

Data Science Project

Image Classification to Detect Road Signs using CNN to Train Self-Driving Cars

Kaushik Kasthurirangan

Maria Pradeep Irudayaraj

Vignesh Kannan

Yagna Dheepika Venkitasamy



TABLE OF CONTENTS

INTRODUCTION	2
METHODOLOGY	3
DATA COLLECTION	3
MODEL EXPERIMENTS	4
ADDITIONAL EXPERIMENTS	13
FUTURE WORK	13
BUSINESS IMPLICATION	14
CONCLUSION	14
REFERENCES	14



INTRODUCTION

Recently, deep learning methods have shown superior performance for many tasks such as image classification and object detection. One particular variant of deep neural networks, convolutional neural networks (CNNs), have shown their strengths for tasks including image classification, localization and detection.

With the modern economy boosting the usage of self driving cars, increased road users and varying road behaviours, it has become more important to strictly adhere to the traffic sign rules even for self-driving cars. By following the same, we can reduce the number of accidents on the road day by day and also increase the user efficiency on the driving behaviour.

Autonomous vehicles must also abide by road legislation and therefore recognize and understand traffic signs. Self-driving cars are becoming very popular due to the efforts of companies like Tesla, Google in automating their electric vehicles. To become level 5 autonomous these cars would have to correctly identify traffic signs and follow traffic rules. After identifying these traffic signs they should also be able to take correct decisions suitably.

Traditionally, standard computer vision methods were employed to detect and classify traffic signs, but these required considerable and time-consuming manual work to handcraft important features in images. Instead, by applying deep learning to this problem, we create a model that reliably classifies traffic signs, learning to identify the most appropriate features for this problem by itself.

Traffic signs may be divided into different categories according to function, and in each category they may be further divided into subclasses with similar generic shape and appearance but different details. This suggests traffic-sign recognition should be carried out as a two-phase task: detection followed by classification. The detection step uses shared information to suggest bounding boxes that may contain traffic-signs in a specific category, while the classification step uses differences to determine which specific kind of sign is present.

This trend motivated us to look deep into this area and we happened to stumble upon a very interesting and useful use case in the self-driving car industry – to recognise and detect traffic signs on the road .

METHODOLOGY

Problem Statement: In order to classify the road signs to be recognised by the self-driving cars, we had to divide our problem into 4 parts each of which has varying levels of complexity.

Level 1: We classified images based on the presence of the road sign or not in the images as a binary class classification problem.

Level 2: We classified images based on 3 classes of road signs like warning signs, guide signs and regulatory signs and treated the problem as multi class classification.

Level 3: We classified images based on the type of the road sign present in the image that are clicked from far thus treating this as a multiclass classification problem with increased level of complexity.

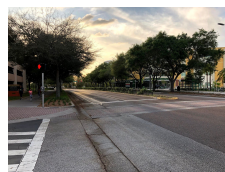
Level 4: Identify, classify and generate a bounding box using object detection techniques on images using pretrained models / algorithms.

DATA COLLECTION & DEMOGRAPHICS

For Level 1, we created our own dataset by trying to replicate the self driving cars image capture that contains visuals on the road which includes road signs, traffic signals, objects on road using a dashboard cam on a regular car. We then also included images of roads, highways, motorways, downtown areas without road signs in them. Level 1 was based on training the model to do binary class classification for images with sign and images without sign. We started off with 100 images of 50 images each class to train the model with reduced complexity.



With sign



Without Sign

For Level 2, we gathered pictures from google images with slightly higher levels of complexity to include images from different types of road signs like guide, regulatory and warning signs. With this we trained the model to identify the images into multiclass classification of what type of class

the image belongs to. We gathered 180 images with 60 images of each class to train the model with medium complexity.



Guide sign



Regulatory sign



Warning Sign

For Level 3, we collected images from google with increased complexity with road signs that are far away for the model to classify the images to which of the multi classes they belong to. We trained the model with 246 images of which 85 belongs to guide sign class, 91 belong to regulatory sign class and 70 belong to warning sign class. The images have some noise like other objects, trees, some disturbances that are trying to impact the performance of the model.



Warning sign



Regulatory sign



Guide Sign

Libraries used for processing data: The libraries used in this project were Numpy, OpenCV, Matplotlib, Scikitlearn, Tensorflow, Keras, PyQt5, lxml, libxml2.

MODEL EXPERIMENTS

Level 1:

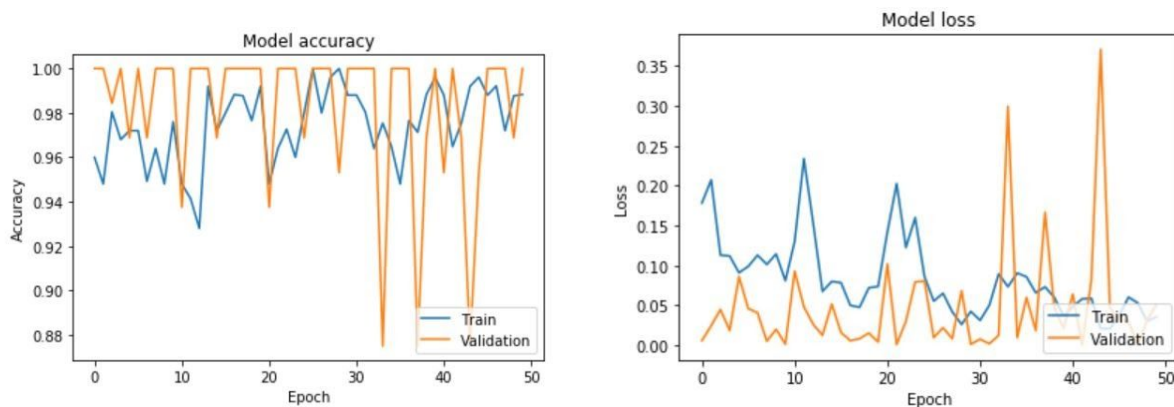
Procedure:

- For the first level of the project, our goal was to build an image classifier that could distinguish whether a given image was a traffic sign or not i.e. binary classification.
- We started off by loading our training & test images using a custom function. Since OpenCV reads color images in the BGR channel format as opposed to RGB, we convert the color channels during reading.

- Following this, the pixel representation of the images are converted into a series of numpy arrays of dimension - (count of images, image_height, image_width, no. of color channels).
- The numpy arrays containing the training data are fed to an Image Augmentation function to apply transformation to our images so that they may be more representative of traffic signs in real life such as random zooming, shearing, shifting the image along the axes etc.
- The sequential model is initialized and we specify our CNN model's various layers.
 - Two sets of convolutional layers were initialized with a (3 x 3) kernel to help the model identify features and shapes within the traffic sign.
 - The convoluted layers were subsequently passed to the "relu" activation function.
 - Following this, the layers were passed through a MaxPooling function to reduce the dimensionality and allow the model to make assumptions about features within the sub-regions.
 - The data was then flattened before passing it to through the dense layers for generating the output vectors.
 - A dropout layer was added to the model to prevent overfitting, given that we had a limited training sample to work with.
 - The final dense layer was connected to the "sigmoid" activation function to get a two-class output.
- We trained the model through continual tweaking of our CNN and once we obtained results that were desirable, we saved the model weights in the HDF5 format.

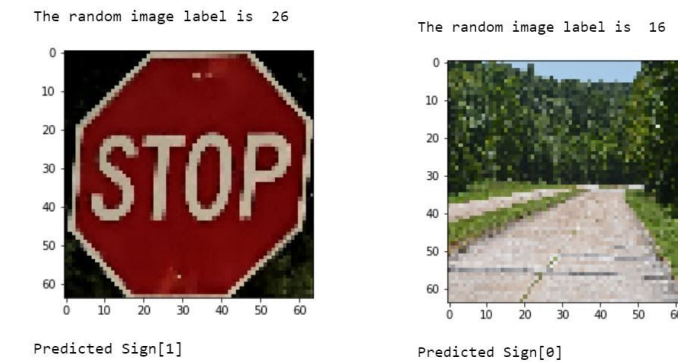
Results:

Below are the training and validation accuracies and losses and we also used it to predict data from our test set. For some of them it gave correct results with a model accuracy of 96%.



Test prediction:

When trying to predict the binary classes, our model did pretty well. We had an overall accuracy of 95% with the same precision and recall values. The below image shows a random prediction by the model.



Classification report & Confusion matrix:

The confusion matrix and the classification report of the test predictions are given below. The model did pretty well in predicting the true positives and true negatives of both the classes. The model might have mis-predicted without sign due to some noise in the image.

Confusion Matrix			
		Predicted	
		Without Sign	With Sign
Actual	Without Sign	20	0
	With Sign	2	19
		Accuracy	95%

	precision	recall	f1-score	support
0	0.91	1.00	0.95	20
1	1.00	0.90	0.95	21
accuracy			0.95	41
macro avg	0.95	0.95	0.95	41
weighted avg	0.96	0.95	0.95	41

Level 2:

For level 2, we added a different set of images with localized pictures of the road signs along with modifying parameters like number of epochs, batch size, steps per epoch etc. Validation Accuracy slightly dropped due to increased complexity. We chose the guide signs, regulatory and warning signs as they are brighter in color and make it easier for the model to differentiate and predict. We had the following classification:

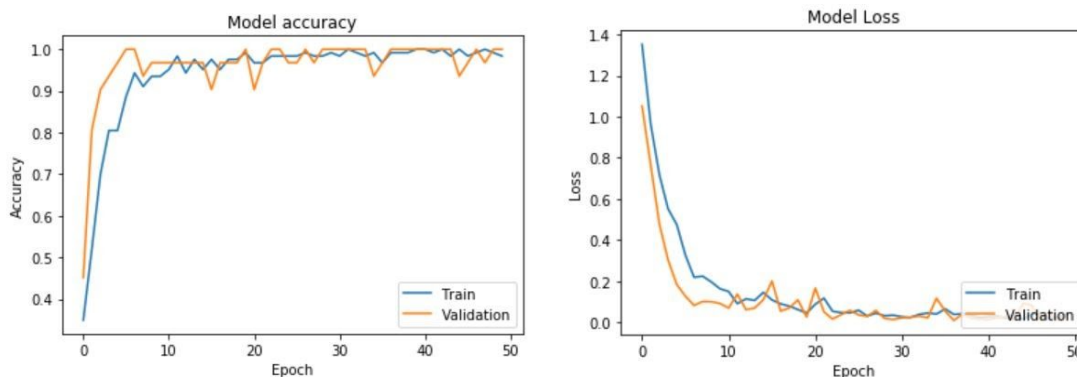
Guide Signs - Class 0, Regulatory Signs - Class 1, Warning Sign - Class 2.

The model was built with the same CNN structure used in Level 1, with changes being made to the activation function of the final Dense layer & loss function in order to support multi-class classification.

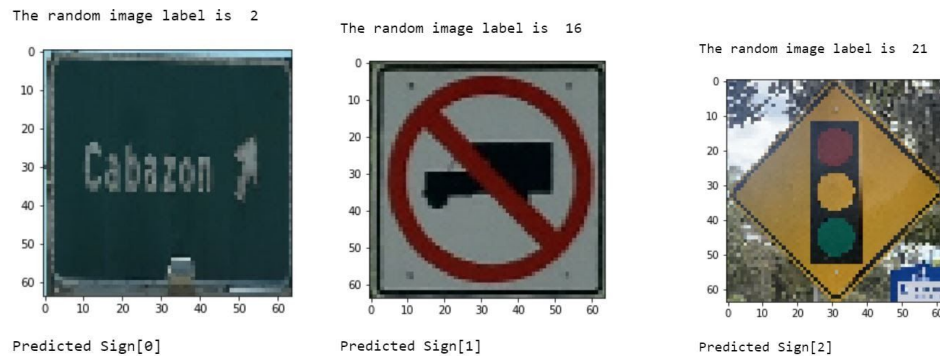
Once we obtained desirable results after multiple iterations of tweaking the hyperparameters & training, the final model weights were saved onto a HDF5 file for reproducibility.

Results:

Below are the training and validation accuracies and losses and we also used it to predict data from our test set. For some of them it gave correct results with a model accuracy of 83%.



When trying to predict the multiple classes, our model did slightly worse than the level 1. We had an overall accuracy of 83% with the precision as 83% and recall 87%. The below image shows a random prediction of all the 3 classes by the model.



Classification report & Confusion matrix:

The confusion matrix and the classification report of the test predictions of level 2 are given below. The model did pretty well in predicting the true positives and true negatives of all the classes. The sensitivity of the model might be affected due to the presence of some warning signs in the guide signs and some other noise in the data.

Confusion Matrix				
		Predicted		
		Guide Sign	Regulatory Sign	Warning Sign
Actual	Guide Sign	6	0	0
	RegulatorySign	3	10	1
	WarningSign	1	0	9
		Accuracy	83%	

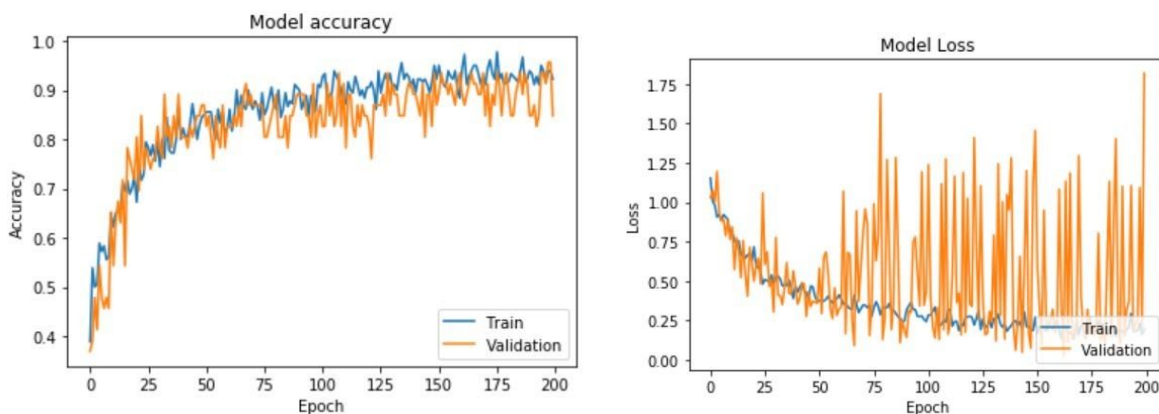
	precision	recall	f1-score	support
0	0.60	1.00	0.75	6
1	1.00	0.71	0.83	14
2	0.90	0.90	0.90	10
accuracy			0.83	30
macro avg	0.83	0.87	0.83	30
weighted avg	0.89	0.83	0.84	30

Level 3:

In this level, we aim to increase the complexity by adding images of different classes from a farther distance rather than localized images we had for previous levels. The images used for this level had more complex features like trees, cars, bushes, mountains, people and so many others in the background along with the road signs. The images were taken in such a way to see whether our model performs well with far images having various features to learn from the background along with the road signs. A similar procedure was followed as done in Level 2 to build the model by processing the images and feeding the far images as input. The hyperparameter tuning was performed by modifying the parameters like Activation function and Optimizer to improve the accuracy and performance on the prediction of the model. We are doing MultiClass Classification with 3 classes namely Guide Signs(Class 0), Regulatory Signs(Class 1) and Warning Sign(Class 2).

Results:

Below are the training and validation accuracies and losses and we also used it to predict data from our test set. For some of them it gave correct results with a model accuracy of 92%.



When trying to predict the multiple classes, our model did well. We had an overall accuracy of 92% with the precision as 93% and recall 94%. The below image shows a random prediction of all the 3 classes by the model.



Classification report & Confusion matrix:

The confusion matrix and the classification report of the test predictions are given below. The model did pretty well in predicting the true positives and true negatives of all the classes. The model might have mis-predicted the regulatory signs with the warning signs due to some noise in the image data.

Confusion Matrix				
		Predicted		
		Guide Sign	Regulatory Sign	Warning Sign
Actual	Guide Sign	10	2	0
	RegulatorySign	0	8	0
	WarningSign	0	0	5
		Accuracy	92%	

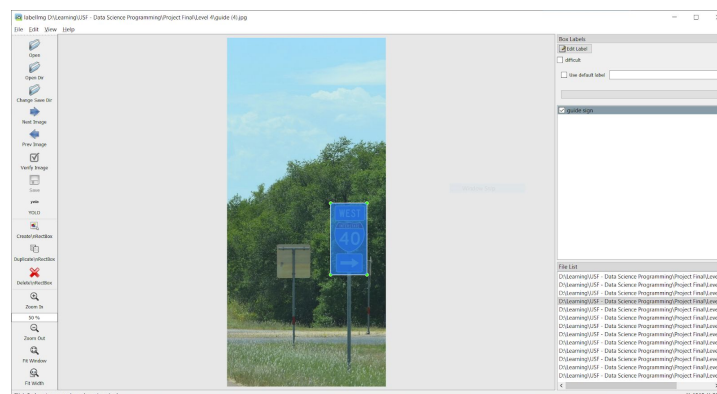
	precision	recall	f1-score	support
0	1.00	0.83	0.91	12
1	0.80	1.00	0.89	8
2	1.00	1.00	1.00	5
accuracy			0.92	25
macro avg	0.93	0.94	0.93	25
weighted avg	0.94	0.92	0.92	25

Level 4:

In order to build the image classifier, we used localized images of traffic signs i.e. images that were either cropped to represent only the sign or images that were cropped to represent the sign and a few other objects in the image from a distance. However, in real life traffic sign recognition is more of an object detection problem rather than a classification problem. We experimented with tutorials to set up the YOLO object detection algorithms to see if we could use the pretrained network to learn from our custom dataset. Our experiments used the YOLO - Darknet framework found at <https://github.com/alexeyab/darknet>

Darknet is an open source neural network framework written in C and CUDA. It is fast, easy to install, and supports CPU and GPU computation. The GitHub repo contained all the pre-trained weights and the various layers in the neural network needed to perform object localization, classification & detection.

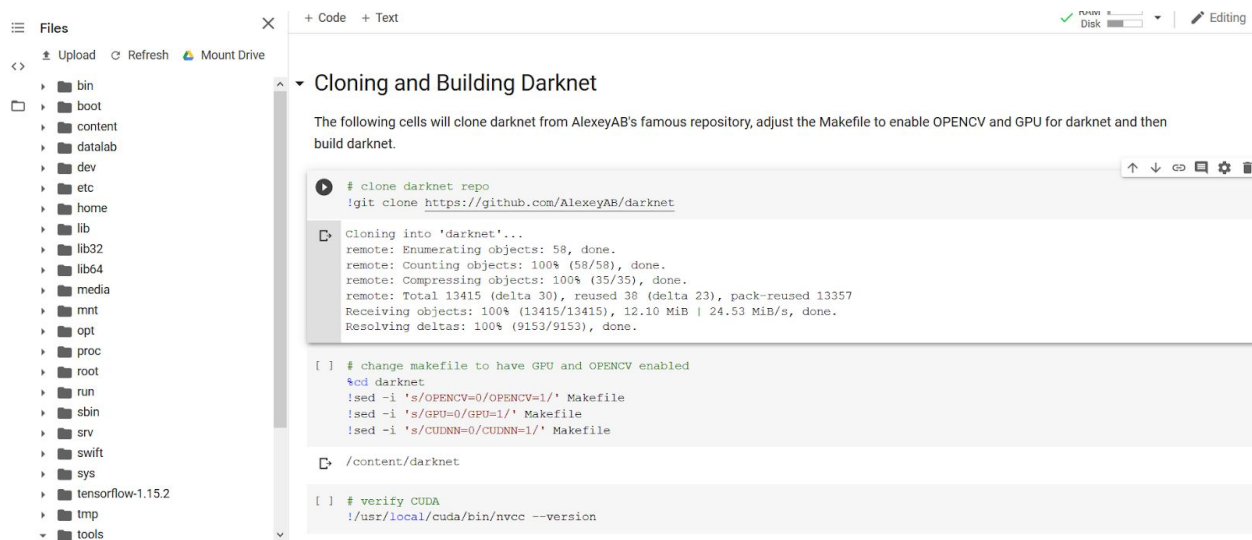
To get the YOLO v3 algorithm to work on our custom dataset, we needed to annotate the region of interest in our images, which in our case was three different classes of traffic signs. We used the Labellmg tool found here <https://github.com/alexeyab/darknet> and manually generated annotations for close to 200 images.



Generating annotations on Labellmg

The YOLO algorithm requires a minimum of 4000 iterations for a single class detection problem. For anything greater than a single class, the algorithm mandates that the **Iterations = No. of Classes * 2000**. This meant that we needed to run our algorithm on the custom dataset for 6000 iterations (since it is a 3 class problem).

We leveraged on the free GPU computation power provided on Google Colab to run the YOLO training on a virtual machine. This was made possible thanks to the code & tutorials provided by **The AI Guy** (<https://youtu.be/10joRJt39Ns>) on YouTube. The Darknet repo, modified config files, annotated images and the weights were cloned onto the virtual machine.



```
# clone darknet repo
!git clone https://github.com/AlexeyAB/darknet

Cloning into 'darknet'...
remote: Enumerating objects: 58, done.
remote: Counting objects: 100% (58/58), done.
remote: Compressing objects: 100% (35/35), done.
remote: Total 13415 (delta 30), reused 38 (delta 23), pack-reused 13357
Receiving objects: 100% (13415/13415), 12.10 MiB | 24.53 MiB/s, done.
Resolving deltas: 100% (9153/9153), done.

[ ] # change makefile to have GPU and OPENCV enabled
%cd darknet
!sed -i 's/OPENCV=0/OPENCV=1/' Makefile
!sed -i 's/CPU=0/GPU=1/' Makefile
!sed -i 's/CUDNN=0/CUDNN=1/' Makefile

/content/darknet

[ ] # verify CUDA
!/usr/local/cuda/bin/nvcc --version
```

Running the algorithm on Google Colab

However, a class ID mismatch owing to a mistake in the annotation process nullified our attempt to get the algorithm to run on Google Colab. In addition to this, it takes an extremely long time (ETA of 7 hours) for the training to complete. Owing to these reasons, we are unable to present our output for the object detection level. We will be pursuing this in future iterations.

ADDITIONAL EXPERIMENTS

Open CV2

We also tried rendering a video from the images in the dataset and tried using open cv2 with which we found the shape of the object in the video. OpenCV is a huge open-source library for computer vision, machine learning, and image processing. By using it, we were able to process images to identify objects. When it is integrated with various libraries, such as Numpy, python is capable of processing the OpenCV array structure for analysis. To identify image pattern and its various features we use vector space and perform mathematical operations on these features.

We were able to draw a line surrounding any particular image using open CV2 but we could not further into object localization due to the limitations of using a pre trained model. We tried to overcome this by exploring more options with faster RNNs and MobileNet.

FUTURE WORK

We laid a baseline model of image classification that can be used for training the self driving cars in classifying the road signs while driving but would like to improve it furthermore with real time complexities. We also want to further train the model for object detection in images and videos. Though there are many resources available in the industry that have successfully implemented similar models in training the self driving cars and deploying the outcome, we'd like to probe further given we have more time, higher computing resources and more data to train the model with. The following are more complex compared to the level we are in now, so we are just listing what can be done to further improvement in the same line of thought:

- Being able to detect objects on road not just as plain signs, but as what type of signs (guide, regulatory, warning, etc)
- Being able to detect road signs even with adversarial conditions with boards closely resembling the traffic signs on road.
- Being able to detect multiple road signs on junctions and on highways accurately.
- After incorporating all the above successfully, being able to implement this model to detect road signs for car companies that are deploying self-driving cars on real time.
- Being able to improve the model by enabling your model to parse images from google to train the model with a large set of available images from the web

BUSINESS IMPLICATIONS

After the successful implementation of our existing model, it can be used to further develop into an object detection model to detect different objects, signs, pedestrians, vehicles on road and learn the objects to make appropriate decisions in implementing the appropriate user behaviour.

Additionally, our model, when implemented with an acceptable amount of accuracy can be used in a wide variety of cases in developing the autonomous car driving industry with real time complexities and challenges in the deployment.

CONCLUSION

Different models for detection, classification and recognition of road signs is presented, specifically developed for defining an automated procedure. This procedure, when the user is appropriately instructed, is able to highlight differences in the signs with respect to the requirements in terms of position and direction, as set by the regulation. In particular, when operating in an urban area the proposed algorithm allows one to recognize, and automatically classify, almost all those road signs that are visible and correctly positioned (no rotation or occlusion). A possible improvement of the performances offered by the proposed algorithm may deal with detection of those roads signs that are not perfectly visible. This problem is less serious when operating in a suburban area or highways, where the signs are more visible.

REFERENCES

- [Traffic Sign Classification with Keras and Deep Learning](#)
- [Keras Tutorial: How to get started with Keras, Deep Learning, and Python](#)
- [A gentle guide to deep learning object detection](#)
- [How to Load Large Datasets From Directories for Deep Learning in Keras](#)
- [A Gentle Introduction to Object Recognition With Deep Learning](#)
- [YOLO: Real-Time Object Detection](#)
- [AlexeyAB/darknet: YOLOv4 \(v3/v2\) - Windows and Linux version of Darknet Neural Networks for object detection \(Tensor Cores are used\)](#)
- [tzutalin/labellmg: !\[\]\(5774573cf757c446bb08af21f46b2969_img.jpg\) Labellmg is a graphical image annotation tool and label object bounding boxes in images](#)
- [The AI Guy](#)
- [Python Programming Tutorials](#)