****

**Master Thesis**

**Master of Science (M.sc)**

**Department of Tech and Software**

**Major: Data Science**

**Topic: A Safety Vehicle Detection System using YOLOv5**

Author: Girish Venugopalan Nair

Matrikel-Number:

First supervisor: Prof Dr. Raja Hashim Ali

Second supervisor: Prof Dr. Talha Ali Khan

Submitted on:

**Bibliographic Description and Presentation**

Bachelor thesis 2022, 90 pages, 20 figures, 4 tables, 101 references.

University of Europe for applied sciences, Department of Tech and Software

.

Content Overview of this paper:

•

**Statutory Declaration:**

I hereby declare that I have developed and written the enclosed Bachelor Thesis completely by myself and have not used sources or means without declaration in the text. I clearly marked and separately listed all the literature and all the other sources which I employed when producing this academic work, either literally or in content. I am aware that the violation of this regulation will lead to failure of the thesis.

Berlin, ..............................................

Place, Date Signature

**Abstract**

Autonomous vehicles are the trend nowadays and many of manufacturers introduce innovations to boast their intelligent vehicles. With the advent of science, many assistive technologies emerged to help the driver during long drives like hands-free driving, cruise control, auto adaptive headlights, etc. Artificial intelligence also takes its position in assisting the commutation to be safer and more effortless with technologies like self-driving, lane detection, collision detection, etc. Although collision detection has been there for a while in the automotive industry for many famous manufacturers like Mercedes, BMW, Audi, and Tesla generic danger prediction and warning have not been explored much. This research is proposing to predict threats in advance so that the driver is aware before it is too late to make a decision. The idea here is to use Ultralytics Yolov5 for quick object detection which is a Convolutional Neural Network designed by Ultralytics. Annotated road surface data RDD2020 was used for performing the tests. The data contains classifications like potholes, debris, and cracks which are used to train the model and then later used to warn the driver. Yolov5 has different models, and a comparison of the models was performed. An analysis of the hyperparameters was done to fine-tune the detection results.

Keywords: road damage; convolutional neural networks; semantic segmentation; Yolov5

**Table of Contents:**

**Introduction**

**Motivation**

Commutation has always been a need of man to achieve his goals, to be productive, to socialize, to engage in business and many more. Various modes of travel also have been invented, yet the road remains the most widely used. The safety of vehicles is also regulated by the governments across the world and manufacturers comply with guidelines and standards Innovations like seat belts and driver assistance technologies have greatly improved road safety. However, human errors are still prone, and the time is right to discuss how we can reduce manual errors by timely warning using intelligent driver-assistive technologies. During a span of 3 years in US around 200,000 accidents occurred due to road debris which could have been avoided if a proper warning system was available in the vehicle.

**Objective**

There have been various attempts of solving this problem of vehicle safety using different sensors, radars, etc. The challenges are the detections should be real-time and should be efficient enough to detect irrespective of the size of the threat being detected. A method well discussed is semantic segmentation which classifies objects at the pixel level.

**Scope**

The vehicle safety system uses visual data collected from high-tech cameras. As with a better understanding of road landscapes, drivers on the side of the road will have a better sense of the driving space available to them. A self-driving car equipped with this technology can detect dangers such as objects in its path and defects in the road. It is possible that sophisticated sensors, such as cameras, can provide visual input that allows it to identify one or more traffic lights. It's the same with road scenes: a better understanding of the surrounding environment helps drivers better utilize the side-of-the-road terrain for vehicle manoeuvrings.

Historically, the identification of traffic signs has been based on color and shape patterns, with two connected phases: detection and classification. Traffic signs are detected in a picture after numerous pre-processing procedures, such as data transformation and normalization, which consists of defining regions of interest (ROI) based on color segmentation and "sliding window" approach. Following the step of pattern recognition, the classification phase classifies each sign feature into categories such as "speed limits" and "pedestrian crossing." The likely traffic indicators are then classified using a shallow neural network (a multilayer perceptron, or MLP). Some researchers have employed shallow classifiers, like support vector machines (SVMs) or random forests, in conjunction with local descriptors, like the histogram of oriented gradient (HOG), for successful feature extraction and classification. Hmida et al. demonstrated a traffic sign identification system based on linear SVMs and the MNIST dataset, for instance. Gecer et al. developed a high-performance technique for identifying traffic signs based on blob detectors and SVM classifiers, which boosted the model's color discriminating power by achieving a 98.94% accuracy rate. Due to the wide diversity of road signs in unexpected locations, hidden and diminutive road signs, and varying weather conditions (e.g., shadows and lightning), it is difficult to discern them using conventional methods; therefore, deep learning techniques are employed.

**Method**

Using the YOLOv5 model, a vehicle safety system with an effective obstacle detection mechanism and increased speed is created. The video of the driver's vision is captured by a camera and sent to the model for precise and rapid object recognition on urban roadways. Each frame of the supplied video is processed as an input to the object recognition and detection algorithm (YOLOv5). In the algorithm, each frame is processed in three stages: backbone, neck, and head.

(i)Backbone: CSPDarknet

(ii)Neck: PANet with spatial pyramid pooling (SPP)

(iii) Head: YOLO layer

Different features of Driver Assistance Systems and Autonomous vehicles are the subject of numerous studies. Using a laser scanner, Chen et al. developed an IoT-based occlusion technique termed multiple targets tracking in occlusion region with interactive object models in urban contexts to solve the challenge of object recognition for autonomous vehicles. The difficulty in identifying the object resulted from the various observed shapes on each laser scan. Consequently, the proposed system is created using a machine learning approach and YOLOv5 to mitigate the occlusion problem.

Yolov5 will be trained with different annotated data sets for road surface detection and road signs and used for predictions and generating warnings. During this research, the performance of Yolov5 is evaluated by gathering various metrics and compared.

**Challenges in Computer Vision**

Typically, any computer vision problem falls into any of the following six categories or a combination of them.

1. Classification: To identify which class out of the given k class an image belongs to.
2. Localization: To identify the bounding boxes for all instances for a given class k
3. Detection: To identify the bounding box of all instances of k classes
4. Semantic Segmentation: Pixel classification into k classes
5. Instance Segmentation: Classify pixels into k classes and identify the instance.
6. Content-based Image Retrieval: Identify u images like a given image x from a list of n images.

**Object Detection**

**Convolutional Neural Networks**

Artificial Neural Networks are used to simulate the decision process of biological neurons using computational networks. They have multiple layers (at least three). All the neurons in a layer are connected to every neuron in the adjacent layers. This is comparable to weighted graphs in which nodes are neurons and connections between the neurons are the directed edges of the graph. Depending upon the prediction validations the weights are adjusted to make further predictions more accurate. This process is called learning and is repeated several times until the errors in the prediction is very small or acceptable after a certain number of iterations called epochs.

Convolution Neural Networks (CNNs) are a category of deep learning containing layers which extract feature maps from images using different kernels. A convolution layer processes the input image using a defined window. The window can be parametrized to extract features. This windows is called a filter. It gives output of only the features it has identified in the input image. This output is referred to as a feature map. Typically multiple such filters are applied. The filters are not explicitly defined but the network automatically learns the parameters during training. Once the feature mappings are created , the depth of the feature map stack decided by the number of layers of filter it has. The filters could have different with and height but the depth should match with the input.

Then there are the pooling layers which does the dimensionality reduction. Pooling layers compresses the spatial information. There are different types of pooling called average pooling , min poooling and max pooling. The max pooling technique simply yields the maximum value in a window per scanned location. This is the most widely used pooling technique. Another technique called global pooling matches the input dimensions and thus compresses the size of feature mappings into a single value. Strided convolution is a learned downsampling than explicitly mentioning the technique. The stride length defines the shifting unit size.

The purpose of layers can also differ such as dropout and dense layers. In convolution layer the connection of a neuron is only to a local area of input neurons to reduce the learning parameters while going into deeper layers. Usually, the architecture would consist of a stack of many convolution layers, pooling layers followed by fully connected layers. Passing of the layer from the input to the output is called forward propagation. They are very effective and the layers in the network which are closer to the input learn low level features while layers deeper learn high order-features. They learn the spatial hierarchies of features with the help of backward propagation adaptively.

Typically, a pooling layer follows one or two convolution layers which forms a block together and it is repeated until the feature map becomes small enough add traditional layers. The reshaping of a multidimensional layer to a one dimensional fully connected layer is called flattening.

**Image Segmentation**

Segmentation is the process of classifying the pixels of an input image into k classes. While semantic segmentation just identifies the k classes , instance segmenetation identifies every instance of a class. It uses Fully Convolutional Network (FCN) which is a CNN without fully connected layers(FC). It can be designed as an encoder follower by a decoder and in Yolo both are FCN. Yolo is regression based and image is split into ‘s x s‘ grid cells .An object whose centre matches with a gird cell is predicted. The CNN predicts as follows;

* B number of bounding boxes (x, y, w, h). (x, y) is identified as the centre of the bounding box relatieve to the cell position. Confidence score is calculated as IOU of predicted bounding box with ground truth bounding box.
* C conditional probabilities are also predicted for each grid cell (one per class)

Prediction of CNN is (S, S, (B \*5+ C)); since there are 5 elements for boundary box. Object detection draws a bounding box around the objects detected in a processed image. However, image segmentation goes one step further to identify the outline of the object. Modelling for this is very difficult and should be applied only it is necessary.

**Image Classification**

Image classification is a classic machine learning problem. [HERE] Although it is very easy for humans to recognize objects like handwriting, warning signs, people, and things, it is a complex process to make computers recognize objects from mere videos or images. There are a variety of techniques that have been applied to solve this problem, like Harr Cascade Classifier, Template Matching, State Vector Machines, etc. However, Convolution Neural Networks (CNNs) generally have been very effective in such complex tasks involving unseen images. The image selected for CNNs processing will be analysed based on its height, width, and channel (grayscale or RGB). The height and width of the image together represent the number of pixels of the image. Convolutional layers in the network process the image and generate a smaller set of features which will be passed to the next layer. Pooling layers perform the down sampling along the spatial dimensions. The fully connected layer does the classification by assigning scores by assessing the features extracted from the previous layer.

Image classification algorithms determine the class of the image while image detection algorithms draw a bounding box around the object found in the image. There can be many bounding boxes based on the number of objects identified in the image. To cater to this, the neural network needs a variable-size output layer. This is not in accordance with the normal convolutional neural network. The R-CNN algorithm by Girshick et al. [HERE] extracts 2000 regions from the image which is fed and then proposed regions are resized and passed into a CNN. The CNN would then classify the input regions. A modified form of this algorithm is the Fast R-CNN.

**YOLO**

The algorithms mentioned in the previous sections have the drawback that they consume too much time for pre-processing. In the case of R-CNN, it defines regions, and then the regions are sent for classification. YOLO (You only look once) on the contrary inputs an image split into SxS default regions and all the regions are processed at the same time. The regions with a confidence level above a certain threshold will be candidates to identify objects within them.

**Releases**

YOLO was first released in May 2016 by Joseph Redmon. In 2017, a better version named YOLO 9000 was released. The most popular and stable version named YOLOv3 was released in 2018 with t paper “YOLOv3: An Incremental Improvement”. “[YOLOv3: An Incremental Improvement](https://arxiv.org/pdf/1804.02767.pdf)”. In April 2020, Alexey Bochkoviskiy introduced YOLOv4: Optimal Speed and Accuracy of Object Detection. YOLOv4 outperformed YOLOv3 by a high margin.

Chart, line chart

Description automatically generated

Source: YOLOv4 paper.

On 9th June 2020, Glenn Jocher, an unofficial author, released YOLOv5 based on PyTorch with exceptional improvements. YOLOv5 is so far the best compared to the previous versions.

Chart

Description automatically generated

source: https://github.com/ultralytics/yolov5

**YOLOv5 Model comparison**

Calendar

Description automatically generated

**Source:** https://github.com/ultralytics/yolov5#user-content-pretrained-checkpoints

**Custom Object Detection with Yolov5**

Training YOLOv5 can be defined in the following steps:

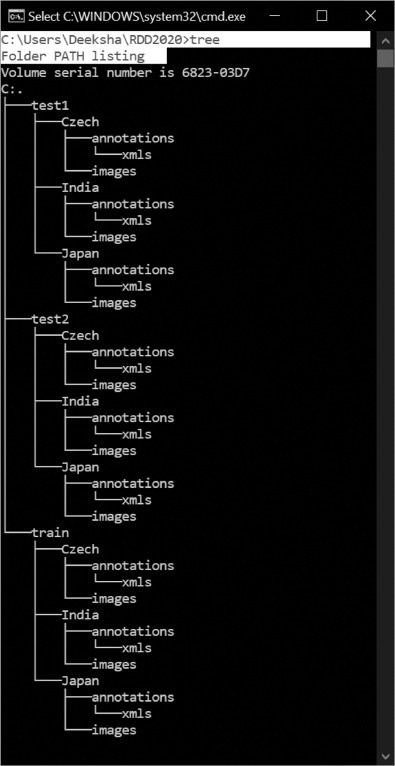
1. Prepare the Dataset
2. Setup the Environment
3. Configuring the parameters in the model
4. Train
5. Evaluate

**Prepare the Dataset**

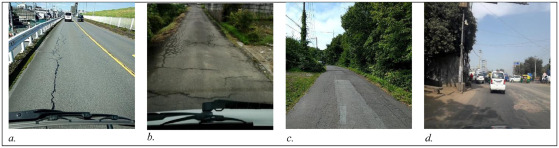
Initially, MS COCO dataset was used to do the validations after the environment was set up. The documentation for YOLOv5 gives pretrained checkpoints given in the above figure based on validation against MS COCO val2017 [http://cocodataset.org/] dataset. Therefore, the initial runs were executed to validate the environment with MS COCO dataset. Since YOLOv5 has different models namely, YOLOv5x, YOLOv5n6, YOLOv5m6, etc each of those models was validated with the MS COCO Dataset. YOLO documentation gives results by training each model to 300 epochs.

Since our intent is to predict threats and hazards during driving, we have categorized two types of data that can be used to train YOLOv5. The first one is RDD2020 [https://www.sciencedirect.com/science/article/pii/S2352340921004170] which can be used for training YOLO on road conditions. The dataset contains 26,336 road images collected from India, Japan, and the Czech Republic with more than 31000 instances of road damage. There are four classes mainly longitudinal cracks D00, Transverse Cracks D10, Alligator Cracks D20, and Potholes.

The directory structure for the data is given in the following diagram.



Source: [https://www.sciencedirect.com/science/article/pii/S2352340921004170]



Sample images for road damage categories considered in the data. a. Longitudinal Crack (D00) b. Transverse Crack (D10) c. Alligator Crack(D20) d. Pothole(D40).

The Second category dataset is a small one called LISA Dataset [https://makeml.app/datasets/road-signs]. This dataset contains four image classes.

1. Traffic Light
2. Stop
3. Speed Limit
4. Crosswalk

It contains only 877 images and is a very small one but suits the purpose of this research.

Both the datasets had to be converted into a YOLO-supported format to ensure the training and detection process. The online tool named Roboflow was used to annotate and generate augmented images to train both categories, Road Surface, and Sign Board warnings. The generated annotated dataset contains more than 12k images which is comparitively very less quantity but sufficient for the purpose of this research.

**Setup the Environment**

The environment consists of:

1. Server hosted by Paperspace.com datacentre in California [8Core CPU, 30 GB Memory, 8 GB GPU, Storage 100GB]
2. PyTorch
3. Jupyter Notebook
4. Wandb.ai [For tracking the metrics during training and testing]

**YOLO Architecture and Working**

Yolov5 architecture consists of three parts:

1. Backbone: CSPDarknet
2. Neck: PANet
3. Head: Yolo Layer

The data is initially fed into the CSPDarknet for feature extraction. The extracted features are then fed into PANet for feature fusion. The last stage Yolo layer detects yields the results (class, score, location, size)

Diagram

Description automatically generated

[https://www.researchgate.net/figure/The-network-architecture-of-Yolov5-It-consists-of-three-parts-1-Backbone-CSPDarknet\_fig1\_349299852]

**Hyper Parameter Evolution of YOLOv5**

**Hyper parameter evolution is the method of parameter optimization by using Genetic algorithm for optimization.** Various aspects of training are controlled by hyperparameters in machine learning and determining their ideal values can be difficult. Grid searches and other conventional techniques can easily become unworkable because of 1) the high dimensional search space 2) unknown relationships across the dimensions, 2) costly nature of assessing the fitness for each point. GA is a good choice for hyperparameter searches due to the above-mentioned concerns.

**Initialize Hyperparameters**

The hyperparameters of YOLOv5 are defined in a yaml file in /data directory. There are around 30 such parameters which can be tuned to improve the performance of the neural network. The better the initial guesses of the parameters are, the more better final values will be. Therefore, it is very crucial that the parameters are properly set before evolving. However, the evolving process also could be started with the optimized value obtained from COCO training from scratch.

**Define Fitness**

### **YOLOv5 has defined the fitness function as a combination of the following:**

* mAP@0.5 contributes 10% of the weight
* mAP@0.5:0.95 contributes the remaining 90%

It can be adjusted as per requirement as it is the value which is maximized during the training.

**Evolve**

Parameters are evolved by starting from a base scenario and then fine tuning is done . For example to fine tune on COCO128 dataset for 10 epochs using pretrained YOLOv5s, the command is as follows:

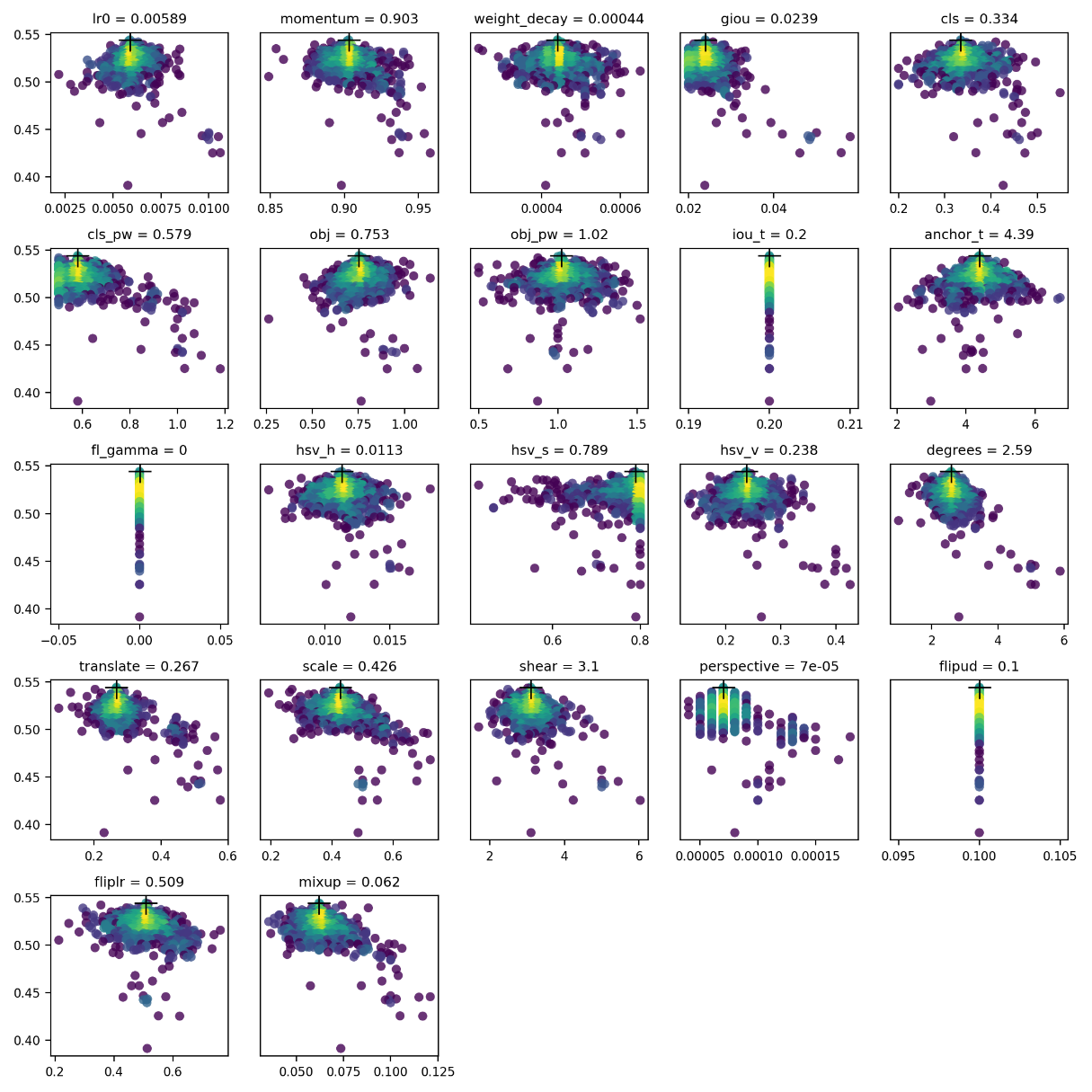
python train.py --epochs 10 --data coco128.yaml --weights yolov5s.pt –cache --evolve

By default, the number of iterations is 300. A minimum of 300 generations is recommended as it is required for best results. Evolution is an expensive and time taking process requiring a lot of GPU hours for execution.

|  |
| --- |
| # Hyperparameter Evolution Results  # Generations: 1000  # P R mAP.5 mAP.5:.95 box obj cls  # Metrics: 0.4761 0.79 0.763 0.4951 0.01926 0.03286 0.003559  lr0: 0.01 # initial learning rate (SGD=1E-2, Adam=1E-3)  lrf: 0.2 # final OneCycleLR learning rate (lr0 \* lrf)  momentum: 0.937 # SGD momentum/Adam beta1  weight\_decay: 0.0005 # optimizer weight decay 5e-4  warmup\_epochs: 3.0 # warmup epochs (fractions ok)  warmup\_momentum: 0.8 # warmup initial momentum  warmup\_bias\_lr: 0.1 # warmup initial bias lr  box: 0.05 # box loss gain  cls: 0.5 # cls loss gain  cls\_pw: 1.0 # cls BCELoss positive\_weight  obj: 1.0 # obj loss gain (scale with pixels)  obj\_pw: 1.0 # obj BCELoss positive\_weight  iou\_t: 0.20 # IoU training threshold  anchor\_t: 4.0 # anchor-multiple threshold  anchors: 0 # anchors per output grid (0 to ignore)  fl\_gamma: 0.0 # focal loss gamma (efficientDet default gamma=1.5)  hsv\_h: 0.015 # image HSV-Hue augmentation (fraction)  hsv\_s: 0.7 # image HSV-Saturation augmentation (fraction)  hsv\_v: 0.4 # image HSV-Value augmentation (fraction)  degrees: 0.0 # image rotation (+/- deg)  translate: 0.1 # image translation (+/- fraction)  scale: 0.5 # image scale (+/- gain)  shear: 0.0 # image shear (+/- deg)  perspective: 0.0 # image perspective (+/- fraction), range 0-0.001  flipud: 0.0 # image flip up-down (probability)  fliplr: 0.5 # image flip left-right (probability)  mosaic: 1.0 # image mosaic (probability)  mixup: 0.0 # image mixup (probability) |

### **Visualize**

One plot is stored for each hyperparameter in the results file, yolov5/evolve.png. The x axis shows values, and the y axis shows fitness. Yellow denotes concentrations that are higher. A parameter has been disabled and is not changing when there are vertical lines next to it. This can be used to fix parameters and stop them from evolving and is user selectable in the meta dictionary in train.py.



**Prune and Sparse YOLOv5**

Modification of weights by reducing the weights parameters, that does not influence on classification of classes is known as "pruning." Yolov5 employs a pruning strategy that "converts to zeros randomly in some proportion of weights parameters, nn.conv2d layers whose weights are virtually zero". Yolov5 must first be trained on the custom data, after which it must validate your trained model using validation data before invoking the prune function to allow the trained model to be pruned. Custom-trained weight values can be specified in best.pt file. Then the following code must be specified in val.py file.

from utils.torch\_utils import prune  
prune(model,0.3)



Once, the code is pasted into the val.py file then the following code can be executed to prune the model.

python3 val.py --data "coco.yaml or your custom data.yaml" --weights " custom file.pt"

**Literature review**

During this research, we will be doing a comparison study of the following Computer vision CNN models based on several published research papers :

1. Vision Transformer
2. OpenAI Clip
3. ResNet
4. EfficientNet
5. EfficientDET
6. Mask RCNN
7. Faster RCNN
8. Detectron2
9. Unet
10. Deeplab

**Vision Transformer vs ResNet**

The Vision Transformer or ViT is a machine learning model for image classification. [https://arxiv.org/abs/2010.11929v2]. Transformer architecture can be considered the de facto standard for natural language processing since it was proposed by Vaswani et al. (2017). Inspired by the success of Transformer Architecture for NLP, Alexey et al. (2021) propose Transformers for Image Recognition at Scale.

The approach mentioned is to directly apply Transformer architecture with minimal changes to train it with supervision. The dataset chosen was ImageNet and it yielded moderate accuracy less than ResNets of similar size. The advantage of CNN noted in comparison with the above was the inductive biases such as translation equivariance and locality. However, when trained on large dataset (14M-300M images) it yields high accuracy from 77 to 94.5% because inductive bias becomes insignificant. The larger models are comparable with the state-of-the-art CNN as claimed by the authors. Also, the study that has been conducted was using generic image datasets not optimized for any specific purpose.

**EfficientDet, Yolov5 and EfficientNET**

Renjie Xu et al. (2021) describes a study conducted to detect forest fires, an ensemble learning using Yolov5, EfficientDET and EfficientNET. Contemporary studies involving RCNN, and SSD are cited by the authors. Characteristics and requirements of forest fire detection and road hazard detection are of different in nature. However, the study performs a critical analysis of different neural networks and enumerates the pros and cons of each. In the case of road hazard detection, it requires only to identify objects fast enough and the target image dataset is of limited dispersion. Hence Yolov5 will be the necessary and sufficient neural network for our case.

The authors have selected Yolov5 as it is a real-time object detector. It has cross stage partial network (CSPNet) built into Darknet making CSP Darknet. It solves the problem of repeated gradient information. It thus captures gradient changes into feature map. This will in effect improve the speed by reducing FLOPS (floating-point operations per second).

Yolov5 has path aggregation network (PANet) in its neck. This incorporates a feature pyramid network (FPN) which allows for propagation of low-level features. FPN improves the location accuracy of the detected objects. The head of the Yolov5 generates 3 types of feature maps (18 x 18, 36 x 36, 72 x 72) and achieves multiscale prediction.

EfficientDET was developed by Google and has excellent performance when processing Pascal VOC and Microsoft COCO datasets. It has the capacity to learn complex features and employees a better PANet called bi-directional feature pyramid network (Bi-FPN) which enables fast feature fusion. EfficientDet was proposed by M. Tan et al. in 2020. D7x configuration of EfficientDet (AP of 55.1 on MS-COCO dataset) is new state-of-the-art average precision (AP). It introduces learnable weights which helps to identify the relevance of input features. In comparison with Yolov5’s neck PANet, Bi-FPN gives performances with less parameters and FLOPS. Also, it embeds compound scaling that uniformly scales resolution, depth, and width for prediction network as well as feature network. This helps to improve the accuracy and efficiency.

EfficientNet is also a network proposed by Google which introduced compound scaling as discussed above. This has made it outperform ResNet, DenseNet and ResNeXt in image classification task. This network is a candidate when we can trade-off between accuracy and efficiency.

CNNs depend on the translational invariance. This means that when a car is identified there could be many positions where the wheels could be. So, at low-level translational invariant features are captured and higher levels, high level feature and or a combination of low-level features.

The impressive deep learning technique of real-time convolutional neural networks (RCNN) is used for image segmentation and object detection. They've been specifically engineered to pick up the presence of an object. Selective Search is used in both RCNN and Fast-RCNN neural networks. Greediness is at the heart of Selective Search's design. Best Result is not often guaranteed by Greedy Algorithms. In addition, it must be repeated numerous times. RCNN, performs about 2000 iterations of selective search on the image. By running the CNN only once per image, Ross Girshick (the creator of RCNN) came up with the idea of sharing this computational burden between the 2,000 regions of the image. The R-CNN architecture uses a selective search process to generate a region proposal network for bounding boxes. In order to generate a feature vector map, a CNN is used to warp these region proposals into standard squares. Features from the image are used to create an output dense layer that is then fed into a classification algorithm to help identify the objects located within the region proposal network and assign them a class. Aside from predicting precision gains, the algorithm also predicts offset values that will be used to improve the region proposal. Convolutional feature maps are generated by feeding the input image to the CNN, which in turn generates them. These maps are used to identify the regions where proposals have been submitted. It is then possible to feed all of the proposed regions into the network by using a ROI pooling layer to reshape them into a fixed size. Fast-RCNN extracts all the regions first, then performs a selective search once for each of them. This has the effect of greatly reducing the complexity of time. The final bottleneck, Selective Search, is eliminated by FRCNN. As an alternative, it utilizes the Region Proposal Network (RPN). The regions are fixed in RPN as a n x n grid. It takes less time to run than a selective search. When it comes to computer vision, object detection has long been a vital part of the field. Bounding boxes are a useful tool for describing and identifying the objects in an image and their relative locations. In today's world, there are a variety of ways to accomplish this task. (2) Using a Region Proposal Network to find objects in an image and a second CNN backbone network to fine-tune the generated proposals to make predictions are two of the most common approaches: (1) Single Shot Detection (architectures such as RetinaNet, YOLOv3, and so on); and (2) (two-stage networks such as RCNN, Faster RCNN).

**ResMLP:**

The convolution Neural Network uses sum of multiplied matrices by a filter. It has a weight sharing mechanism so that it has fewer number of parameters that deep neural networks. CNNs are not the only mechanism by which we can perform computer vision tasks. The transformer architecture used for Natural Language Processing can also do image classification and has been proved on ImageNet dataset [].

Fully connected Layer based image classification was proposed by Touvron et al. (2021) which is called Residual Multilayer Perceptrons(ResMLP). It could perform well on the ImageNet-1k data. ResMLP is based on ViT architecture based on the transformers model.

**ResMLP Working:**

ResMLP divides the image into N X N patches where N is 16. Each of those patches is flattened into a vector and those are input into the ResMLP layer independently. The ResMLP will take a matrix X of size d x (N x N) where d is the vector dimension and N x N is the number of patches. The matrix will then undergo several transformations until a matrix Y of the same size as X is obtained.

Company name

Description automatically generated with low confidence

Text

Description automatically generated

GELU is the activation layer and aff is the operator that transforms the column of the input by shifting and rescaling and A B C matrix is the weight learned by the model.

Once the matrix Y is obtained, it will be averaged, and a d-dimensional vector will be derived which will be used as a feature for the linear classifier. Although Highway Network introduced gated shortcut connections and the solution space contains ResNet, ResNet performs better in comparison.

Diagram

Description automatically generated

**ResNet** [https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035]

Universal approximation theorem states that a single layer feedforward network with enough capacity is sufficient to represent any function. The two problems associated with this approach is that the layer can become huge, and overfitting can happen. Inorder to solve this problem , the number of layers in the neural network is increased. Increasing the depth by just stacking up the layers doesn’t give required results. Deep Networks inherently pose a threat of vanishing gradient problem becoming infinitively small as it is back propagated to the earlier layers. Due to this as the network grows deeper and deeper the performance degrades.

ResNet proposes a solution called “identity shortcut connection”. It skips one or more layers as follows

[https://www.cv-foundation.org/openaccess/content\_cvpr\_2016/papers/He\_Deep\_Residual\_Learning\_CVPR\_2016\_paper.pdf].

Diagram

Description automatically generated

The shortcut connections turn the network into residual network. The following equation can be used when input and output are the same dimensions.

Text

Description automatically generated with medium confidence

Where x and y are the input and output vectors of the layers. The function n F(x, {Wi}) represents the residual mapping. We have two approaches when dimension increases.

1. The shortcut will perform identity mapping with extra zeros padded for increasing dimensions. This doesn’t require additional parameters.
2. The projection shortcut in the following equation is used to match dimension(1x1 convolutions)

A picture containing text, watch, gauge

Description automatically generated

Where Ws is the linear projection by the shortcut connections to match the dimensions

For both above-mentioned approaches, when the shortcuts go across feature maps of two sizes, they are performed with a stride of two. The comparison table for plain network vs ResNet is shown below:

cmdTable

Description automatically generated

The findings from the comparison performed by Kaiming He et al. (2016) as per the table above are the following :

1. 34-layer ResNet is better than 18-layer ResNet by 2.8% and also has lower training errors and thus the degradation problem is well addressed, and accuracy gain is obtained from increased depths.
2. Compared to plain network, the ResNet reduced the error by 3.5% which justifies the effectiveness of residual learning on extremely deep systems.
3. 18-layer ResNet converges faster and thus eases optimization.

**YOLOv5**

YOLO in comparison to the above mentioned comes with many versions as YOLO, YOLOv2, YOLOv3 and YOLOv4, YOLOv5, YOLOv6 and YOLOv7 so far. The mechanism and architecture are different. It is a single pass which is defined as ‘You Only Look Once’. During training the input images along with ground truth (segments and labels) is fed into it. Outputs are also bounding boxes which is faster to train and predict. This is due to the tradeoff with the accuracy compared to Faster RCNN.

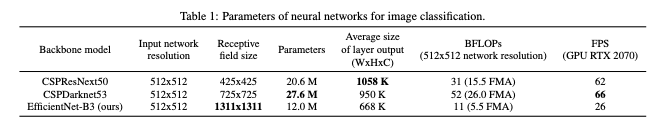
**Architectural details**

The Models backbone extracts relevant features from the image threat is fed into it. YOLOv5 makes use of CSP- Cross stage Partial Networks as its backbone to extract feature rich information. It has excellent results with respect to processing time compared to other deep networks.

The YOLOv5 object detector's backbone network is often pretrained using ImageNet classification. Pretraining refers to the network's weights being modified for the new task of object recognition even when they have already been trained to recognize important features in an image.

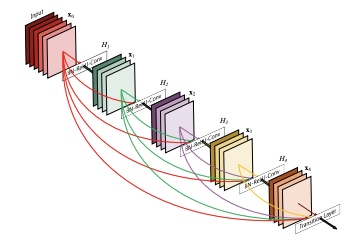
For the YOLOv5 object detector, the authors took into account the following neural backbones.

* CSPResNext50
* CSPDarknet53
* EfficientNet-B3

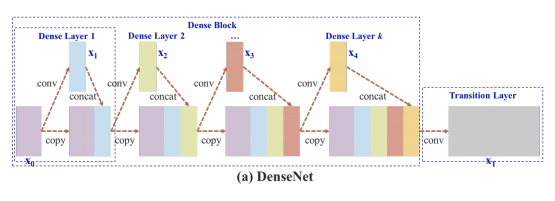


**citation**

DenseNet serves as the foundation for both the CSPResNext50 and the CSPDarknet53. It was created to connect layers in convolutional neural networks with the following goals in mind: to improve feature propagation, encourage the network to reuse features, and decrease the number of network parameters; to address the vanishing gradient problem (it is difficult to backprop loss signals through an extremely deep network).

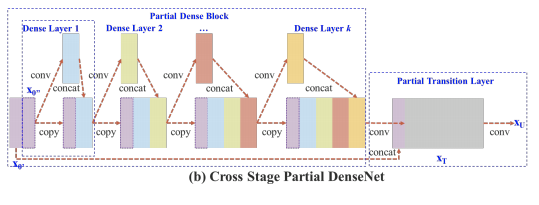


Source{“https://arxiv.org/pdf/1608.06993.pdf”}

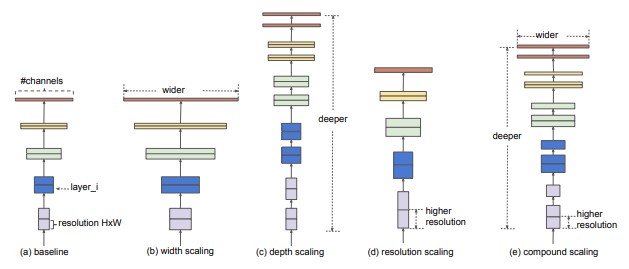


{https://arxiv.org/pdf/1608.06993.pdf}

DenseNet has been modified in CSPResNext50 and CSPDarknet53 so that the feature map of the base layer is separated by copying it, passing one copy through the dense block, and passing another directly to the next layer. By transmitting an unedited version of the feature map, the CSPResNext50 and CSPDarknet53 aim to reduce processing bottlenecks in the DenseNet and enhance learning.

{“https://arxiv.org/pdf/1911.11929v1.pd”}

Google Brain created EfficientNet particularly to research the scaling issue with convolutional neural networks. While scaling up the ConvNet, you have a variety of options, including input size, width scaling, depth scaling, and scaling all of the aforementioned options. According to the EfficientNet study, all of these have an optimal value, which they locate through search.



<https://arxiv.org/pdf/1905.11946.pdf>

EfficientNet has better performance compared to other similar networks, however, the authors of YOLO decided to try out other networks as well to ensure that which works best in the case of object detection. After a lot of experimentation, evaluation and comparison of results, finally it was decided that YOLOv5 will have **CSPDarknet53** as its backbone.

A picture containing text, receipt

Description automatically generated

Text

Description automatically generated

We can see the difference of YOLOv5 backbone with Darknet-53. YOLOv5 backbone and head have slowly evolved over many releases and contain C3 modules which are reduced CSP modules. The Focus layer introduced in the beginningappears to be equivalent to a simple 2d-convolutional layer without the need for the space-to-depth operation. For example, a Focus layer with kernel size 3 can be expressed as a Conv layer with kernel size 6 and stride 2.

The primary purpose of Model Neck is to produce feature pyramids. Pyramids of features enable models to scale objects successfully in general. The ability to recognize the same thing in various sizes and scales is helpful. Models that use feature pyramids perform well on unobserved data. Other models, such as FPN, BiFPN, and PANet, employ various other feature pyramid methodologies. <https://arxiv.org/abs/1708.02002>, <https://arxiv.org/abs/1911.09070>, <https://arxiv.org/abs/1803.01534>

PANet architecture is used for generating feature pyramids as neck in YOLOv5. <https://arxiv.org/abs/1803.01534>

[Understanding Feature Pyramid Networks for object detection (FPN)](https://medium.com/@jonathan_hui/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c)

The final detecting step is generally carried out using the model Head. It's using anchor boxes on the features and produced final output vectors that included bounding boxes, objectness scores, and class probabilities. The head of the YOLOv5 model is the same as that of the earlier generations, V3 and V4.

**Activation Function**

Any deep neural network's selection of activation functions is extremely important. Many activation functions, such Leaky ReLU, mish, and swish, have recently been introduced. The Leaky ReLU and Sigmoid activation function was chosen by the authors of YOLO v5.

Leaky ReLU, mish swish

In YOLO v5, the final detection layer uses the sigmoid activation function whereas the middle/hidden layers use the Leaky ReLU activation function.

**Optimization Function**

There are two options given by YOLOv5 for optimization.

* Stochastic Gradient Descent https://arxiv.org/pdf/1609.04747.pdf
* Adam https://arxiv.org/abs/1412.6980

By default, SGD is used as optimization function. However, it can be changed to Adam by using the command –adam as the commandline argument.

**Cost Function or Loss Function**

The Loss function is evaluated as a compound loss calculated using the objectness score, bounding box regression score and class probability score. Also, another choice of using Focal Loss function is given.

**Hyperparameters in YOLOv5**

**The kind of parameters whose values govern a deep learning model's entire learning process are called hyperparameters. While not a part of a deep learning model, it also impacts the values of model parameters that a learning algorithm ultimately learns.**

**About thirty (30) different hyperparameters are employed in YOLO-v5 depending on the training environment. We can optimize the results if we wish to get better ones. By optimizing the parameters competitive results can be obtained based on the target of optimization. default values from training on the COCO dataset can be used if optimization is not a major concern.**

**Important Hyper parameters** explored during the research are given below:

* **Learning-rate start (lr0):** It defines the step size for each iteration which means how fast the network learns from training.
* **Learning-rate end (lr1):** It checks if the current learning rate is greater than or equal to the value of this parameter then it ends training.
* **Momentum: This parameter is used for the Stochastic Gradient Descent optimization. This parameter helps to replace the gradient with an aggregate of gradient.**
* **Mosaic: This parameter helps to create a new image from multiple input images and thus the new image which has been created is used for training. This parameter is useful in the case of data augmentation and optimal feature techniques.**
* **Degree: This parameter rotates input images by certain angles and uses it for training so that the accuracy is improved.**
* **Scaling: The images are resized for optimization as well as matching it with the grid size.**
* **Flipud: This parameter helps to flip the images from the input dataset randomly up and down so as to get better results. Again, this also falls under data augmentation technique.**
* **weight-decay: This is the penalty term impacting the cost function during back propagation which helps to shrink/compress the weight.**

**Although there are many other hyper parameters that can be tuned for better results like warmup epochs , warmup momentum this research focuses mainly on the above said parameters alone.**

**Different Types of YOLOv5**

Diagram

Description automatically generated with low confidence

### **YOLOv5s**

YOLOv5s has 224 layers and 7.2 million trainable parameters and is as fast as 2 milliseconds (FLOPs or floating-point operations around 17 billion in number) with an mAP of 36.7. The predictions are saved in ‘runs/exp’ directory.

### **YOLOv5m**

YOLOv5m has 308 layers, 21 million parameters, a mean average precision of 44.5, and an average speed of inference of 2.7ms(FLOPs value at 51.3 billion).

### **YOLOv5l**

The YOLOv5l version has 47 million parameters and 392 layers. It recorded an mAP of 48.2 percent with an inference speed of 3.8ms on average(since FLOPs value is 115.4 billion).

### **YOLOv5x**

This is the biggest model referred to as the YOLOv5x and it has 476 layers and 87 million parameters along with a FLOPs value of 218.6 Billion in value. It recorded an inference speed of 6.1ms and an mAP of 50.4 percent.

**Glossary**

1. ANN artificial neural network.
2. ASO Automatic Structure Optimization.
3. CMO Confusion Matrix Ordering.
4. CNN Convolutional Neural Network.,
5. ELU Exponential Linear Unit.
6. ES early stopping.
7. FC Fully Connected.
8. FLOP floating point operation.
9. GA genetic algorithm.
10. GAN Generative Adverserial Network
11. GPU graphics processing unit.
12. HSV hue, saturation, value.
13. LCN Local Contrast Normalization.
14. LDA linear discriminant analysis.
15. LReLU leaky rectified linear unit.
16. MLP multilayer perceptron.
17. NAG Nesterov Accellerated Momentum.
18. NEAT NeuroEvolution of Augmenting Topologies.
19. OBD Optimal Brain Damage.
20. PCA principal component analysis.
21. PReLU parametrized rectified linear unit.
22. ReLU rectified linear unit.
23. SGD stochastic gradient descent.
24. ZCA Zero Components Analysis.

**List of figures and tables:**

**References:**