# Introduction

All over the world, construction is one of the most hazardous industries due to its unique nature. Construction safety is always a matter of concern for both practitioners and researchers. The number of fatal injury accidents reported every year is very high in construction industry. Even after sufficient risk assessments and the implementation of adequate controls in the work environment, workers could still be subject to health and safety risks from hazards, which is why Personal Protective Equipment (PPE) is important.

Every week, a huge amount of pictures and videos are being captured on projects around the world, with many of them being used to track progress, resolve claims, identify quality issues, etc. Safety is of highest priority in construction, as per the OSHA report approximately 1 in 10 workers injured on the job site each year. It is not practically possible for dedicated safety experts to visit every site. But what if there was a way for these computer vision engines to review images and predict the level of safety in the site.

This paper gives a model to detect people wearing PPE namely hardhat and safety jacket, save image instances and generate date wise report of those who are not wearing both. The model is created by applying transfer learning to YOLOv3 network.

# Features of YOLOv3

Transfer learning is applied to YOLOv3 to create the model; hence the basic network features remains the same.

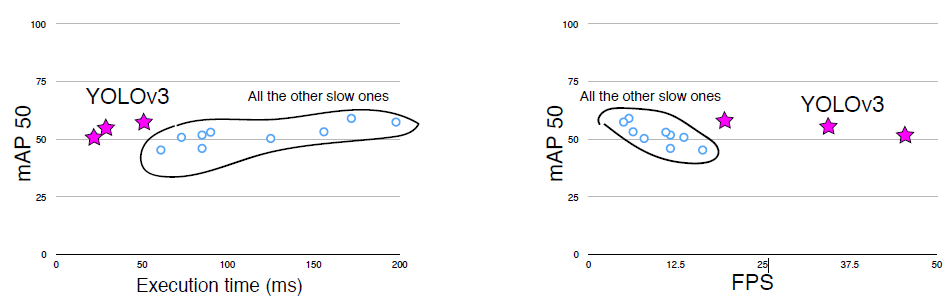
YOLOv3 is a recent object classifier network, released on April 2018. YOLOv3 has a Frames Per Second (FPS) rate and Execution time of 35 and 29 milliseconds respectively on COCO dataset as shown in figure 1, mAP-50 is 55.3 for the same [1]. Another important feature is that YOLOv3 makes predictions at three different scales [1]. The predictions for the third scale benefit from all the prior computations as well as fine-grained features from early on in the network [1]. YOLOv3 performs very well in the fast detector category when speed is important [2]. When accuracy vs speed is plotted on the AP50 metric, YOLOv3 has significant benefits over other detection systems. Namely, faster and better [1]. For the same reason, this network is used, as we are dealing with real time detection of people wearing PPE on construction site.

Figure 1: Comparison of Execution time and FPS rating between YOLOv3 and other networks [1]

# Experimental Method

## Network Modifications

YOLOv3 model written in darknet is made into a keras model with help of the convert code given in [3]. Basically the codes from [3] is used for training. The number of classes for the used model is 80. This was changed into four and corresponding changes to filter size is made to layers before all the three yolo layers. The number of filter is updated using the equation:

Here 5 includes four bounding box offsets, namely width, height and x & y coordinates of the center of bounding box and the remaining one is for the objectness. The three represents the number of anchor boxes at each scale of prediction. Looking closely, we can see that there are 9 anchors used in YOLOv3, but as mask is used, each yolo layer predicts only 3 bounding boxes, hence three. The number of classes that we have is four, which are SAFE, NoHardhat, NoJacket & NOTSAFE. Hence the number of filters comes up to be .

The training code in [3] has only train set, it randomizes the train set and uses the last 10 percentages of images as the validation set. As I had images from different sources and I wanted to be sure that validation set contains images from all the sources, I included a validation set importing as text file into the code and used the set for validation purpose and the train set for training alone. Some minor changes in the training code like saving the tensorboard checkpoints in date wise folders is also added. A for loop is added in the input of prediction code to process all images in a single folder and give output by replacing the same file, the folder name is taken as the input. After prediction a beeper and an image saving is added if the predicted class is NOTSAFE. The image saving part is such that the image is saved into date wise folders inside a folder named NOTSAFE in the root folder. The images are named as NOTSAFE along with date and time of prediction. A new code is written to generate date wise reports in DOCX format having images from a particular date. This date is the input to the code, the reports is saved in Reports folder present in root folder and named as the input date. The same date is also the prediction date of the images in the folder. Each image has heading as the filename itself.

## Data set preparation

Data is the core part of training any CNN network, hence data set preparation becomes the most important part of this experiment.

* + 1. Data Collection & Augmentation: Data collection is done using two methods. First method is manual collection, which is visiting construction sites and taking video footages. Images are captured at an interval of 5 seconds from these video files using a python code. Total number of images collected by this method is 260. The second method is image scraping, here google\_images\_download library is used for automated search and download of google image results. Most of the images are collected by this method only. Approximately 2000 images are collected by this way, but some are duplicate and blur. So after avoiding the non-satisfactory images, duplicated and blurred one’s, the collected data set comes up to be 1760 images. An important thing to check while making data set for training is that we always need a balance data set. By balanced data, here it means that all the classes are well balanced, like let’s assume there are two classes, class A & Class B and a total of 100 data points or images of class A & class B. To be a balanced data set the split between both classes should be around 50:50, i.e. 50 images of class A and 50 of class B. To balance the data set obtained augmentation was done to selected classes. The augmentations done are flipping image horizontally, rotating 30 degrees right and 30 degrees left. Hence the final data set comes up to be of 2509 images and 4132 data points.
    2. Image labelling & Train-Validation-Test set separation: Image is labeled using image annotation tool, labelImg-master available in []. The bounding boxes are drawn manually and label is given based on the object as shown in figure 1. This annotation data is saved as XML files.

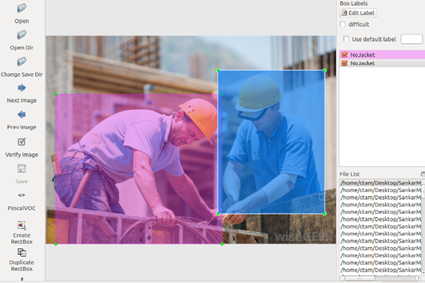
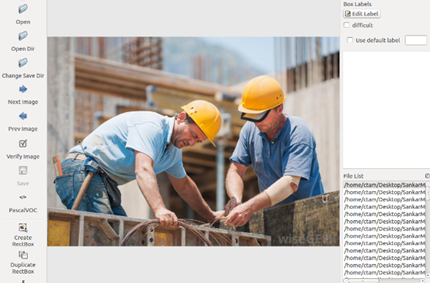


Figure 2: Image labelling using labelImg-master

After labelling all the images the data set is split into train, validation and test set in the ratio 88:10:2. The number of images in each set and data point wise split is as shown in table 1. Then data from each XML file is combined into a txt file with single image’s data in a single line, in the format [imageid bounding\_box\_values(xmin,ymin,xmax,ymax),class].A small space is left after each bounding box data in the image. Classes are coded as 0 for NOTSAFE, 1 for NoHardhat, 2 for NoJacket and 3 for SAFE. An example is *train/ngimage490.jpg 60,13,147,157,0 224,123,339,262,2.* Train and validation set image’s bounding box data is ordered in the above format in separate txt files. These two txt files alomg with the images in corresponding folders are the input to training.

Table 1: Image and data point distribution for different sets

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | | **Train Set** | **Validation Set** | **Test Set** |
| **Number of Images** | | 2217 | 238 | 54 |
| Number of Data Points For Each Class | NOTSAFE | 882 | 96 | 19 |
| NoHardhat | 779 | 72 | 17 |
| NoJacket | 970 | 112 | 23 |
| SAFE | 1040 | 115 | 27 |

## Training

Let’s take a look at the environment for training first. The system is a basic 8 \* Intel Core i7-4790@3.60GHz CPU with just an Intel HD Graphics GPU. The system has a RAM of 8GB 1600MHz. Comparing to the TitanX GPU machine which is used for training the YOLOv3 network, we can say that the computational resources available is very primitive. Hence bad allocation error, system out of memory error comes usually while training the network. To avoid this clear session from keras is used and the epoch numbers and number of untrainable layers are also controlled. The training has been done in 3 stages. Each stage has 2 parts. In first part, only last 10 layers of the model is trainable, others are kept untrainable. Then after a particular epoch number all the layers becomes trainable which is the second part.

The result model from previous stage is used as the base model for the later stages. The initial model for first stage training is prepared as mentioned in [] by combining the cfg and weights file of YOLOv3 downloaded from []. The model is an h5 format with a total of 252 layers. The model includes 23 adding layers, 72 batch normalization layers, 75 2D convolutional layers, 72 leaky ReLU layers, five 2D zero padding layers, two each concatenate and up-sampling layers and a yolo loss layer. The data after first part of training is saved as a new model in date wise folder in logs directory. While training, during second part of each stage, model is saved after each 3 epochs. The batch size for the first stage of training is 8 for first part and 4 for second part. The later stages have batch sizes of 4 and 2 for the first and second part respectively.

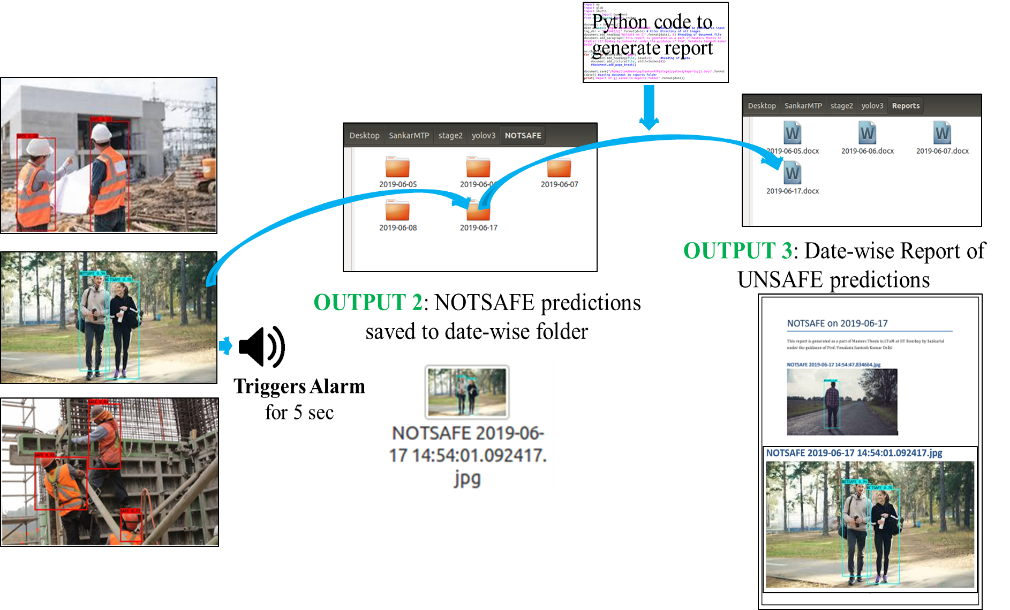
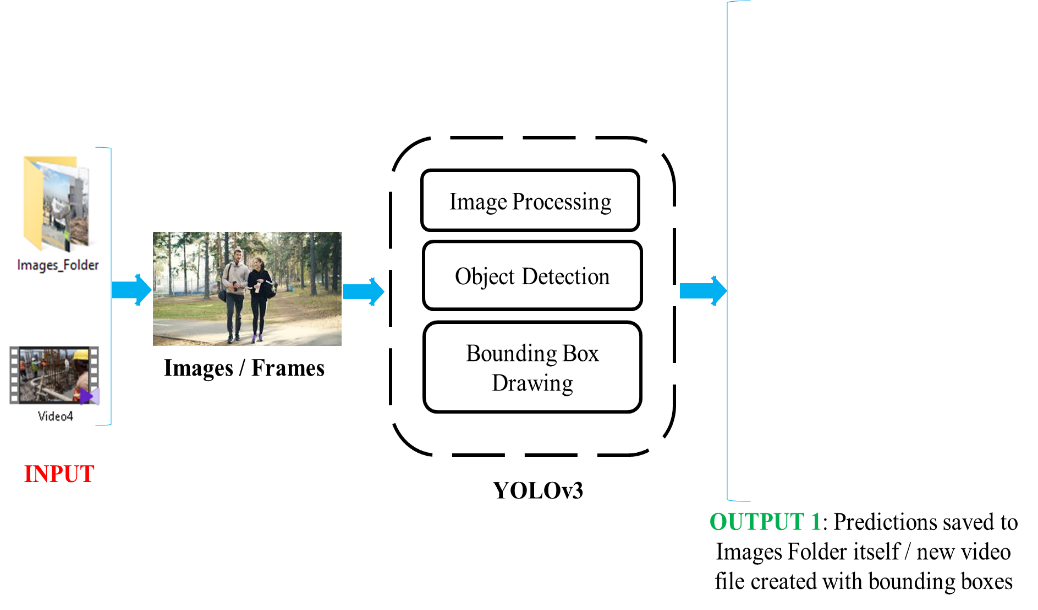
The training progress of a network can be monitored using any of the two parameters; loss and accuracy. In this model, a loss function named yolo loss as described in [] is used to monitor the training process. Tensorboard callbacks are added for visualizing the progress in yolo loss function. For better training, reducing the learning rate if a change in validation loss is not less than 0.1, considering the last three epochs is incorporated in the code. To avoid overfitting, early stopping if a change in validation loss is nil for the last ten epochs is used.

If the model loss is converging to higher values even after a large number of epochs, then it means that there is issue with your data. So if this case comes up, make sure the data is balanced and without any noise. Noise here refers to blurred or distorted images. This was the case, when I was using a different data set, so I balanced the data and removed noise to get the data set which is used for training. The first stage of training took approximately 10 days and stopped at 509 epochs after converging loss from a value as high as 6486 to 21.88. The second stage training took approximately 3 days and last stage 2 days. The second and third stage stopped after 92 and 70 epochs respectively. The final loss was reduced to 12.06 after all the 3 stages of training as shown in figure XX. All this data of training is tabulated in table 2.

Table 2: Training data

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Stage 1** | **Stage 2** | **Stage 3** |
| **No. of epochs**  (before / after making all layers trainable) | 500 / 500 | 50 / 50 | 50 / 50 |
| **Batch Size**  (before / after making all layers trainable) | 8 / 4 | 4 / 2 | 4 / 2 |
| **Epoch after which training stopped** | 509 | 92 | 70 |
| **Time taken** | 9d 13h 55m 20s | 3d 11h 51m 45s | 2d 5h 8m 7s |
| **Initial Loss** | 6486 | 21.88 | 12.87 |
| **Final Loss** | 21.88 | 12.87 | **12.06** |





# Working Model

## Object Detection and Classification

The working model is well illustrated in the figure XX. The input to the model can be a folder containing images or a video file in mp4 format. Each image in the folder or image frame of video file is converted into an image of size (416,416) where 416 is the width and breadth of the new image. Then the new image is processed by the model, class detection and bounding box drawing is done. The boxes has label as Class and the percentage bywhich the model is sure about the detection being the labelled class. Different colour is assigned to each class. The new image with bounding box drawn is saved in the same folder as input replacing the original file. In case of video input, a new video file is created with the name we provides and gets saved in root folder.

## Report Generation

Whenever the model predicts a NOTSAFE class, it triggers a beeper for 5 seconds and that image or image frame is saved into a new directory in NOTSAFE folder with folder name as current date and filename containing date and time of prediction. These saved files can be agglomerated in the form of a report by executing Generate\_report.py python file in the root directory and giving input as the date for which we need a report. The reports get saved in Reports folder with the filename as the date provided and has all instances of NOTSAFE class for the provided date with each image heading containing date and time of prediction.

# Performance Evaluation

Accuracy and F1 score are the parameters used to comment on the model performance and calculated with help of confusion matrix. In confusion matrix, human interpretation and model predictions are counted and marked in a table. The maximum value should be along the diagonal to say that the model is predicting well. The True Positive (TP), True Negative (TN), False Positive (FP) and False Negative (FN) values of each class are counted and tabulated which is the confusion matrix. Accuracy of model in an image set is calculated as the sum of TP over Total number of objects in the set. Precision is; given a class prediction from the classifier model, how likely is to be correct and calculated as TP over the sum of TP and FP. Recall is; given a class, will the classifier model detect it and calculated as TP over the sum of TP and FN. F1 score is the harmonic mean of precision and recall and is calculated as two times product of precision and recall over their sum. The equations used for calculating precision, recall and F1 score are as shown in equations XXX. Performance on three different sets namely validation set, test set and new data set are calculated.

The confusion matrix for validation set, test set and new data set are as shown in Table XX, Table XX, table XX respectively.

Table 3: Confusion Matrix for Validation Set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **HUMAN** | | | |
| NOTSAFE | NoHardhat | NoJacket | SAFE |
| **MODEL** | NOTSAFE | 85 | 0 | 1 | 0 |
| NoHartdhat | 0 | 75 | 0 | 2 |
| NoJacket | 5 | 0 | 105 | 2 |
| SAFE | 0 | 0 | 4 | 96 |

Table 4: Confusion Matrix for Test set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **HUMAN** | | | |
| NOTSAFE | NoHardhat | NoJacket | SAFE |
| **MODEL** | NOTSAFE | 19 | 0 | 1 | 0 |
| NoHartdhat | 0 | 17 | 0 | 0 |
| NoJacket | 0 | 0 | 21 | 1 |
| SAFE | 0 | 0 | 1 | 26 |

Table 5: Confusion Matrix for New data set

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | | **HUMAN** | | | |
| NOTSAFE | NoHardhat | NoJacket | SAFE |
| **MODEL** | NOTSAFE | 14 | 0 | 0 | 0 |
| NoHartdhat | 0 | 8 | 0 | 0 |
| NoJacket | 0 | 0 | 21 | 0 |
| SAFE | 0 | 2 | 0 | 20 |