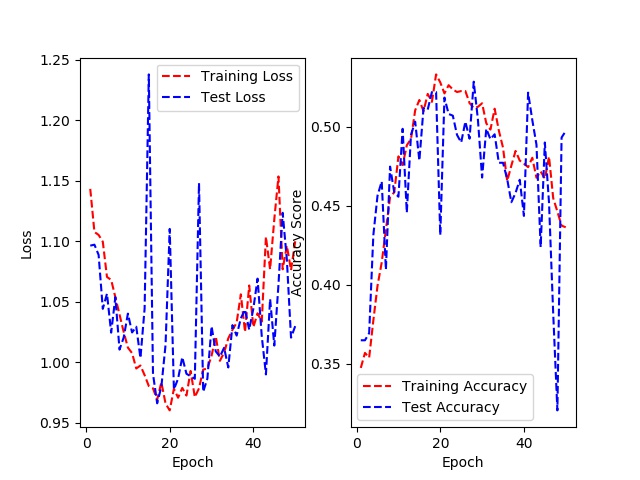
1. **ANALYSIS OF DESIGNED DNN**

The custom designed DNN was developed with the help of Tensorflow library. The model has the ability to perform image classification. It is only able to identify the class of a random uploaded image. It is able to predict the class only from a given set of class labels, provided at the time of feeding the ground truth. Only three classes were used for this task i.e. People, Cars and Heavy Vehicles. The model has dash number of layers.



**Fig. 4**

The above given figure 4 shows the architecture of the designed model. The model has in total 30 layers including the input and the output layers. If the architecture is followed carefully, it contains four blocks of Convolution Layers. Each of the four layers have two layers of 2D Convolution layer of same number of filters taken from the tensorflow library. The ‘ReLU’ is taken as the activation function at each stage. The kernel filter window is kept as 3x3. MaxPooling and Dropouts have been provided at essential stages. The Dense layer value is kept 3 at the end, this because only three classes are considered in this task. The loss function and the optimizer are categorical\_crossentropy and rmsprop respectively. The large number of layers have been added in order to overcome the underfitting conditions of the system.

The min batch size, epoch and valid split were 32, 50 and 0.30 respectively. During the commencement of the task, a number of 3,474 car images, 3,086 heavy vehicle images and 3,056 images of people were considered as the dataset.

**Fig. 5**

The above figure 5 shows the performance graphs of the trained model. Both the Epoch vs Loss and the Epoch vs Accuracy Score graphs have been plotted. The pattern of the plots resembles an example of underfitting. This because a smaller number of data used for this task. Using a large number of image data can fix this issue.

1. **ANALYSIS OF THE TRAINING PROCESS**

**YOLOv3:**

Yolov3 has a unique format for the annotation files. The annotations of the whole dataset is kept in a text file. In order to covert the drone dataset ground truth to Yolov3 format, at first it is converted to xml files, then to a csv file and then finally to the required format in a .txt file. All these objectives are fulfilled using several Python scripts. The training requires a configuration file and a file with anchor points. A pre-trained weight model is also utilised so that the training does not start from scratch. The training can be done both with the help of CPU or GPU. It can be toggled with the help of a single line of code in the Python script. At last the training is separated into two stages. The first stage trains the images from 0 to 40 epochs with batch size of 32. At the second stage, it continues the training to 60 epochs with a batch size of 8. The batch size is decreased in the second stage in order to prevent the RAM from getting crashed with full memory. Each of the stages stores its weights in a particular weight file in .h5 format.

**ResNet:**

The ResNet requires the image annotations to be in a .xml format. The annotations of the drone dataset were converted to the required .xml file format. Then with the help of a python code all the xml annotations were piled up in a csv file in order to feed it into the model. Normal training parameters were set to start the training procedure in ResNet. Each Epoch is counted as a checkpoint and all the weights are particularly stored as each file. Suppose if the epoch was set to 10 then the model created 10 files in .h5 format in order to save the weight files at every stage.

**Custom Designed DNN:**

* **Image Pre-Processing-** During the commencement of the task, a number of 1k car, bbkbknkb were considered as the dataset. Still this count was very less for the task. Image Augmentation process was used to increase the number of data to some extent. A python script was coded to create the mirror image of the original image using OpenCV module.

The above shown code helps to increase the number of images in the existing classes by two times.

The images were resized into a fixed dimension, which was 220x220. Every coloured image has 3 channels Red, Green and Blue. Training with all the 3 channels is very time consuming for the computer. Hence all the images were converted into grey-scale. Then the images were randomly shuffled and sorted into two pickled out files, X.pickle and y.pickle. X.pickle contains the data of the images in numpy array format and y.pickle contains the ground truth or the class labels.

* **Training-** The DNN was trained with 40 number of Epochs, the batch size as 32 and the train data as 70% and test data as 30%.

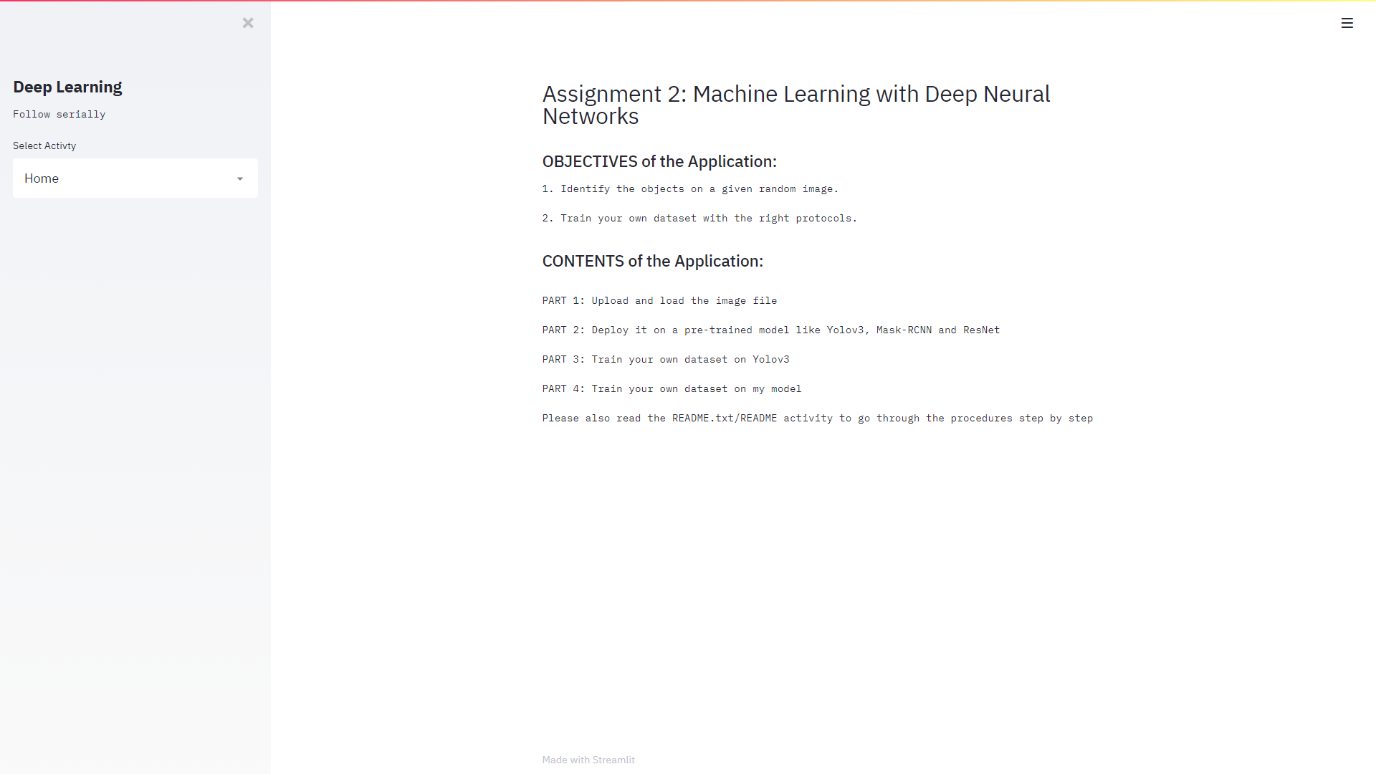


**Fig. 6**

The above figure 6 shows the fit command for starting the training process. All the training stats are recorded in a python variable called history. This is required for plotting the loss and the accuracy curves.

1. **ANALYSIS OF THE DESIGNED WEB BASED GUI**

The web-based GUI is developed with the help of Streamlit and python. It brings all the required objectives of this assignment in a nutshell. The application is able to upload a random image as a .jpeg file for prediction. Then all of the existing DNNs are implemented on the random image for prediction. It also has an option to show all the three outputs together in order to judge the performance of the detections. It can be found in ‘Which Is The Best?’ section of the application. One is even able to train their own dataset in this application.



**Fig. 7**

The above given figure 7 shows the user interface for the web-based application. The figure resembles the screenshot of the home page of the application. The centre of the page shows the page content. The side bar helps to navigate through the activities of the application.

A Readme section is provided to guide the user. Hence, the UI of this application is very user friendly.

1. **COMPARATIVE ANALYSIS AND PERFORMANCE ANALYSIS**

Each of the existing DNNs mentioned in section 2 of the report have a metric to evaluate the outputs. Each of the DNNs have Average Precision in their own detected outputs. It numerically tells that how precise was the algorithm to identify the objects.



**Fig. 8**

Out of all the existing DNNs, the Mask R-CNN algorithm shows a better outcome. This is because the model is able to detect the objects. It is also able to provide good values of Average Precision scores. The best part is that it is able to mask the edges of the classified objects. Hence, the mask-rcnn model shows acceptable results for this assignment.

1. **LIST OF MY CONTRIBUTIONS**

* Part 1 (Uploading and loading image data)
* Part 2 (All sections shown)
* Part 3 (Research and Assembling the code for both Yolov3 and ResNet)
* Part 4 (All parts)
* Part 5 (Web-Application)
* Converting VisDrone annotations to required formats

1. **REFERENCES**

* Nishad, G. (2020). *Garima13a/YOLO-Object-Detection*. [online] GitHub. Available at: https://github.com/Garima13a/YOLO-Object-Detection [Accessed 01 Mar. 2020].
* GitHub. (2020). *matterport/Mask\_RCNN*. [online] Available at: https://github.com/matterport/Mask\_RCNN [Accessed 02 Mar. 2020].
* He, K., Gkioxari, G., Dollár, P. and Girshick, R. (2017). *Mask R-CNN*. [online] arXiv.org. Available at: <https://arxiv.org/abs/1703.06870> [Accessed 02 Mar. 2020]..
* Zhang, X. (2018). *Simple Understanding of Mask RCNN*. [online] Medium. Available at: https://medium.com/@alittlepain833/simple-understanding-of-mask-rcnn-134b5b330e95 [Accessed 04 Mar. 2020].
* Analytics Vidhya. (2019). *Step-by-Step Implementation of Mask R-CNN for Image Segmentation*. [online] Available at: https://www.analyticsvidhya.com/blog/2019/07/computer-vision-implementing-mask-r-cnn-image-segmentation/ [Accessed 05 Mar. 2020].
* VisDrone (2020). *VisDrone/VisDrone-Dataset*. [online] GitHub. Available at: https://github.com/VisDrone/VisDrone-Dataset [Accessed 07 Mar. 2020].
* Kumaar, S. (2020). *dronefreak/VisDrone-dataset-python-toolkit*. [online] GitHub. Available at: https://github.com/dronefreak/VisDrone-dataset-python-toolkit [Accessed 07 Mar. 2020].
* Samjith888 (2020). *Samjith888/Keras-retinanet-Training-on-custom-datasets-for-Object-Detection-*. [online] GitHub. Available at: https://github.com/Samjith888/Keras-retinanet-Training-on-custom-datasets-for-Object-Detection- [Accessed 10 Mar. 2020].
* AntonMu (2020). *AntonMu/TrainYourOwnYOLO*. [online] GitHub. Available at: https://github.com/AntonMu/TrainYourOwnYOLO [Accessed 12 Mar. 2020].
* Patidar, P. (2019). *patidarparas13/Basic-Object-Detection*. [online] GitHub. Available at: https://github.com/patidarparas13/Basic-Object-Detection [Accessed 15 Mar. 2020].
* pylessons.com. (n.d.). *Python Lessons*. [online] Available at: https://pylessons.com/YOLOv3-custom-training/ [Accessed 18 Mar. 2020].
* GitHub. (n.d.). *pythonlessons/YOLOv3-object-detection-tutorial*. [online] Available at: https://github.com/pythonlessons/YOLOv3-object-detection-tutorial/tree/master/YOLOv3-custom-training?fbclid=IwAR11x3OK6jY-oX\_wY8dd-bG52-ZJMpixkpHg6tuu0D\_ssxMk-vH6squjg-g [Accessed 18 Mar. 2020].
* RomRoc (2020). *RomRoc/objdet\_fizyr\_colab*. [online] GitHub. Available at: https://github.com/RomRoc/objdet\_fizyr\_colab [Accessed 21 Mar. 2020].
* Pythonprogramming.net. (2017). *Python Programming Tutorials*. [online] Available at: https://pythonprogramming.net/convolutional-neural-network-kats-vs-dogs-machine-learning-tutorial/ [Accessed 21 Mar. 2020].
* Pythonprogramming.net. (2019). *Python Programming Tutorials*. [online] Available at: <https://pythonprogramming.net/loading-custom-data-deep-learning-python-tensorflow-keras/?completed=/introduction-deep-learning-python-tensorflow-keras/> [Accessed 22 Mar. 2020].
* pythonprogramming.net. (n.d.). *Python Programming Tutorials*. [online] Available at: https://pythonprogramming.net/convolutional-neural-network-deep-learning-python-tensorflow-keras/?completed=/loading-custom-data-deep-learning-python-tensorflow-keras/ [Accessed 25 Mar. 2020].
* pythonprogramming.net. (n.d.). *Python Programming Tutorials*. [online] Available at: https://pythonprogramming.net/using-trained-model-deep-learning-python-tensorflow-keras/?completed=/tensorboard-optimizing-models-deep-learning-python-tensorflow-keras/ [Accessed 25 Mar. 2020].
* jesse\_jcharis (2020). *Building A Face Detection App with Streamlit and OpenCV*. [online] JCharisTech. Available at: https://blog.jcharistech.com/2020/02/23/building-a-face-detection-app-with-streamlit-and-opencv/ [Accessed 27 Mar. 2020].