VIP – IPA

Human Detection Team

Final Report

Spring 2023

April 18, 2023

Robert Sego, Xilai Dai, Alex Weber, Patrick Li, Sun Ahn

Advisors: Professor Carla Zoltowski, Professor Edward J. Delp

Department of Electrical and Computer Engineering

Purdue University

Table of Contents

Abstract

Pedestrian detection models are often inaccurate for datasets other than the one they were trained on; they generalize poorly due to their specialized structure that often relies on constructs such as anchor settings. Additionally, their source datasets are uniform and often do not contain images of crowds or diverse enough scenarios. Intention estimation, crowd counting, and subject identification are all related problems that require the detection of humans. The purpose of this project is to create a broader human detection model capable of finding people in a variety of contexts. The end model must detect people in unusual poses, obscured partially by the scenery, distorted by fast movement, or surrounded by a crowd. To meet these constraints we implemented YOLOv3 and Faster RCNN, both general object detection neural networks, and trained them on the ETH Pedestrian dataset then tested on the WIDER 2019 Pedestrians dataset.

By using a more general architecture and a web-scraped, varied dataset, we seek to create a more robust detector that is effective on a wider range of datasets than contemporary models for pedestrian detection, such as F2Dnet, EGCL, and Pedestron. The resulting model will be an effective research tool for various tasks and can inform future designs for pedestrian detectors.

Technical Background

Neural Networks

Neural Networks are composed of node layers, containing an input layer, one or more hidden layers, and an output layer. Each node, or neuron, connects to another as shown below.

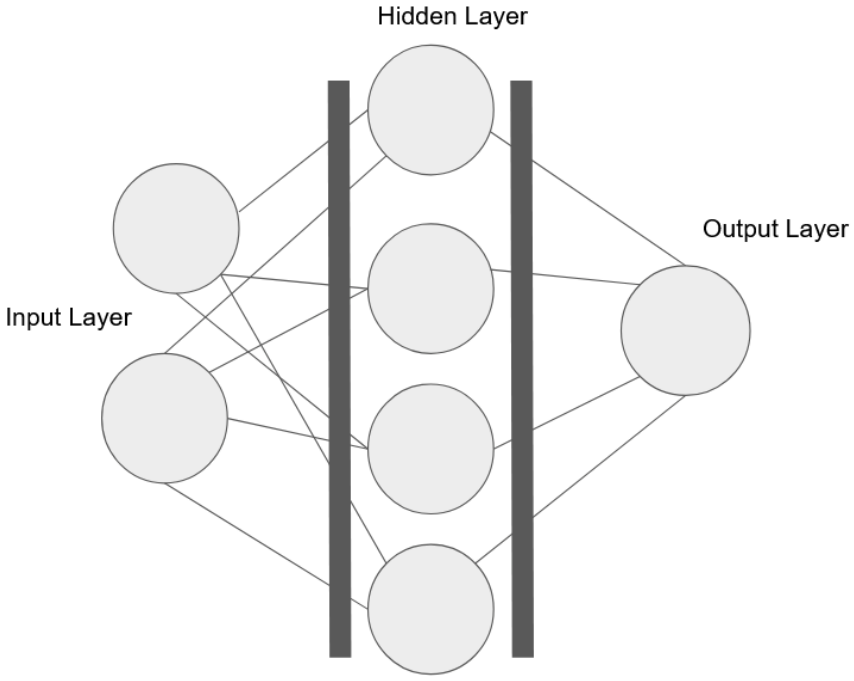


Figure 1: Neural Network

Each input node takes input data and is composed of weights and a bias. It outputs values to each of the connected nodes by multiplying the individual connection weight with the input and adding bias. Once the model structure is established, weights are initialized randomly, usually between 0 and 1. At the hidden layer node, inputs from all the connected previous layer nodes are then summed together. The output is then passed through an activation function, specified on the creation of the model. The neural network forms a large function, and using a nonlinear activation function on the output of each node can allow the model to learn a nonlinear mapping of the input to the output. This process is repeated until the output layer is reached. All useful activation functions are non-linear, a popular example being sigmoid and relu activation, both shown below:

$$\sigma(X) = \frac{1}{1+e^{-X}}$$

Figure 2: Sigmoid Activation

$$r(X) = \begin{cases} 0 & \text{if } X < 0 \\ X & \text{if } X \geq 0 \end{cases}$$

Figure 3: Relu Activation

Mathematically the entire process at each node can be represented by the following equation:

$$Y = z(\sum_{i=1}^{N} X_i w_i + b)$$

Figure 4: Node Equation

z(): Activation function

N: number of connections to node

X: Node input

b: Bias

w: Connection weight

Y: Node output

Consider this visual representation of this equation where the nodes are multiplied by the corresponding weights and summed together, producing an output.

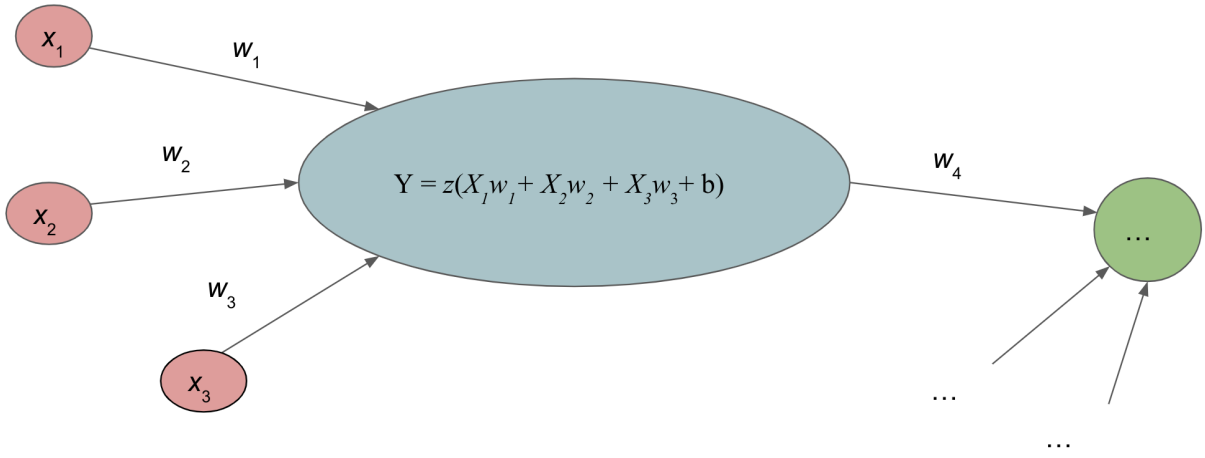


Figure 5: Node Operation Diagram

Neural networks "learn" by updating these weights using functions known as "loss functions." In training, these loss functions often represent a comparison of the output of the model with the correct output for the given input. The partial derivative of the loss relative to each output layer weight is used to determine the change in those weights. Then the partial derivative relative to the weights for each of the nodes before the output layer is calculated, and this process is repeated through the entire network back to the input layers. This is called "backpropagation."

Convolutional Neural Network

A Convolutional Neural Network (CNN) is a type of deep learning model often used for image recognition, image classification, and other computer vision applications. A CNN can be used to identify and classify objects by learning to recognize and identify different features in an image. Previous CNN's have been used for the classification of hand-written digits, animals, or clothing items. Grayscale input images can be represented as a 1-dimensional matrix of pixel values. However, color images in the red, green, and blue (RGB) color space are represented with a three-dimensional matrix where the last axis represents the RGB color values.

In an image, pixel values have patterns that relate to each other on both axes in the matrix. Typical dense models require flattened inputs, which would destroy the vertical information stored in an image. A CNN handles these dependencies by using trained finite impulse response (FIR) filters of a specified kernel size for each layer. With these filters, a CNN can transform an input image into a lower or higher channel form that is easier to process without changing or losing image features (e.g. edges, color, gradient orientation). The CNN extracts and learns important features of these input images by assigning and updating filter weights and biases using a loss function that calculates its total distance from the ground truth, which are images labeled by hand. This is followed by dense layers operating on these extracted features which are updated through a similar process to improve its final decision-making mechanism. A typical CNN uses convolutional layers followed by a pooling operation and densely connected layers before the output.

$$O[m,n] = I[m,n] * h[x,y] = \sum_{x=-\infty}^{+\infty} \sum_{y=-\infty}^{+\infty} (h[x,y])(I[m-x,n-y])$$

Figure 6: 2D Finite Impulse Response Discrete Convolution

To illustrate this operation, consider this example of the convolution of an example image input matrix with a 3x3 filter matrix.

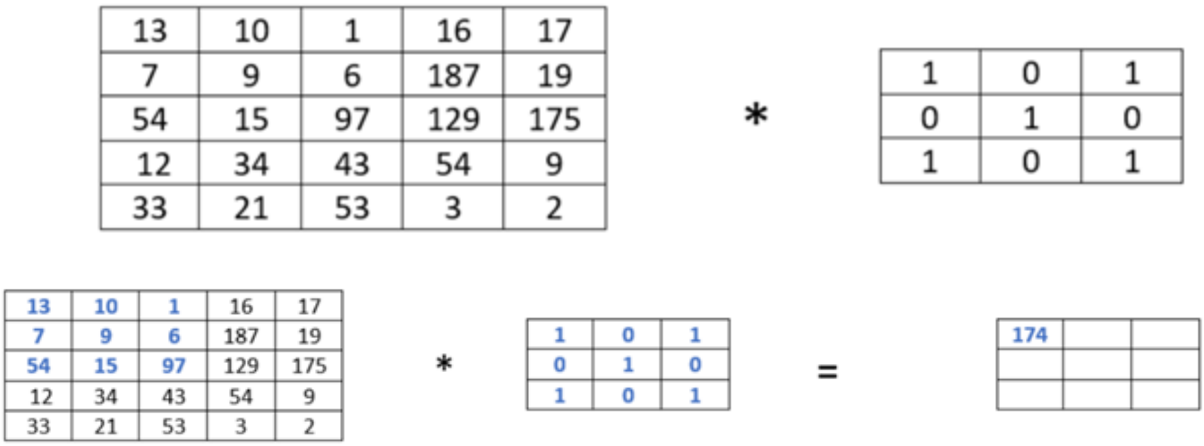


Figure 7: 2D FIR Discrete Convolution Visualization

The window of the example data is shown on the left in a 5x5 matrix. The blue numbers represent the region of the input that is being convolved with the filter, producing a summation output of 174, the dot product of the filter and the highlighted region, in the upper left. This is repeated for each available 3x3 region in the input, producing a 3x3 output. The resulting output is two pixels shorter than the original image in both the x and y direction as a result. If an equal-size output is required, the input may be padded with zeros or the nearest pixel value on the edges of the image.

Pooling layers are implemented to reduce the amount of data the model must handle in each layer. Within each windowed data segment layer, low-weighted and unimportant values are removed from being processed. This is done by decreasing the spatial size of the pixel matrices. An essential requirement for the pooling layer is to reduce the dimension of the pixel matrices without losing important information. Reducing the image size would consequently decrease the computation power and computational time required to process the data. Furthermore, cutting down the dimensionality size also contributes to suppressing noise. Some typical pooling methods that can be implemented are max pooling, average pooling, and sum pooling. This project uses max pooling and average pooling.

Consider this example of a 4x4 pixel matrix:

13	10	1	16
7	9	6	187
54	15	97	129
12	34	43	54

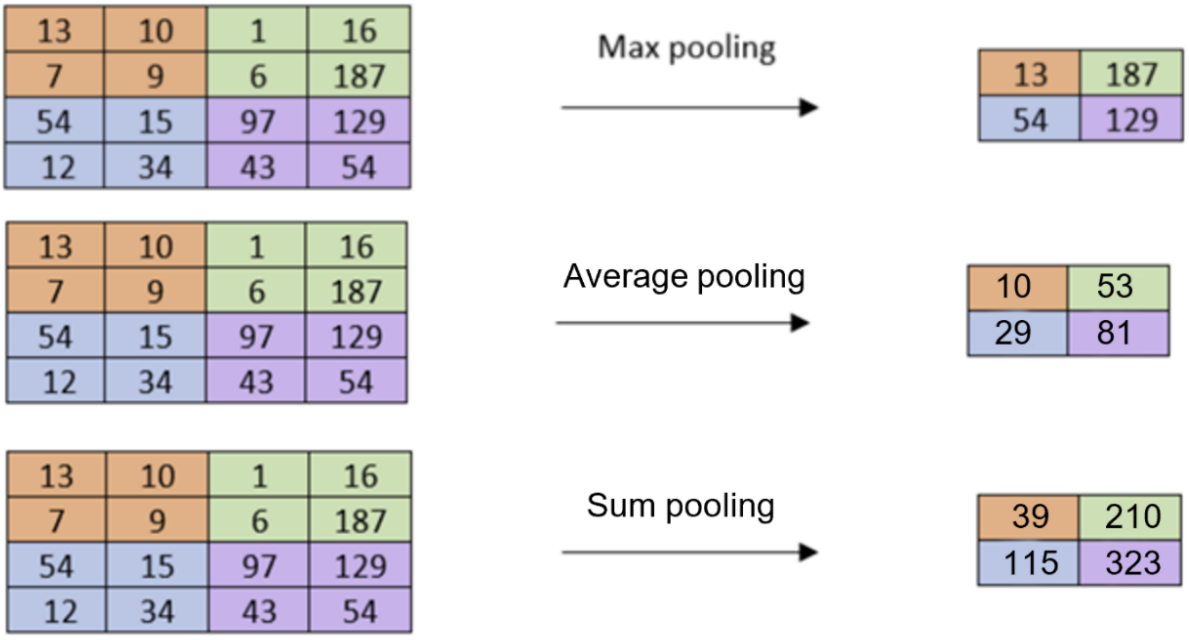


Figure 8: Pooling

Sum pooling returns the sum of all values in the portion of the pixel matrix, average pooling returns the average value of all values in that portion, and max pooling returns only the maximum value out of all values in that portion of the pixel matrix.

Residual Blocks

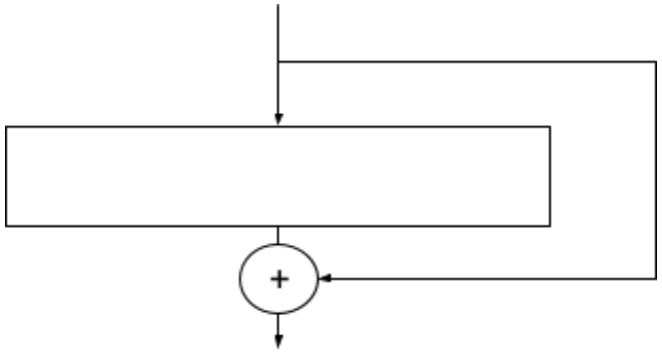


Figure 9: Skip Connection

The output of each layer in conventional neural networks is the input to the following layer. However, going layer by layer can be very time-consuming and inefficient so residual layers allow the neural network to skip some layers and reach later layers faster. Normal neural networks can be represented by the i + n since they move one layer at a time from layer i to i + n

and residual connections can be represented by the function F(x) + x where x is the input and F(x) + x is the output.

The accuracy of neural networks often improves with layer count, making them universal function approximators, but there is a limit to how many more layers may be added without decreasing accuracy. Neural networks would be able to learn any simple or complex function if they were universal function approximators, but sufficiently deep networks may not be able to learn basic functions like an identity function due to issues like vanishing gradients. When layers are added, it is observed that the accuracy eventually plateaus at a certain point and this is usually not caused due to overfitting. Therefore, it may appear that shorter networks are learning more effectively than deeper networks. However, the degradation problem is what is observed in practice. In the degradation problem, the shallower networks perform better than the deeper counterparts with a few more layers added to them. A possible solution to at least match the accuracy of the shallow sub-networks is to skip the extra layers. Residual connections can be used to skip the training of a few layers using skip connections. Different parts of networks are trained at different rates for different training data points based on how the error flows backward in the network. This is similar to training an ensemble of different models on the dataset and getting the best possible accuracy. The optimal number of layers, or residual blocks, required for a neural network depends on the complexity of the dataset.

The network is allowed to skip training for the layers that are not useful and do not add value to overall accuracy. In a way, skip connections make neural networks dynamic to tune the number of layers during training optimally. The idea of skipping connections between the layers was first introduced in Highway Networks. Highway networks had skip connections with gates that controlled how much information was passed through them, and these gates can be trained to open selectively. Residual blocks are essentially a special case of highway networks without any gates in their skip connections. Residual blocks allow memory, or information, to flow from the initial to the last layers. Despite the absence of gates in their skip connections, residual networks perform as well as any other highway network in practice.

Methods

Datasets

For our advanced detectors, we utilized the ETH [3] and WIDER 2019 [2] datasets. Both are pedestrian datasets that provide annotations to use as ground truths. This was important to us as it allowed us to focus on our code development instead of determining ground truths for our datasets.

The Wider 2019 dataset is a dataset provided by CodaLab for the WIDER Face & Person Challenge 2019. It contains images from surveillance cameras and camera-mounted vehicles driving through traffic in urban areas. Images in this dataset contain a variety of weather, time of day, and crowd conditions. Features include the variety of camera angles and uniformity of person size in the images. See below for examples from the WIDER 2019 dataset.

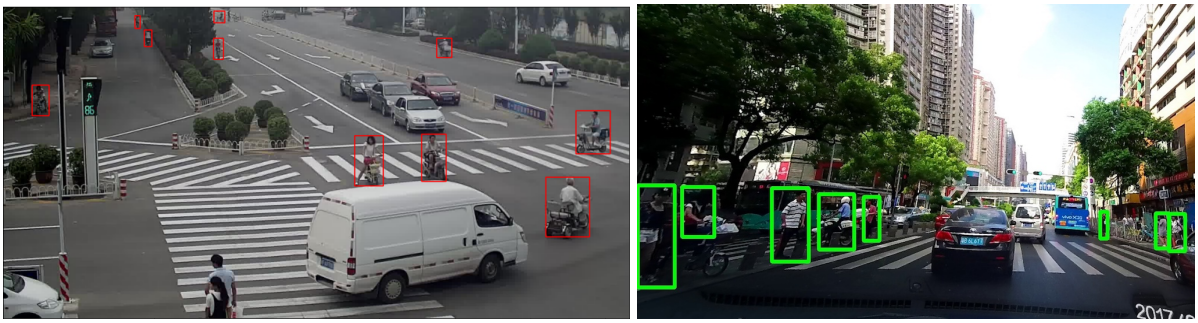


Figure 10: Wider 2019 Dataset Ground Truth Examples

The ETH dataset is a project funded in part by the Toyota Motor Corporation. It contains images of multiple 13-14 FPS with a resolution of 640 x 480 videos taken from a kart-mounted camera. These videos are taken as the kart is moved through crowds of people. Key features include the different sizes of persons in the images, with people in both the foreground and background. See below for image examples from the ETH dataset.

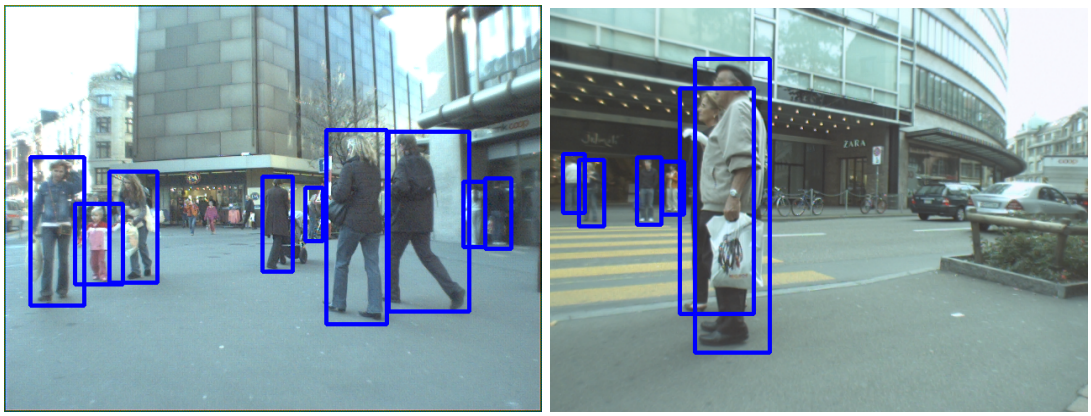


Figure 11: ETH Dataset Ground Truth Examples

Metrics

IOU

$$IOU = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

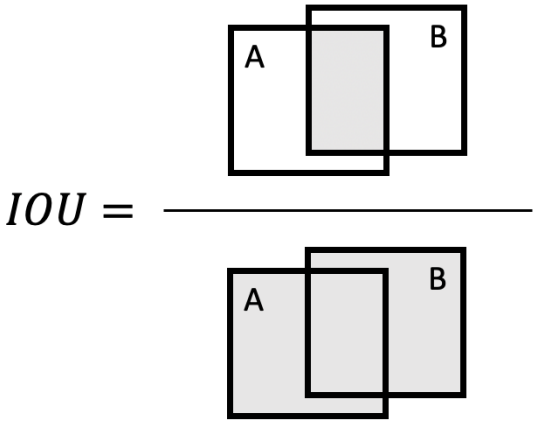


Figure 12: IOU Equation and Visualization

IOU or Intersection Over Union is used to evaluate model performance. IOU quantifies the amount of overlap between two regions, in this case output and groundtruth, into a number between zero and one. Zero indicates no overlap, while one indicates a perfect overlap. This was

used as a way to estimate if model wasmaking correct predictions based on the groundtruth values from the datasets.

The equation for IOU is shown in Figure 12. One feature of IOU is that it is continuously differentable when measuring the shared region between 2 bounding boxes. This enabled IOU to be used in the loss function for YOLOv3 [5] without causing issues with the backpropagation.

Confusion Matrix

		Actual	
		Positive	Negative
Model output	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

True Positives (TP): The model predicted a label and matches correctly as per ground truth.

True Negatives (TN): The model does not predict the label and is not a part of the ground truth.

False Positives (FP): The model predicted a label, but it is not a part of the ground truth.

False Negatives (FN): The model does not predict a label, but it is part of the ground truth.

Figure 18: Confusion Matrix

The YOLO v3 implementation uses false positives to estimate the effectiveness of the model, notably the majority of these false positives are removed using non-maximum suppression. Faster R-CNN [8] tests its performance using average precision and average recall, both of which are calculated using terms from the confusion matrix information.

Precision & Recall

$$Precision = \frac{TP}{TP + FP}$$
$$Recall = \frac{TP}{TP + FN}$$

Figure 12: Formula of Precision and Recall

Precision and recall are two important metrics used in the field of machine learning to evaluate the performance of classification models, particularly binary classification problems.

Precision measures the proportion of correctly predicted positive instances among all instances that are predicted as positive. Recall measures the proportion of correctly predicted positive instances among all instances that are actually positive.

Object Detectors

YOLO v3

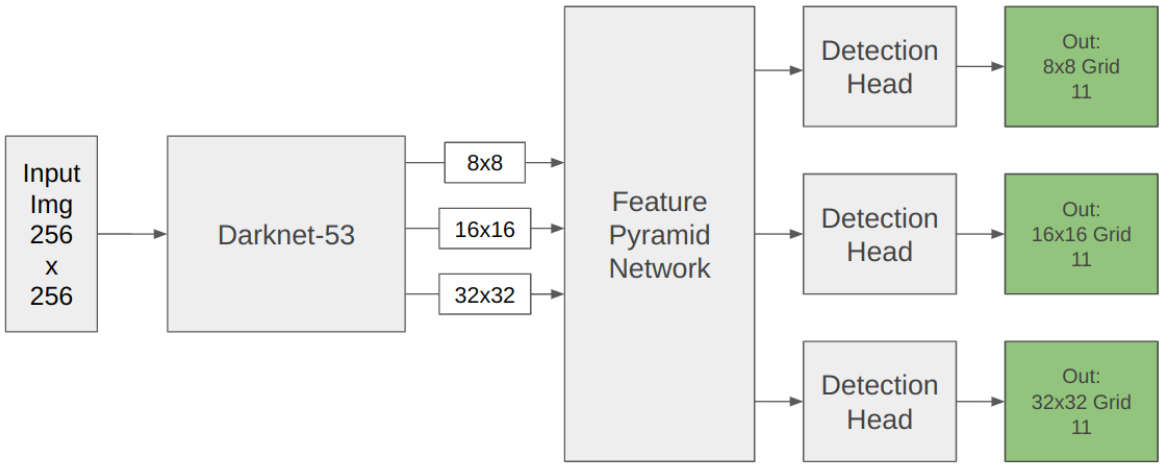


Figure 18: YOLO v3 Model Diagram

This model architecture first extracts image features using a series of convolutional layers and residual connections called Darknet-53.

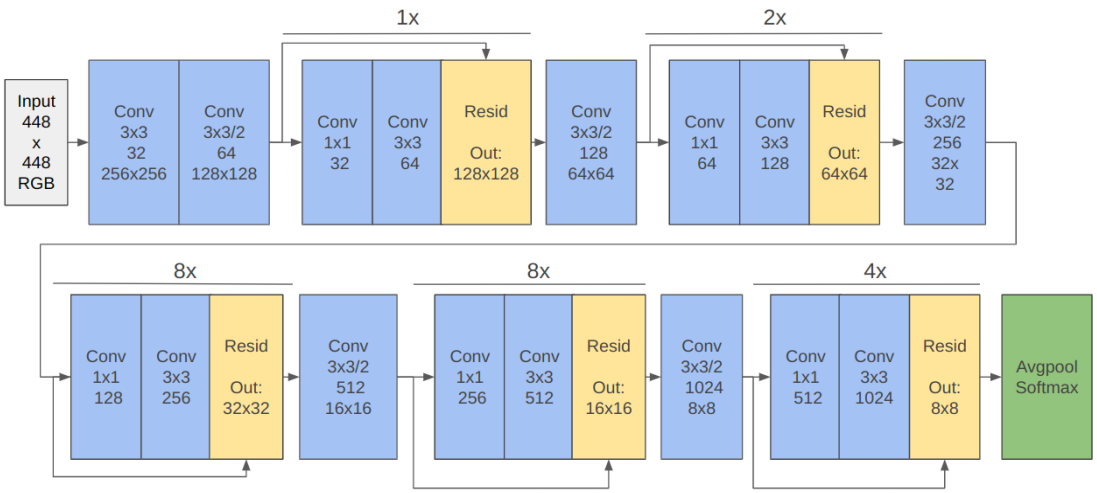


Figure 13: Darknet-53

Darknet-53 outputs three different feature maps with three different scales. One map is a 32 by 32 grid, one is a 16 by 16 grid, and the last is an 8 by 8 grid. The feature pyramid network directly uses these feature maps as inputs, as shown in the following diagram.

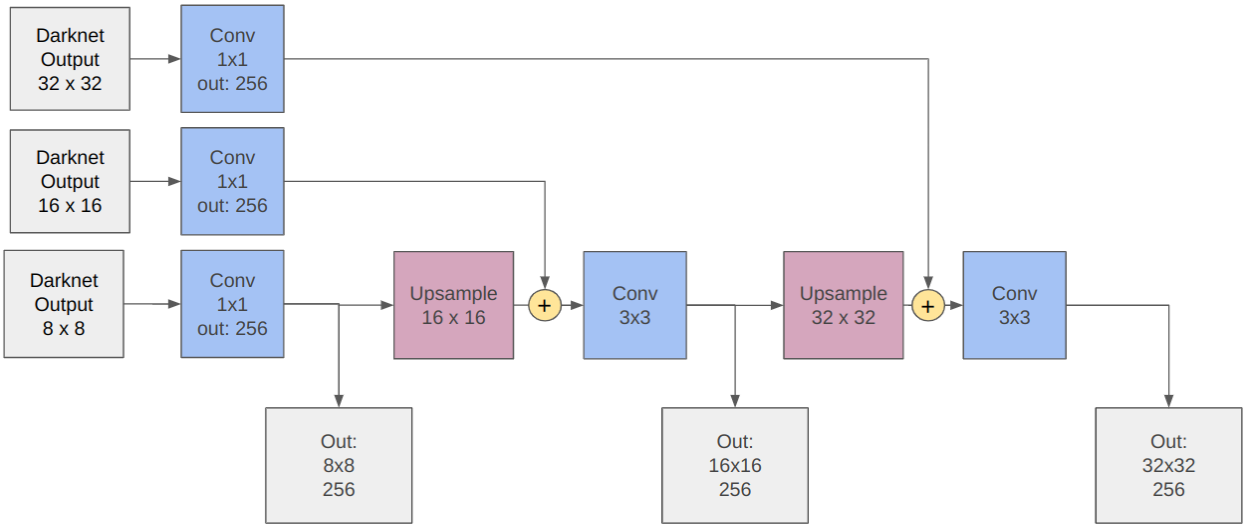


Figure 14: YOLO v3 Feature Pyramid Diagram

Specifically, the feature pyramid network utilizes upsampling and lateral connections to ensure that higher resolution feature maps do not miss features only found in the lower resolution feature maps. The three output feature maps from the feature pyramid network are then passed through various detection layers, which use the extracted features to find bounding box constraints and detection probabilities. To do this, the extracted features are passed through a

dense layer that transforms the input into a list of values that will be reshaped to a tensor of the following size:

$$S \times S \times (B \times (5 + C))$$

Figure 15: YOLOv3 Grid Cell Output Shape

S: Number of horizontal or vertical grid cells

B: Number of bounding boxes

C: Number of classes

This effectively divides up the features extracted from the original image and creates B number of bounding boxes for each grid cell.

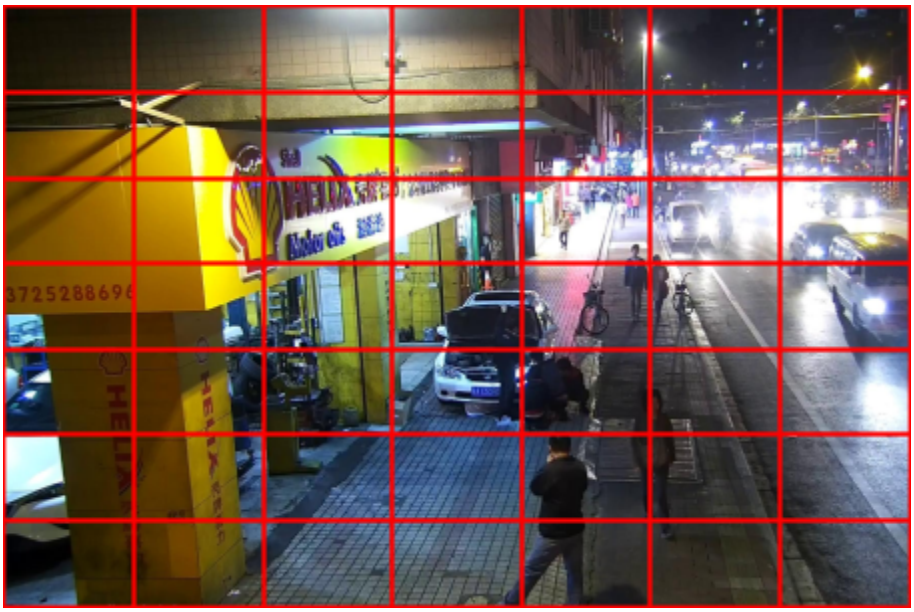


Figure 16: Grid Cell Example

If B is 2, and there is only one class (humans), then the output in each grid cell will be 12 values long.

$$t_{x1}, t_{y1}, t_{w1}, t_{h1}, C_1, p(c)_1, t_{x2}, t_{y2}, t_{w2}, t_{h2}, C_2, p(c)_2$$

Figure 22: Labeled Grid Cell Output List

C and $p(c)$ represent object confidence and class confidence, respectively. t_x, t_y, t_w, t_h represent offsets for the x-coordinate, y-coordinate, width, and height of an object from predefined anchor

boxes. The following diagram shows the relationship between the dimensions of the actual bounding box d_x, d_y, d_w, d_h and the values that the YOLOv3 model outputs.

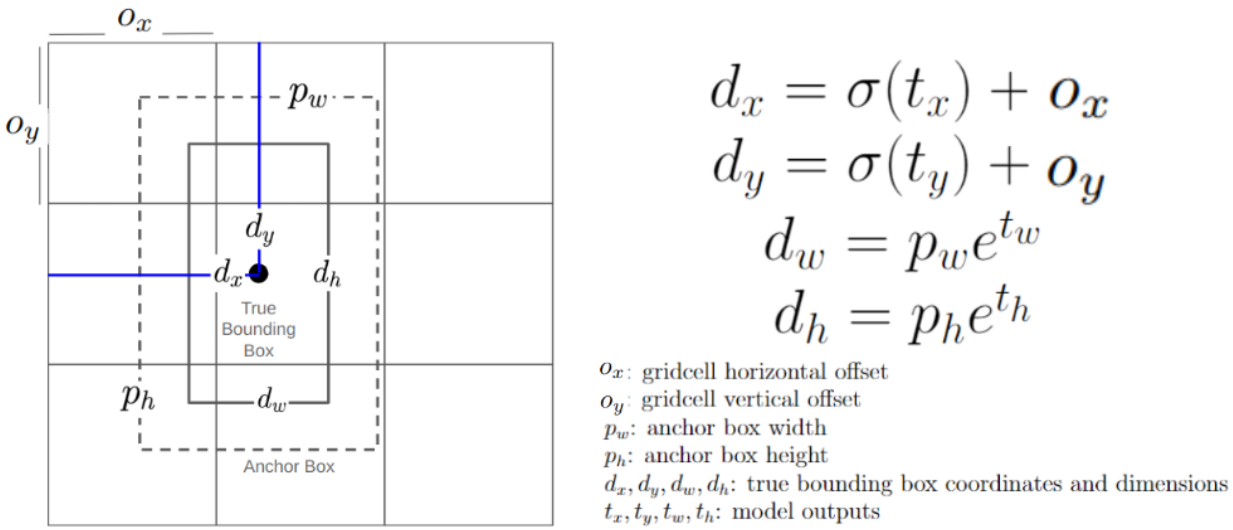


Figure 17: YOLOv3 Anchor Boxes

The equations show how the net predictions are calculated. The t_x, t_y, t_w, and t_h represent the basic model predictions of the horizontal distance from the top left, vertical distance from the top left, width prediction, and height prediction respectively. For t_w and t_h, the sigmoid function is applied, which bounds the values between 0 and 1 in an S-shaped curve. Then, a grid offset o_x or o_y is added, respectively to obtain the true horizontal distance and vertical distance from the top left corner to the actual center of the bounding box, d_x and d_y. Then, for the bounding box width and height, t_w and t_h, the base model's predictions, become an exponent, which is multiplied by the anchor width and height, p_w and p_h to obtain d_w and d_h. These net predictions are then used as the true dimensions for the bounding box.

The number of bounding boxes B is directly related to the number of predefined anchor boxes. For instance, if there are 3 predefined anchor boxes for each scale, then the number of bounding boxes B is also 3. Thus, each bounding box corresponds to a single anchor box.
To calculate the anchor boxes, a simple training process shown in Figure 18-20 was utilized. Figure 18 shows the calculation for the initial anchor box guesses. Then utilizing the fitness score shown in Figure 19, the accuracy of the anchor boxes are increased over N iterations via the training method shown in Figure 20.

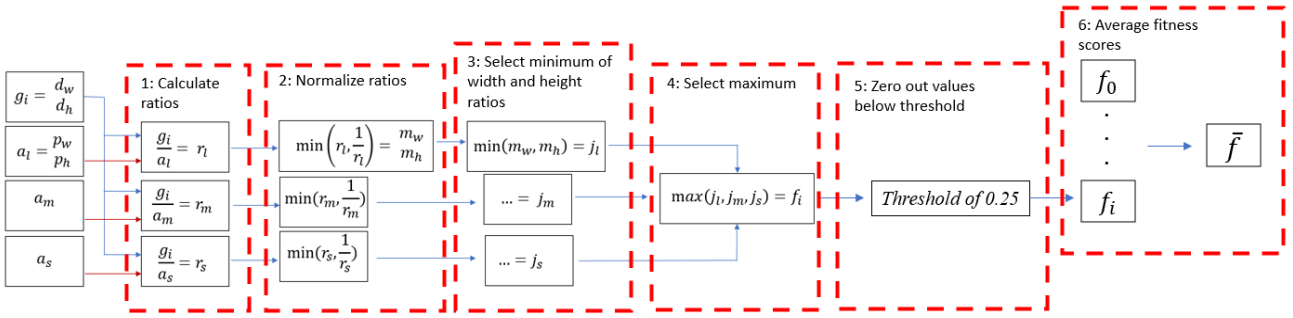
$$a_l = \frac{p_w}{p_h} = \frac{b_w + \sigma_w}{b_h + \sigma_h}$$

$$a_m = \frac{b_w}{b_h}$$

$$a_s = \frac{b_w - \sigma_w}{b_h - \sigma_h}$$

a_l: Large anchor box
a_m: Medium anchor box
a_s: Small anchor box
p_w: Width of anchor box
p_h: Height of anchor box
b_w: Average width of ground truth bounding boxes
b_w: Average height of ground truth bounding boxes
σ_w: Standard deviation of the width of ground truth bounding boxes
σ_h: Standard deviation of the height of ground truth bounding boxes

Figure 18: Initial Anchor Boxes Guess



j_l: minimum of normalized width and height ratios from step 2 for large anchor boxes
j_m: minimum of normalized width and height ratios from step 2 for medium anchor boxes
j_s: minimum of normalized width and height ratios from step 2 for small anchor boxes
m_w: normalized width ratio
m_h: normalized height ratio
d_w: bounding box width
d_h: bounding box height
f_i: fitness score for the ith bounding box
$\bar{f}$: average fitness score

Figure 19: Anchor Box Fitness Score

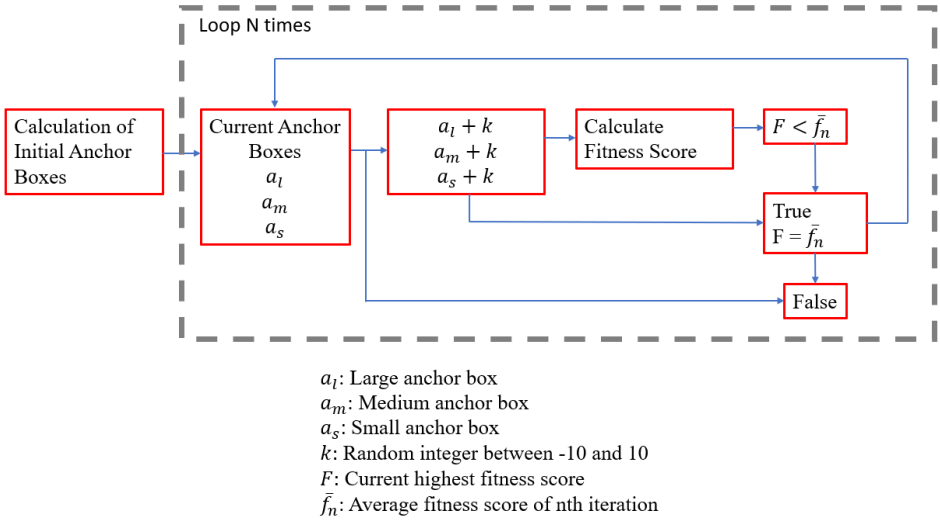


Figure 20: Anchor Box Training

NOTE: Width and height are constrained to a minimum of 15 pixels.

A key issue with the anchor box prediction is the lack of diversity in the output. Due to the nature of optimizing the fitness score, the output boxes often converge over large numbers of iterations. To address this, the preferred method was to inspect the initial guesses to verify they are diverse in height and width. If many diverse anchor boxes are required, it is recommended that an adjustment to this training method or fitness score calculation be made to incentivize diverse outputs.

Loss

Yolov3 loss, as defined in the official paper, is calculated using the following function:

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} [(t_{xij} - \hat{t}_{xij})^2 + (t_{yij} + \hat{t}_{yij})^2]$$

$$+\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} [(\sqrt{t_{wij}} - \sqrt{\hat{t}_{wij}})^2 + (\sqrt{t_h} + \sqrt{\hat{t}_{hij}})^2]$$

$$+\sum_{i=0}^{S^2} \mathbb{1}_{ij}^{\text{obj}} [-(\hat{C}_i \ln C_i + (1 - \hat{C}_i) \ln(1 - C_i))]$$

$$+\lambda_{\text{noobj}} \sum_{i=0}^{S^2} \mathbb{1}_{ij}^{\text{noobj}} [-(\hat{C}_i \ln C_i + (1 - \hat{C}_i) \ln(1 - C_i))]$$

$$+\sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \sum_{c \in \text{classes}} -(\hat{p}(c) \ln p(c) + (1 - \hat{p}(c)) \ln(1 - p(c)))$$

Figure 21: YOLO v3 Loss

$\boldsymbol{\lambda}_{\text{coord}}$: Bounding box loss scaling factor

$\boldsymbol{\lambda}_{\text{noobj}}$: No object confidence loss scaling factor

I^{obj}: 0 or 1 indicating presence of ground truth object

I^{noobj}: 0 or 1 indicating absence of ground truth object

The YOLOv3 loss consists of three parts: bounding box loss, confidence loss, and classification loss. In the above diagram, the first two terms calculate the bounding box loss, the next two terms calculate the confidence loss, and the last term calculates the classification loss. All these terms are calculated using sum squared error. However, YOLOv3 loss can be calculated using an alternative method, while still calculating bounding box loss, confidence loss, and classification loss.

$$C_{IOU} + \sum_{i=1}^{S^2} \sum_{j=1}^{B} I_{ij}^{\text{obj}} FL(C_i) + \sum_{i=1}^{S^2} \sum_{j=1}^{B} I_{ij}^{\text{noobj}} FL(C_i) + \sum_{i=1}^{S^2} \sum_{j=1}^{B} I_{ij}^{\text{obj}} FL(p_{ij}(c))$$

Figure 22: Alternative YOLO v3 Loss

C_{IOU}: complete IOU loss

$FL()$: Focal loss

The first term in the above equation accounts for the bounding box coordinates, the second term accounts for the object confidence, and the third term accounts for the classification confidence. Complete IOU loss has three terms; the first one minimizes loss by overlap area, the

second one minimizes loss by central point distance, and the third term minimizes loss by aspect ratio.

$$C_{IOU} = (1 - IOU) + D_{IOU} + R_{IOU}$$

Figure 23: Complete IOU loss

IOU is the area shared between the ground-truth bounding box and the model-output bounding box divided by the total area that the two bounding boxes cover. D_{IOU} is the Euclidean distance between the centers of the ground-truth and model-output bounding boxes divided by the diagonal length of the smallest box that contains both the ground-truth and model-output bounding boxes. R_{IOU} is the squared difference between the angles of the diagonals of the bounding boxes multiplied by some scaling factor. By convention, the scaling factor is $4/\pi^2$.

$$\mathrm{FL}(p_\mathrm{t}) = -\alpha_\mathrm{t}(1 - p_\mathrm{t})^\gamma \log(p_\mathrm{t})$$

Figure 24: Focal loss

γ: tunable parameter

α_t: balancing factor

p_t: model-output probability if comparing to ground-truth of 1; 1 - model-output probability if comparing to ground-truth of 0

Within the YOLOv3 loss function, focal loss [7] is used to account for the object confidence and classification losses. Focal loss is similar to binary cross-entropy loss, except that there is a modulating factor $(1 - p_t)^\gamma$. When there is a misclassification, the modulating factor is close to 1. When there is a correct classification, the modulating factor is close to 0. Thus, this allows the model to learn harder examples better when using focal loss compared to standard binary cross-entropy loss.

Training

To train the model, pretrained YOLOv3 weights are imported, but the pretrained weights for the detection layers are then deleted. The model is then left to train on the dataset, whether that be ETH, WIDER 2019, or Eurocities. Because the pretrained weights require a specific YOLOv3 architecture, the model was adjusted to take input images of size 608 by 608. In addition, the feature maps were also adjusted to be 19 by 19, 38 by 38, and 76 by 76. For comparison, the model originally took input images of size 256 by 256 and had feature maps of size 8 by 8, 16 by 16, and 32 by 32.

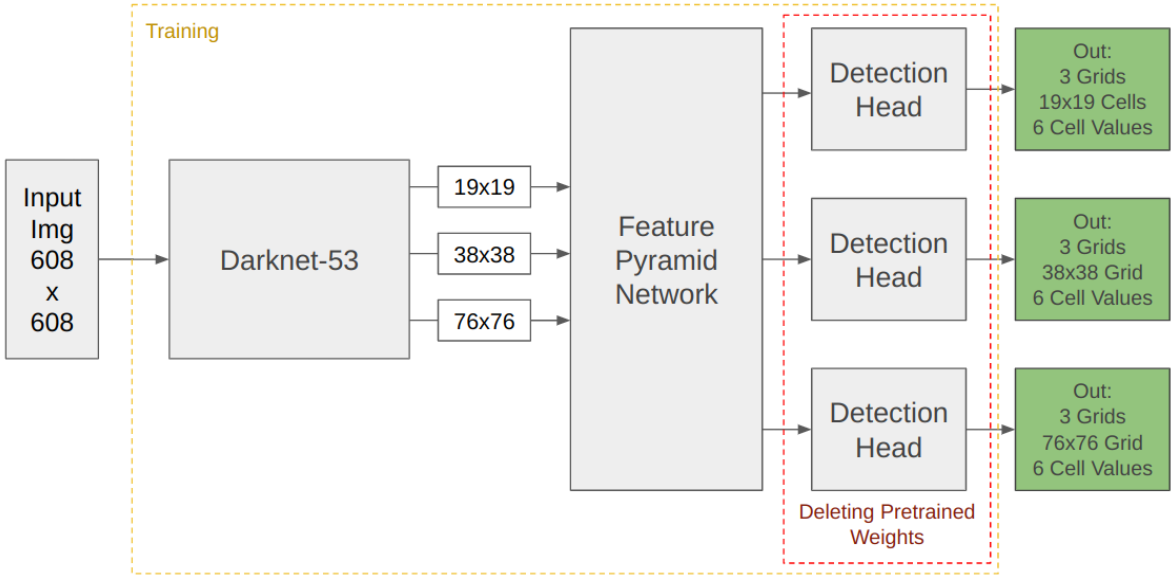


Figure 25: YOLOv3 Training

Faster R-CNN

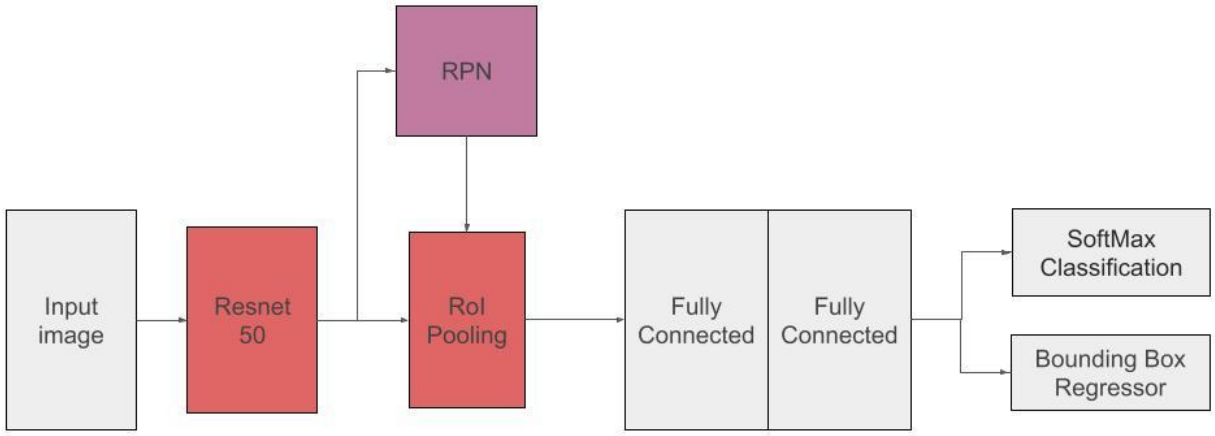


Figure 26. Faster RCNN Model Diagram

This model architecture first extracts image features using a series of convolutional layers and residual connections called Resnet-50. The SoftMax classification layer measures the probability of the class. The bounding box regressor measures the coordinates of the output boxes.

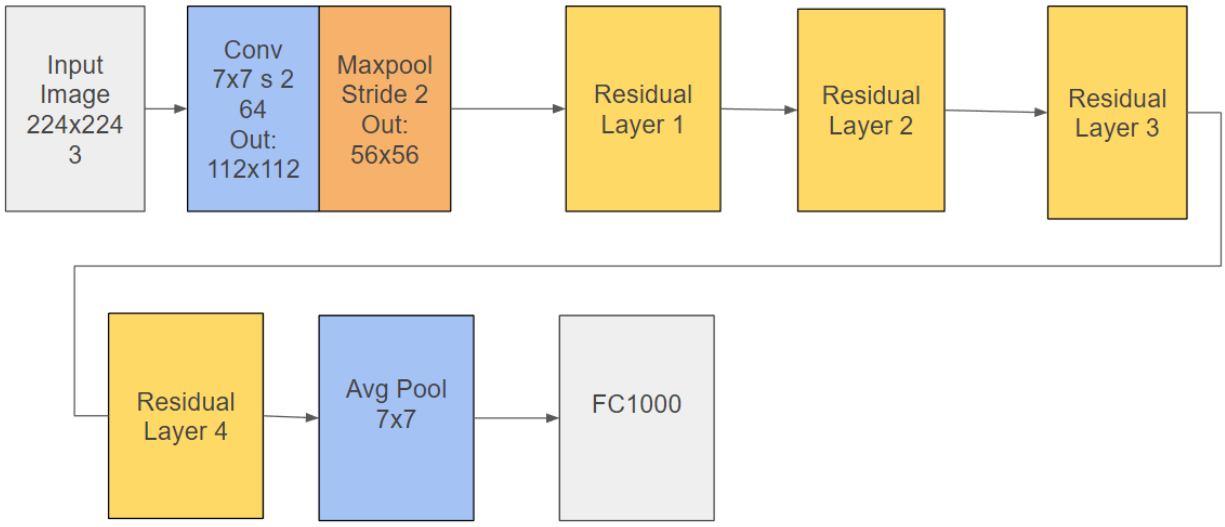


Figure 27. Resnet-50 Model Diagram

ResNet50 is a convolutional neural network (CNN) architecture. It consists of 50 layers including convolutional layers, pooling layers, and residual layers. The extracted features will be transferred into RPN network and RoI pooling layers.

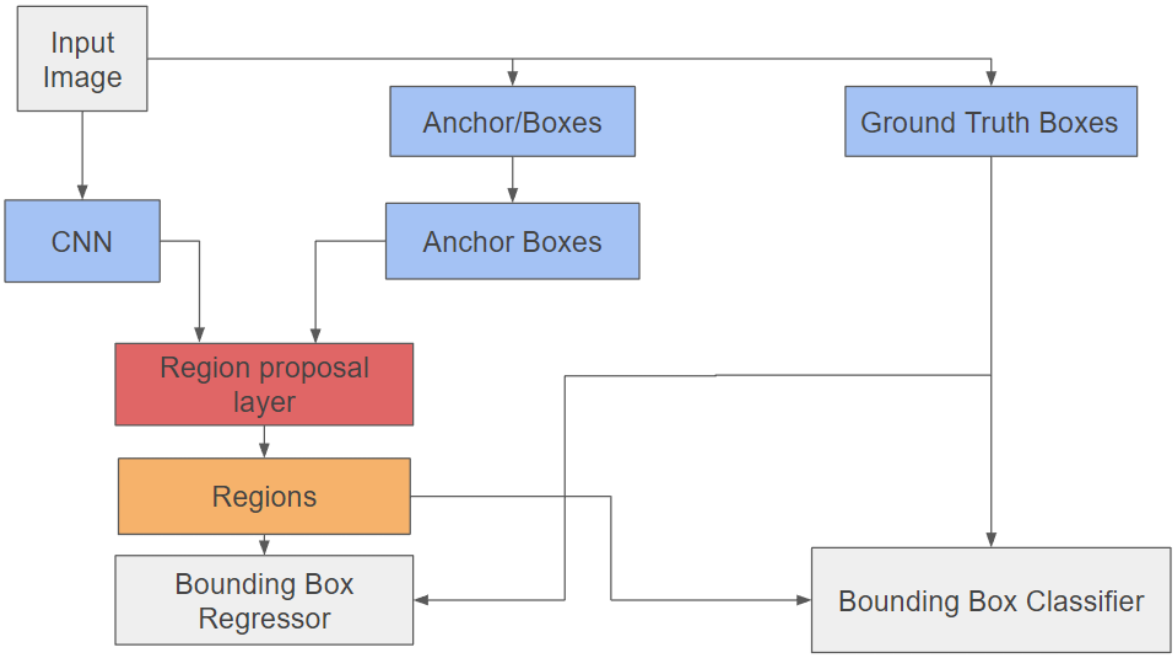


Figure 30. RPN Model Diagram

The RPN network takes an input image and generates a set of anchor boxes, which are potential object locations at different scales and aspect ratios. The network then assigns a probability to each anchor box, indicating the likelihood that it contains an object, and predicts an offset from the anchor box to the true object boundary. These predicted object proposals are then fed into the subsequent layers of the Faster R-CNN framework for classification and bounding box regression.

Loss

RPN Loss *Faster RCNN Loss*

$$L\left(\{p_i\},\{t_i\}\right) = \frac{1}{N_{cls}}\sum_i L_{cls}\left(p_i, p_i^*\right) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) + L_{cls}(p_i, {p_i}^*) + \lambda L_{reg}(t_i, {t_i}^*)$$

N_{cls}: batch size
N_{reg}: number of anchor locations
λ: parameter coefficient (default=10)
L_{cls}: classification loss
L_{reg}: bounding box regression loss

i: anchor index
p_i: output anchor object confidence
p_i^*: ground truth object label
t_i: output bounding box values

Figure 28. Faster R-CNN Loss

Pre-trained Results

One method to reduce training time and ensure the model is functioning correctly is to used pretrained weights. This avoids the time consuming task of determining accurate weights for each node by importing them from a previous run or model. In our case, we imported the weights from the COCO dataset and used them as a starting point for both YOLOv3 and Faster R-CNN. This greatly increased the accuracy and efficiency of our models, which we later fine tuned with additional training. Results shown below are produced by using only pretrained weights.

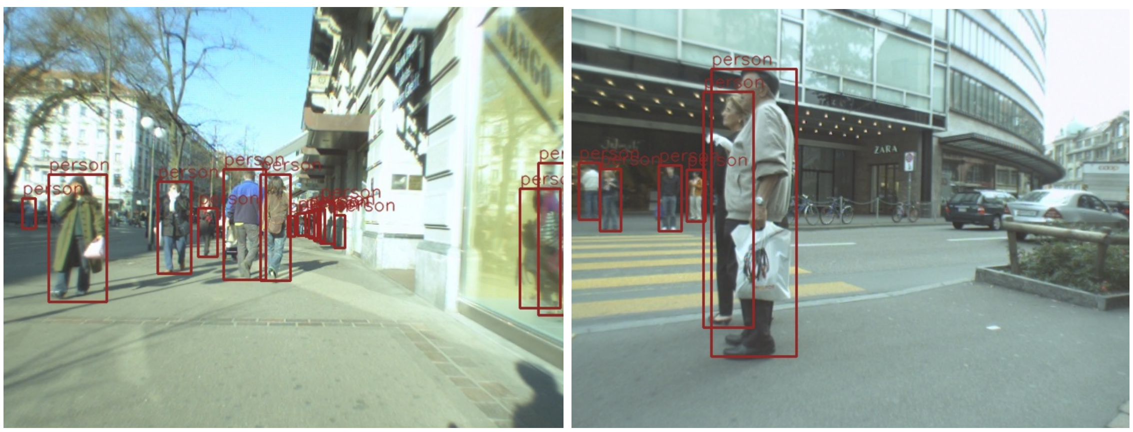


Figure 29: Results on Pretrained COCO Weights

As seen above, the pretrained weights were accurate in detecting the humans in the images. Unfortunately, the model detects reflections in the windows as full human beings. Additional training of the model was required to remove those artifacts.

Results

YOLO v3

After training on booth WIDER andd ETH individually and cross evaluating the results, an early stop mechanism was implemented to prevent overfitting. If the model's loss for validation remained higher than its lowest value for 10 epochs, the training was stopped and final results on the test set collected. YOLO v3 was then trained on 200 images from ETH with a learning rate of 0.0001 for 52 epochs, and tested on the WIDER 2019 dataset.

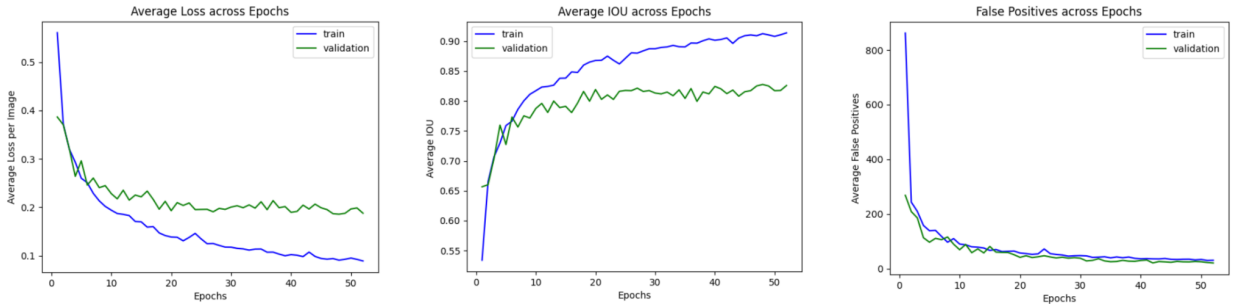


Figure 30. Graphs of loss per epoch, IOU per epoch, and False positives per epoch respectively

Figure 31. Left image is ground truth for an ETH sample, right image is model output

Figure 32. Left image is ground truth for an WIDER sample, right image is model output

	IOU	Average False Positives (FP)
WIDER 2019	0.412	4.65
ETH	0.923	6.21

Table 1. IoU and False Positives for both WIDER 2019 and ETH datasets for YOLO v3

Faster R-CNN

Faster R-CNN was trained on 283 images from the ETH dataset. Its learning rate can be adjusted after each epoch or be fixed, depending on the the mode selected by the user. The model was initialized with a learning rate of 0.005 and trained on RTX 2080 super and with a batch size of 2. Each epoch cost around 1.5 minutes. Through training, the model overcomes the overfitted problem, which it will not detect the shadow any more and can even detect human not bounded in the ground truth bounding box.

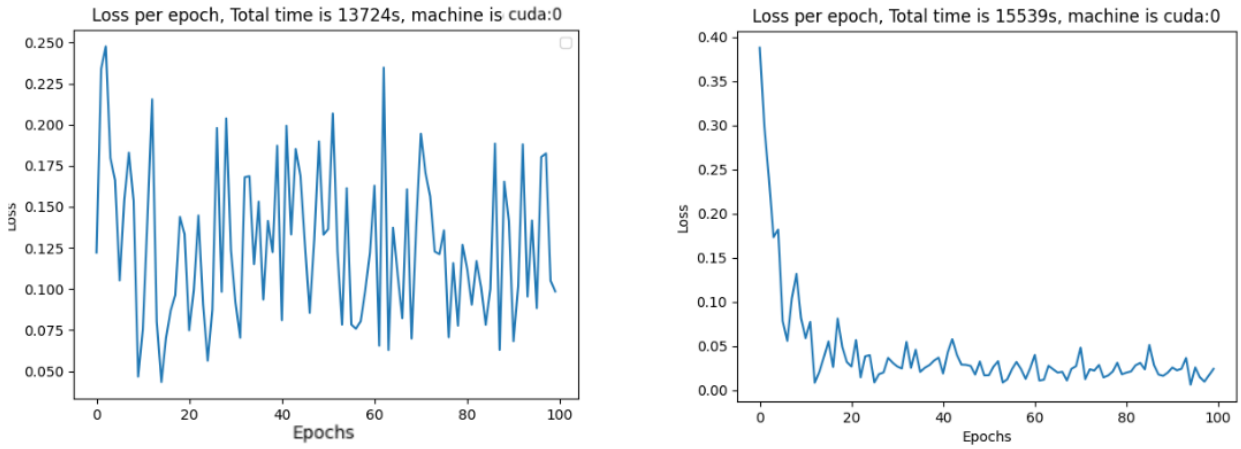


Figure 33. Left plot is the loss with learnable learning rate. Right side is the loss with fixed 0.005 learning rate

IoU Metrix Train on the ETH after 100 Epoches		
	Average Precision(AP)	Average Recall(AR)
WIDER 2019	0.654	0.671
ETH	0.906	0.756

Table 2. Metrics on both WIDER 2019 and ETH datasets after 100 epoches

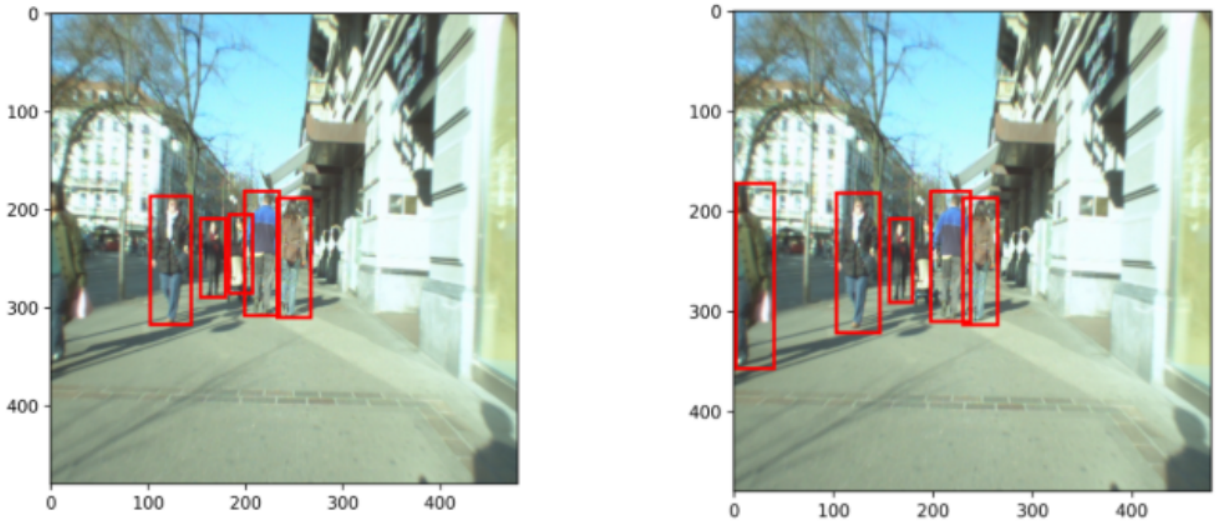


Figure 34. Left image is the gound truth image of the ETH sample, the right image is the model output

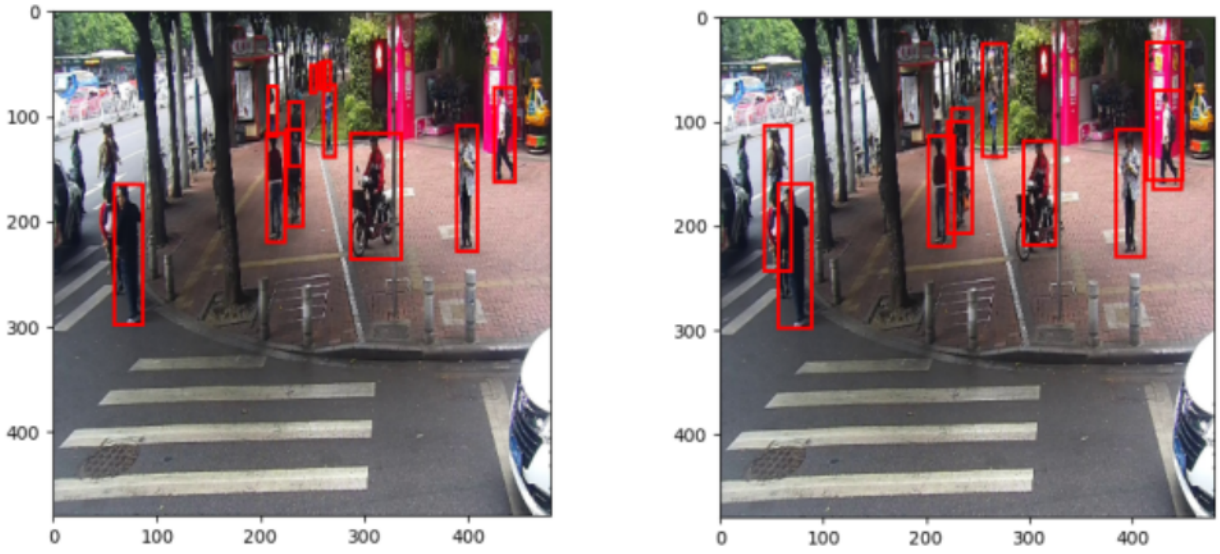


Figure 35. Left image is the gound truth image of the WIDER 2019 sample, the right image is the model output

Conclusions

The WIDER 2019 dataset is a highly varied dataset due to it's numerous subjects and high camera angles. Though the ETH dataset does provide a large variance in distance from the camera, models trained on it do not easily detect people far in the background. Because of these features, when cross evaluating, WIDER 2019 provides a through test to a model generalizability if it is trained on ETH.

Pretrained YOLO v3 learns human detection datasets relatively quickly, but may more easily overfit than contemporary models. Using weights acquired from a model pretrained on the COCO object detectio dataset, it achieves upwards of 0.92 IOU on ETH in around 50 epochs. While these results are notable, cross-dataset evaluation reveals that it generalizes very poorly to the WIDER 2019 dataset, achieving an IOU of only 0.41. Further training on ETH only widens this gap in collected metrics, indicating that the model is overfitting. While YOLO v3 trains very quickly, it is subject to the same issues of generalizability as many pedestrian detection models. The Faster R-CNN requires more training epochs and more training time. It achieved an average recall of 0.76 on its training dataset, and an average precision of 0.65. On its testing dataset, the gap between the average precision was significant, with a testing precision on WIDER of 0.65, but the gap between the average recall was fairly diminished, with a testing average recall f 0.67. On visual inspection, the testing results for Faster-RCNN are significantly better than those of YOLOv3. Moving forward, Faster RCNN is the superior model for a general human detector, its only downside being the increased training and output time, but it is otherwise significantly more precise on external datasets.

Future Work

Subsequent work on this project could compare the Faster RCNN model to currently relevant pedestrian detectors such as Pedestron and F2Dnet on the Eurocity Persons dataset, or on WIDER 2019 as both are large datasets that contain streetviews. A later version of YOLO could also be implemented, as the more recent versions heavily optimize the original structures of YOLO v1 and v3. These models could all be compared using mean average precisions on newer datasets that contain other natural events that obscure or distort people that the tested datasets lack, such as heavy snow and rainfall or rotated angles of view.

References

[1] Hasan, I., Liao, S., Li, J., Akram, S. U., & Shao, L. (2020, December 9). Generalizable Pedestrian Detection: The Elephant In The Room. *arXiv*. Retrieved November 13, 2022, from https://arxiv.org/pdf/2003.08799.pdf

[2] WIDER Face & Person Challenge 2019 - Track 2: Pedestrian Detection (n.d.). *CodaLab*. Retrieved November 13, 2022, from https://competitions.codalab.org/competitions/20132#learn_the_details-overview

[3] A. Ess and B. Leibe and K. Schindler and and L. van Gool (2008, June). A Mobile Vision System for Robust Multi-Person Tracking. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR'08)*. *IEEE Xplore*. Retrieved November 13, 2022, from https://data.vision.ee.ethz.ch/cvl/aess/dataset/

[4] J. Redmon, S. Divvala, R. Girshick and A. Farhadi (2016). You Only Look Once: Unified, Real-Time Object Detection. I*EEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91. Retrieved November 13, 2022, from https://arxiv.org/pdf/1506.02640.pdf

[5] J. Redmon and A. Farhadi (2018). YOLOv3: An Incremental Improvement. I*EEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, arXiv: 1804.02767. Retrieved November 13, 2022, from https://arxiv.org/pdf/1804.02767.pdf

[6] Tsung-Yi Lin, Maire, M., Belongie, S. J., Bourdev, L. D., Girshick, R. B., Hays, J., … Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *CoRR*, *abs/1405.0312*. Retrieved from http://arxiv.org/abs/1405.0312

[7] Lin, T.-Y., Goyal, P., Girshick, R., He, K., & Dollar, P. (2020). Focal Loss for Dense Object Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42(2), 318–327. https://doi.org/10.1109/TPAMI.2018.2858826

[8] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards real-time object detection with region proposal networks. Advances in neural information processing systems, 28. https://doi.org/10.48550/arXiv.1506.01497

[9] Girshick, R. (2015). Fast R-CNN. In Proceedings of the IEEE international conference on computer vision (1440-1448). https://doi.org/10.48550/arXiv.1504.08083

Appendix

Robert Sego

Robert has been part of the Vertically Integrated Projects' Image Processing and Analysis team for 8 semesters and used his experience from prior projects to lead the Human Detection project. He held and scheduled the initial team selection meeting and collaboratively defined the goal and structure of the project. He led twice-weekly meetings and assigned subtasks and presentation responsibilities to individual team members. He created multiple tutorials for using common Python image-processing libraries, working with neural network frameworks, and accessing other shared resources for the VIP-IPA team.

After investigating various different models for performing pedestrian detection, he created a simple five-layer convolutional model to test simple machine-learning solutions. He then directed the team toward the You Only Look Once (YOLO) group of neural network architectures. In the first semester of research, he implemented YOLO v1 and refactored it for both local machines and the Bluepill computing cluster. He collected the WIDER 2019 dataset and created the data loader framework for both it and future datasets the team would add to the project, such as ETH Pedestrian. He collected and analyzed the output images of various different model structures and hyperparameter configurations, and determined that the team should move toward the more computationally expensive but effective YOLO v3 architecture. He created new model, loss, and training scripts for this more advanced architecture, and developed a cross-framework pretraining loader to speed training of the model. Once he had collected metrics and example detections on the ETH and WIDER datasets, he compared the results for different input resolutions, anchor-box selections, and other hyperparameters.

Robert pulled on old work from previous classes to give the team a crash course in neural networks and the math behind them such as gradient descent and activation functions. He spent time with individual team members to discuss each of these methods and ensure progress and understanding for the entire team. Overall, Robert helped the team both organizationally and technically using insights and experience from prior semesters with VIP-IPA.

Xilai Dai

Xilai has been part of the Vertically Integrated Projects' Image Processing and Analysis team for 2 semesters. He initilized ideas for human detection and expect to work on the trajectory prediction if have time after implementing human detection. He helped the new team members to catch up on basic image processing skills and led the implementation of Faster R-CNN. He also helped revised YOLOv3.

After researching on different models on human detection. He made a simple several layers CNN model to build a basic neutral network and based on that point to learn further about deep learning. Initially, Xilai was working on YOLOv1 model in the first semester. After successfully implementing the model, he trained on the WIDER 2019 dataset and test on that dataset. This was accelerating on the Bluepill server. This model did not perform as well as expected, so Xilai turned to look at the RCNN families: RCNN, Fast RCNN, and Faster RCNN in the second semester. Firstly, Xilai implemented VGG-16 model and tested on WIDER 2019 dataset to extract the image features. After successfully extracting the image features, Xilai began to implement the pre-trained Faster RCNN, which was trained on the COCO dataset. Then Xilai replaced VGG-16 by Resnet-50 for feature extraction. The pretrained model performs well but still has the overfitted problems on our datasets: the ETH and the WIDER 2019.

After the pre-trained Faster RCNN worked, Xilai began to finetune the model. Firstly, he changed model structure to fit human detection, which means the model just detects background and human. Then Xilai implemented dataloader for the ETH dataset and the WIDER 2019 dataset to fit the Faster RCNN format. After dataloader worked, Xilai noticed that the model output bounding boxes are often fitted together so he applied the non-maximum suppression method to eliminate duplicate detections of the same object. Lastly, the funetined Faster RCNN fixed the overfitted problem on the ETH and the WIDER 2019 datasets, and even detect some human who was not included in the ground truth but shown in the image.

Xilai built the pre-trained Faster RCNN model, implemented the associated dataloader for the ETH dataset and the WIDER 2019 dataset, loss, training process, testing process, and results visualization on his own. Xilai also helped debugging YOLOv3. Overall, Xilai respond on Faster RCNN and helped the whole team on track.

Alex Weber

Alex has been a part of VIP for two semesters and used his experience in python to build functionality for the Human Detection team's designs. Examples include his extensive involvement in the research and implementation of different datasets, such as ETH in the team's dataloader, as well as quality of life improvements such as the config file implementation to store commonly manipulated settings. His work has also been seen in the troubleshooting and running of the team's models.

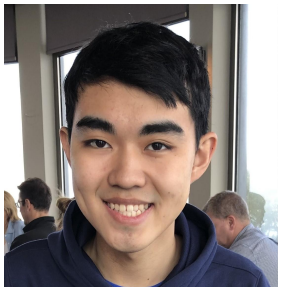
This semester Alex started in researching YOLOv3 to help with the adjustment of last semester's YOLOv1 code to YOLOv3. After reading the original papers and analyzing the previous work done by the Lane Detection team on YOLOv3, he built the initial Feature Pyramid Network and Detector sections of the model. In doing so, it was noticed that the team did not have anchor box data specifically for the datasets. This led him to begin research into anchor box generation and training.

As discussed above in this report, the anchor boxes are key to YOLOv3. Each dataset needs it's own anchor boxes as each has different ground truths. Alex went about building a dynamic anchor box generation and training system to solve this problem. The results can be seen above in the Anchor Box section of the YOLOv3 report. These results assisted in making more accurate predictions from the YOLOv3 model in less time than before.

In his anchor box work, Alex noticed there was not an efficient way to switch between sets of anchor boxes depending on the dataset and training run. To solve this, he researched a python implementation of the .ini config standard he used with LabVIEW at his internship with Caterpillar inc. Once the generation and reading functionality was complete he instructed the team in the use of the tool and created documentation for future semesters.

During the last stretch of the semester, Alex focused on running the models to get results for the team. This time also included extensive troubleshooting of bugs as he switched his focus to the Faster RCNN model with results seen above.

Patrick Li



Patrick has been a part of VIP for 2 semesters and has used his experience in python to help research and test the python code that was developed by the Human Detection team's design. Patrick mainly helped develop the the loss function for the yolov3 function by reading and analyzing other research papers on their loss functions. His main function later in the project was to help train and test the model on the ETH, WIDER, and EuroCities dataset.

This semester, Patrick started researching YOLOv3 in how it can be used to improve object detection in order to better understand the teams transition from YOLOv1 to YOLOv3. Not feeling that this was satisfactory, Patrick then went and performed research on YOLOv2 and Retinanet in order to gain a better understanding of the inner workings of YOLO. This lead him to conduct research on Feature Pyramid Networks which showed new ways of approaching our data and training the YOLO model by using the bottom-up feature pyramid network.

After gaining a better understanding on the improvements YOLOv3 had to offer, Patrick then started work on the YOLOv3 loss function. For example: the issue with the code was that there was an issue with how the model understood the self.metric.box_iou function. He reached out to the previous writers of the code in order to gain a better understanding of it for the function to work properly.

The next step of Patricks goals was to start getting the Eurocities dataset training on the model. This was a difficult step as the Eurocities dataset contained different formats for the training labels as they were in a JSON file type. One of the major issues was that the dataset was massive so uploading it to bluepill was a complicated and lengthy process that require multiple attempts due to network issues. In the end, Patrick was able to get both the model and the Eurocity dataset onto bluepill however he encountered a major issue in that the model was incapable with outputting images into the directory. Multiple attempts on different operating systems did not yield significant results so Patrick had to debug the code from the beginning. Eventually, it was discovered that the Windows operating system was preventing Patrick from writing any kind of file to the specified directory and rewriting the way the program constructed the output image was needed.

Sun Ahn

This was Sun's first semester on the Image Processing VIP team. Although Sun had prior experience coding, his exposure to Python was quite limited. In addition, Sun was also unfamiliar with machine learning in general. During the first few weeks of the semester, Sun focused on learning about image processing and neural networks through official papers and onboarding materials provided by other team members. In the process, Sun successfully created various image filters for grayscaling, gaussian blurring, and sobel edge detection.

Sun then focused on the team's implementation of the YOLOv3 model. In order to better understand the structure of YOLOv3, Sun spent several weeks reading through the official papers for YOLOv1, YOLOv2, and YOLOv3. After gaining a solid understanding of the model's architecture, Sun was able to implement a loss function in PyTorch for the team's YOLOv3 model. Although the team ultimately decided to use a loss function created by the previous semester's Lane Detection team, the experience greatly benefitted Sun in his understanding of YOLOv3 as well as machine learning models in general.

For the rest of the semester, Sun spent his time debugging, testing, and improving the team's YOLOv3 model. For example, Sun fixed the non-maximum suppression algorithm so that duplicate bounding boxes were successfully removed. In addition, Sun was able to get a pre-trained YOLOv3 model to work with images from the ETH and WIDER datasets. The team's YOLOv3 model was then given the weights from the pre-trained model so that the team did not need to train their YOLOv3 model from randomized weights. Lastly, Sun produced code to plot validation data as well as code to dynamically stop training of the model to prevent overfitting.