Trimas Christos

Comparison of Artificial Intelligence systems for the detection of objects on UAV-based images.

Πολυτεχνείο Κρήτης

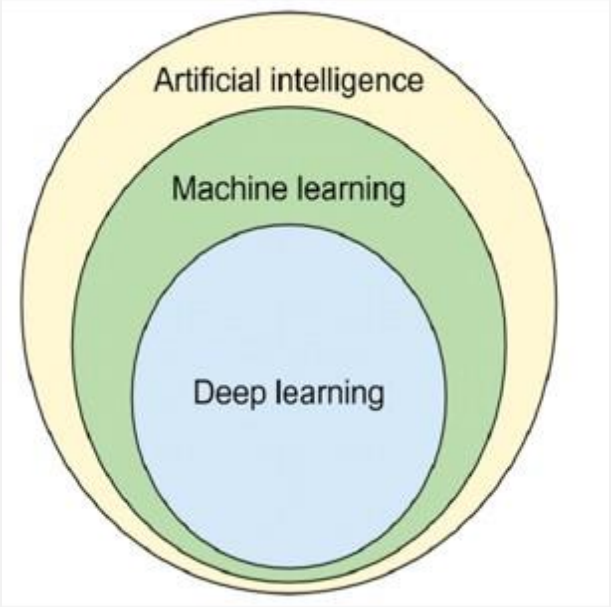
Contents

- **What are UAVs?**
- **Object Detection systems.**
- **Convolutional Neural Networks.**
- **Two-stage / single-stage detectors.**
- **RetinaNet.**
- **Modified RetinaNet.**
- **Conclusions.**
- **Limitations and Future Work.**

UAVs

Unmanned Aerial Vehicles:

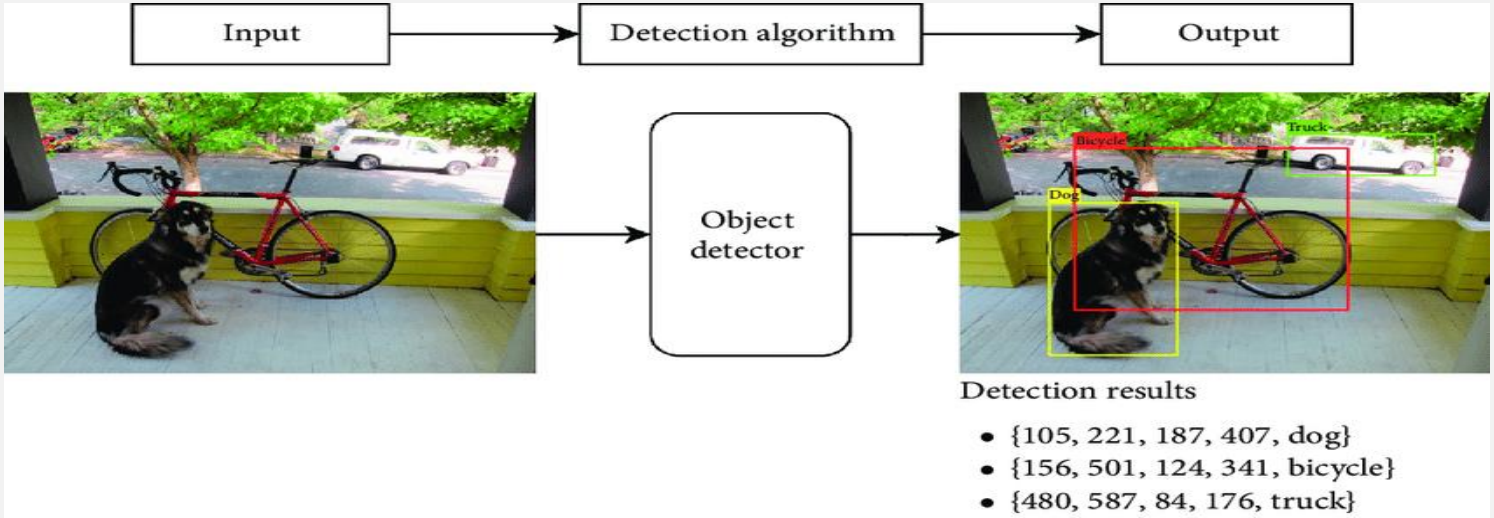
- **Search and Rescue missions.**
- **Surveillance.**
- **Information gathering.**
- **Object Detection.**



Evolution of Object detection

- **Better electronics = better cameras.**
- **Artificial Intelligence.**
- **Data Revolution.**
- **Higher Computational Power.**

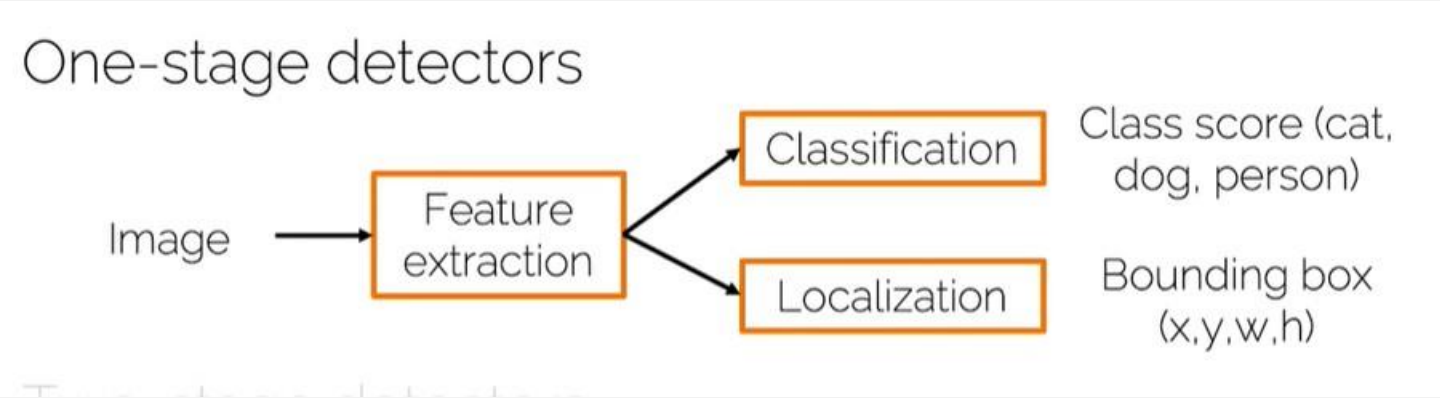
Object Detection

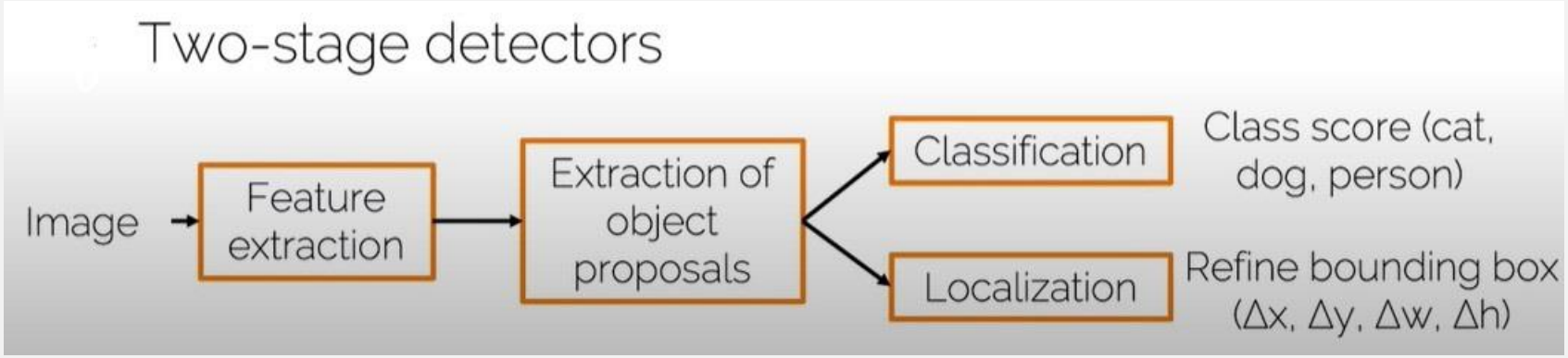


General Idea: Object detection = localization + classification.

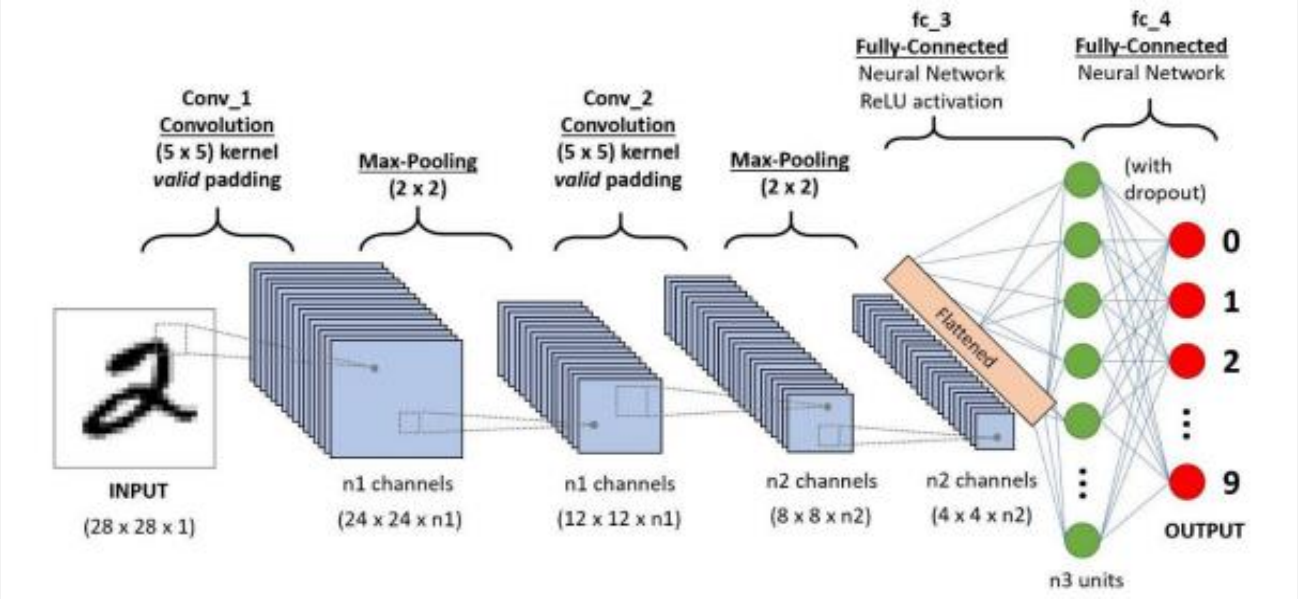
Two Stage detectors: First stage generates a sparse set of candidate proposals that should contain all objects while filtering out the majority of negative locations. Second stage classifies the proposals into classes. Well known architectures are R-CNN, Fast/er R-CNN.

Single Stage detectors: Requires a single pass through the Neural Network and predicts all the bounding boxes in one go. They have been tuned for speed, but their accuracy trails that of two stage methods. Well known architectures are YOLO and RetinaNet.


One-stage detectors
Image
Feature extraction
Classification
Localization
Class score (cat, dog, person)
Bounding box (x,y,w,h)


Two-stage detectors
Image
Feature extraction
Extraction of object proposals
Classification
Localization
Class score (cat, dog, person)
Refine bounding box (Δx, Δy, Δw, Δh)

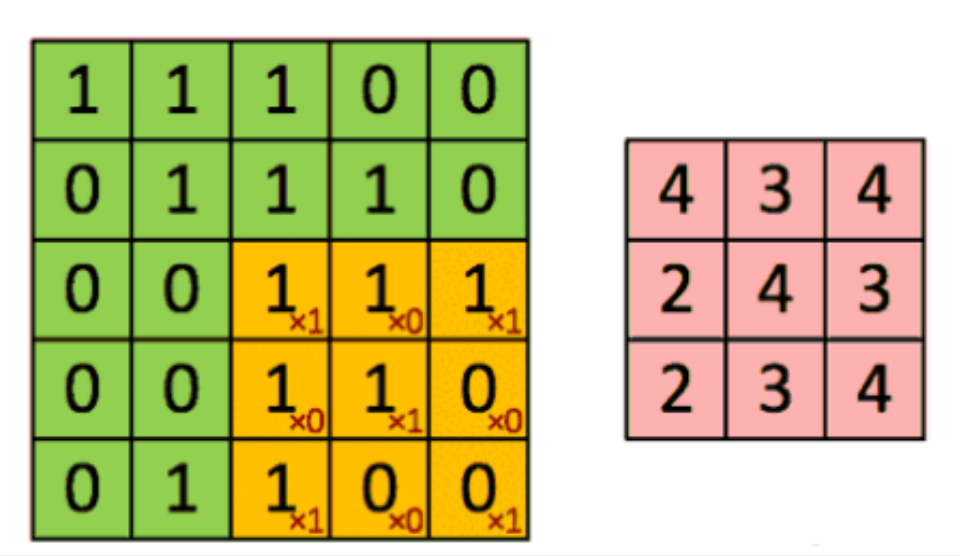
A typical **CNN**:



- Convolution Layers
- Max / Average Pooling Layers
- Fully Connected Layers

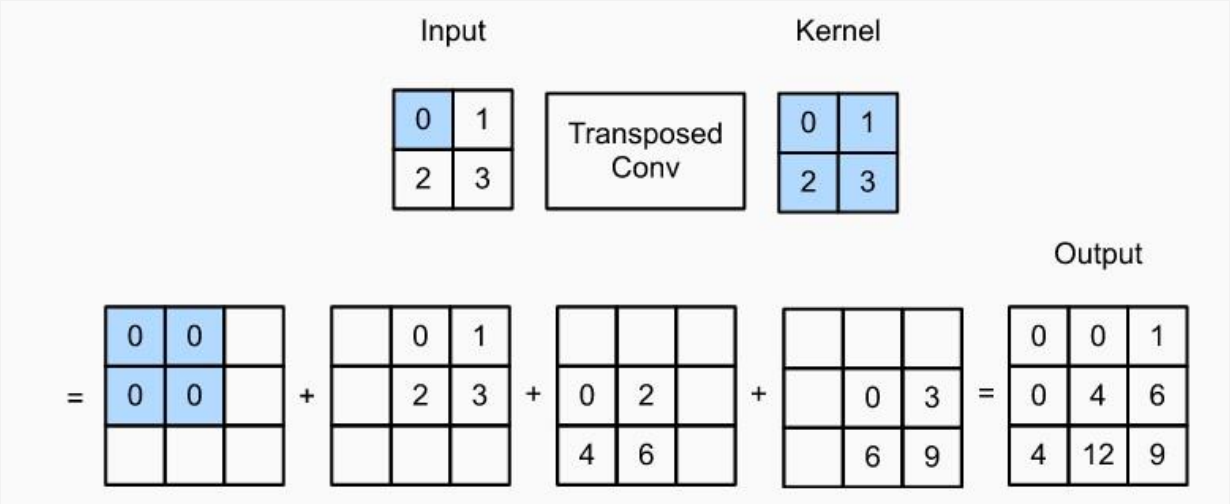
Convolution:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$



Goal: Extract high-level semantically rich features from the input image

Transpose Convolution:

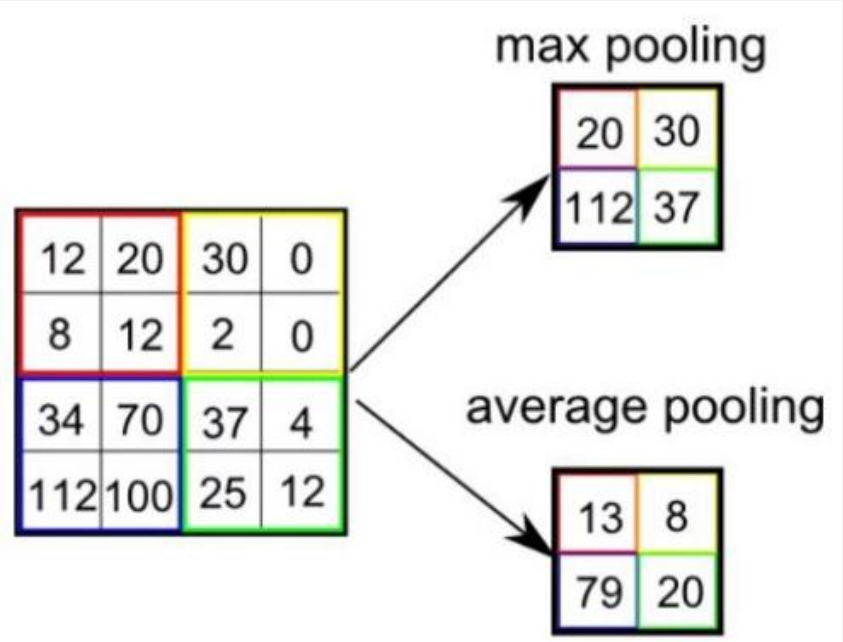


Goal: to increase (up-sample) the spatial dimensions of the intermediate feature maps

Pooling Layer

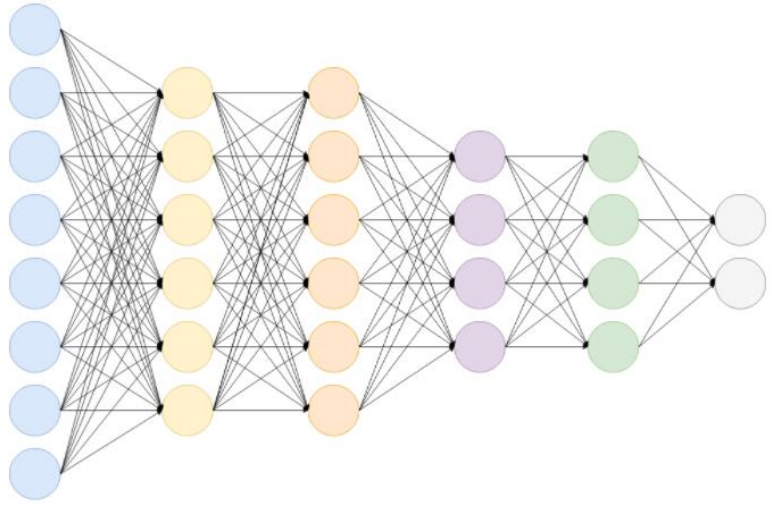
Two types: Average and Max Pooling.

Goal: Decrease computational power required to process data.
Extract dominant features through dimensionality reduction.



Fully Connected Layers

Cheap way of learning non-linear combinations of the high-level features as represented by the output of the convolution and pooling layers.

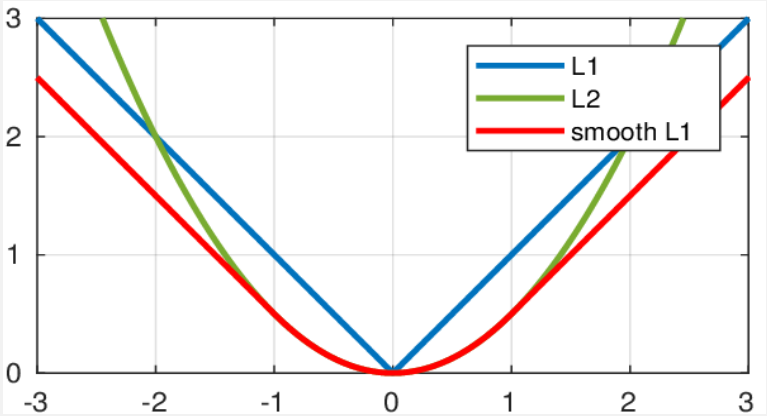


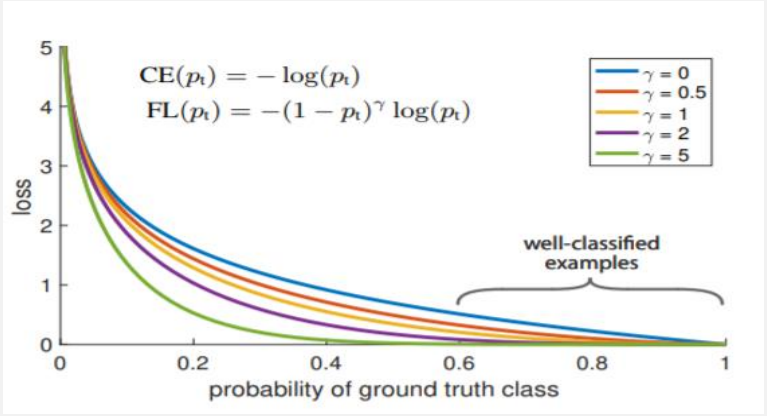
Final layer: gets the probabilities and classifies each input into particular classes.

Loss function : the prediction error of the Artificial Neural Network.
Responsible for the update of the weights of the Neural Network.

$$sL1 = \begin{cases} |\mathbf{x}-\mathbf{y}|, & if \ |\mathbf{x}-\mathbf{y}| > a \\ \frac{1}{|a|}(x-y)^2, & else\ if \ \ |\mathbf{x}-\mathbf{y}| \le a \end{cases}$$

$$FL(p_t) = (1-p_t)^{\gamma} log(p_t)$$



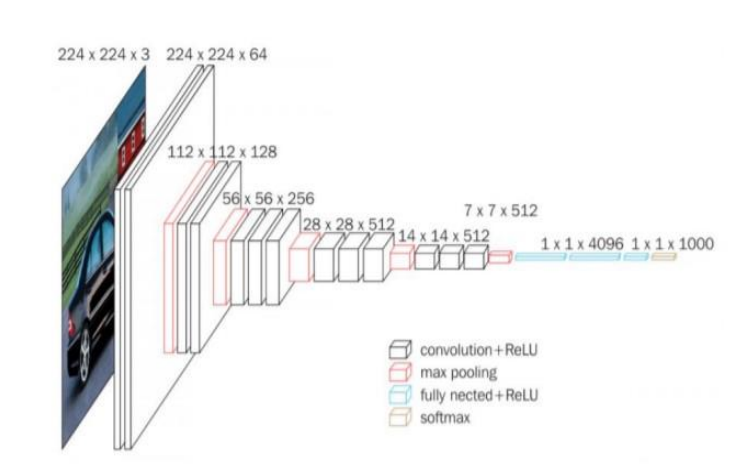


CNNs v Fully CNN

CNNs:

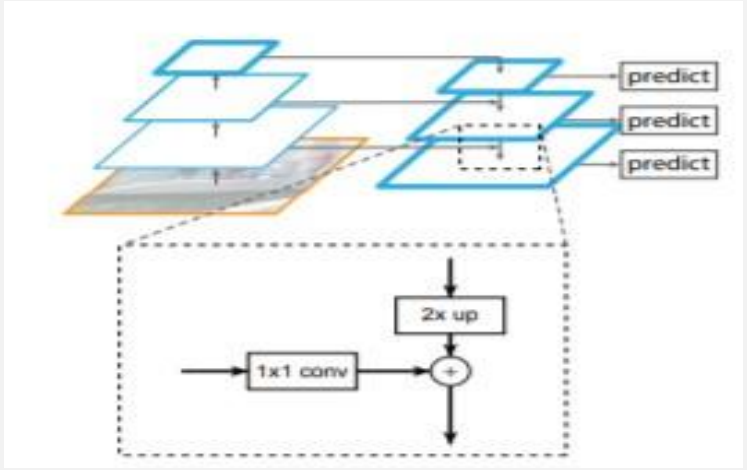
- Contains Fully Connected Layers.
- Can be used as backbone networks.
- ResNet, VGG, etc.

A typical **CNN** usually contains fully connected layers.



Fully CNNs:

- No Fully Connected Layers.
- Performs only Convolution, up-sampling and down-sampling operations.
- Can handle different input sizes.
- Unet, Feature Pyramid Network, etc.

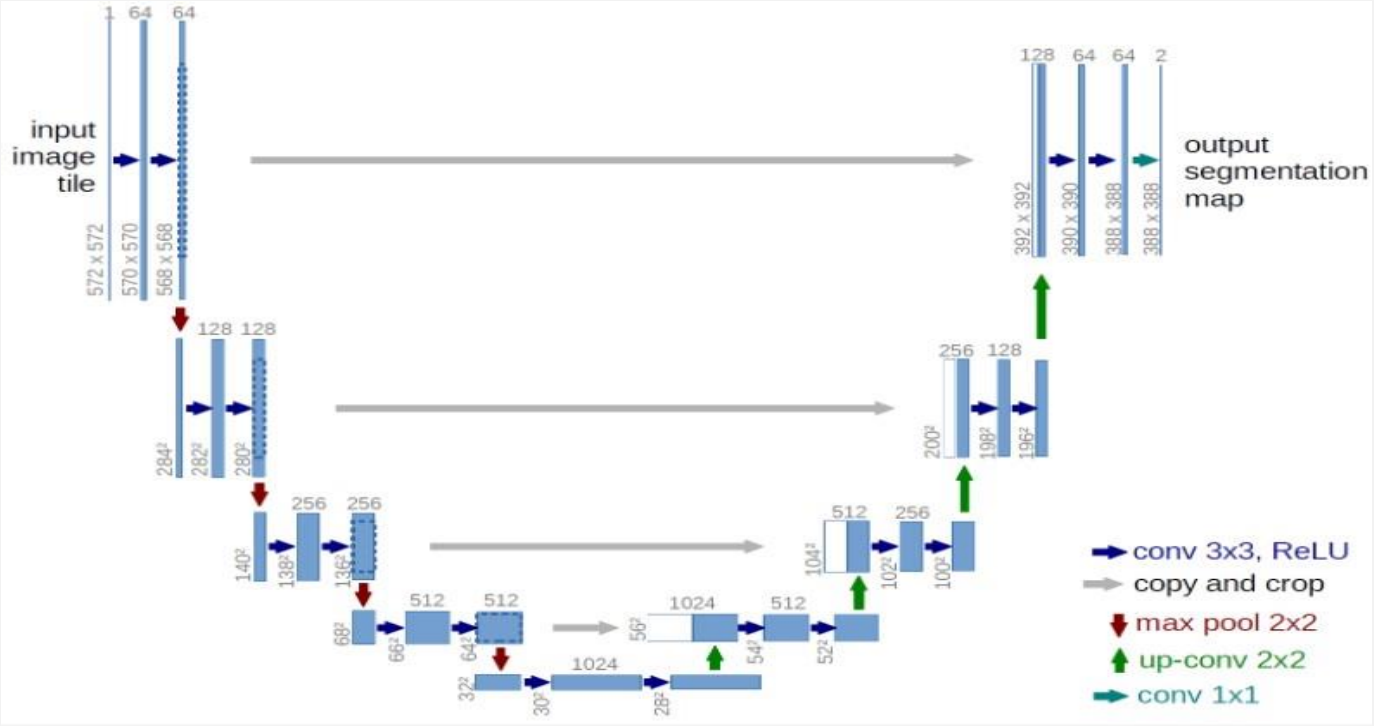


U-Net

A well known Fully Convolutional Network is the UNet. It is commonly used for semantic segmentation, but its architecture is quite similar to the Feature Pyramid Network.

Top-down pathway: Simple ConvNet consisting of convolutions and max pooling operations.

Bottom-up pathway: "Deconvolution" and convolution operations as well as concatenation to combine high level and low level features.



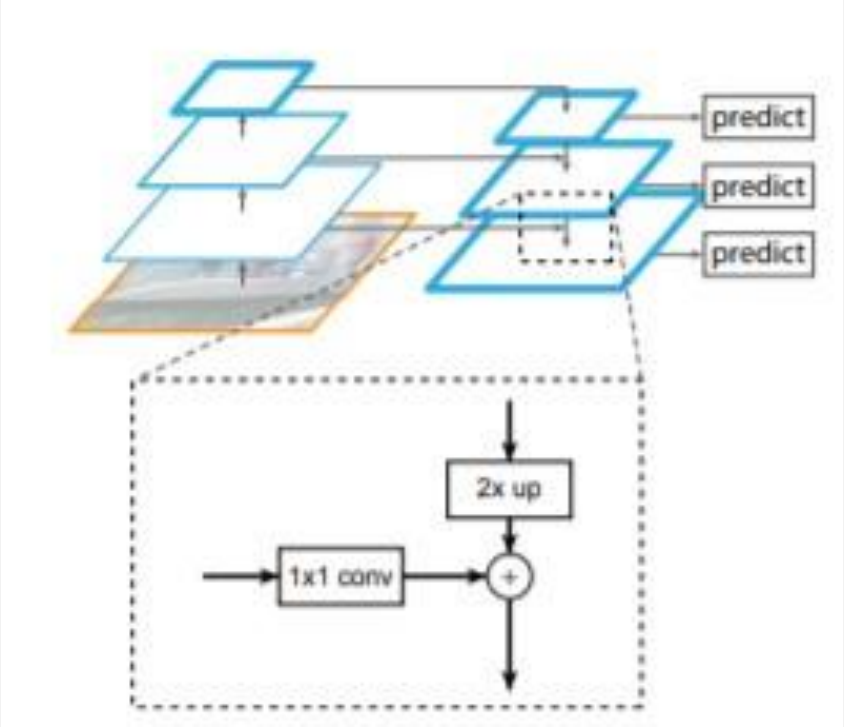
Feature Pyramid Network

Feature Pyramid Network can be used as backbone network for object detection or segmentation. Consists of an "encoder" (bottom-up pathway) and a "decoder" (top-down pathway).

Bottom-up pathway: Usually a ConvNet is used for feature extraction such as VGG or ResNet.

Using lateral connections the output of the ConvNet is fed to the top-down pathway as input.

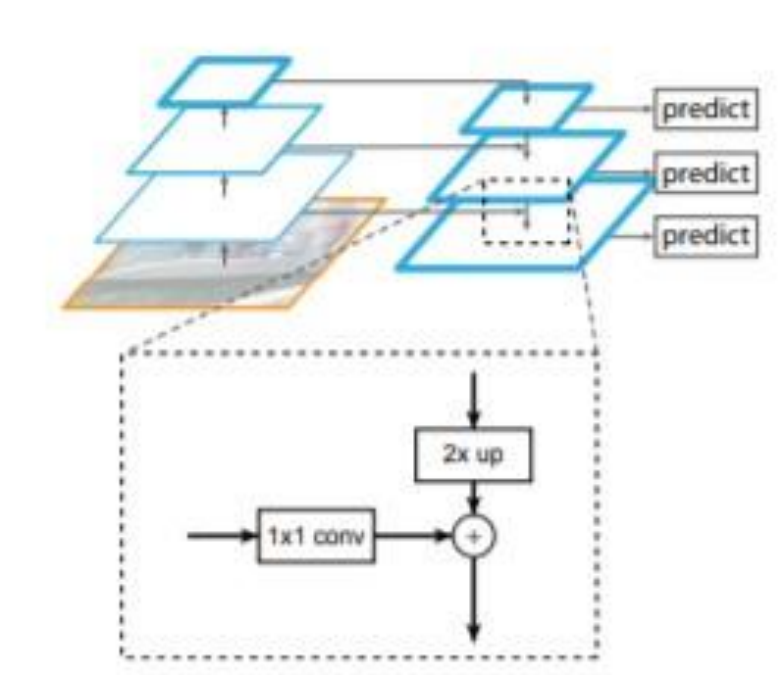
Top-down pathway: Using up-sample to match the spatial dimensions between the output of two consecutive Convolutional Blocks it adds the outputs in order to create a rich multi scale convolutional pyramid.

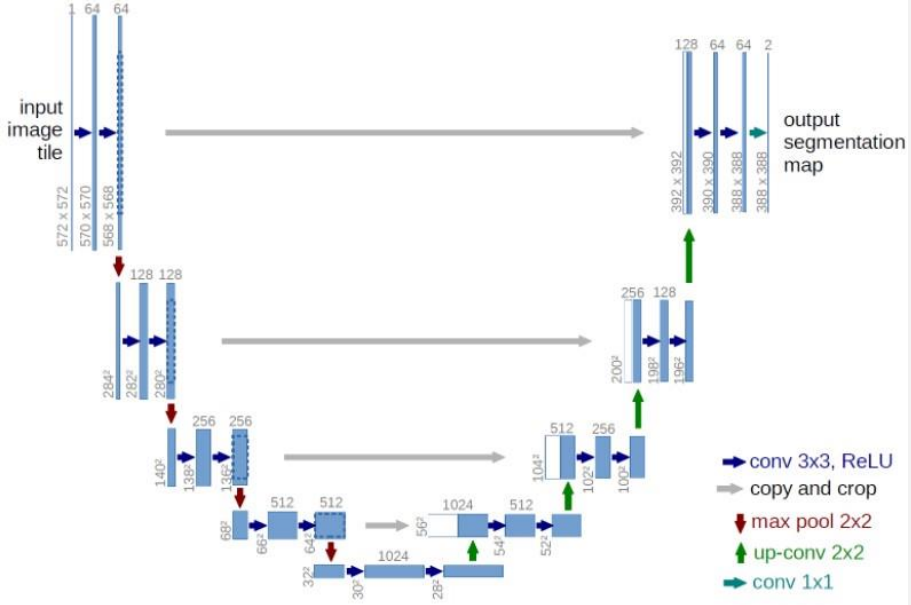


FPN v U-Net

Main difference: FPN has multiple prediction layers, one for each upsampling layer.

Both architectures have lateral connections, but Unet only crops and copies the features, while the FPN applies a 1x1 convolution layer before adding them.

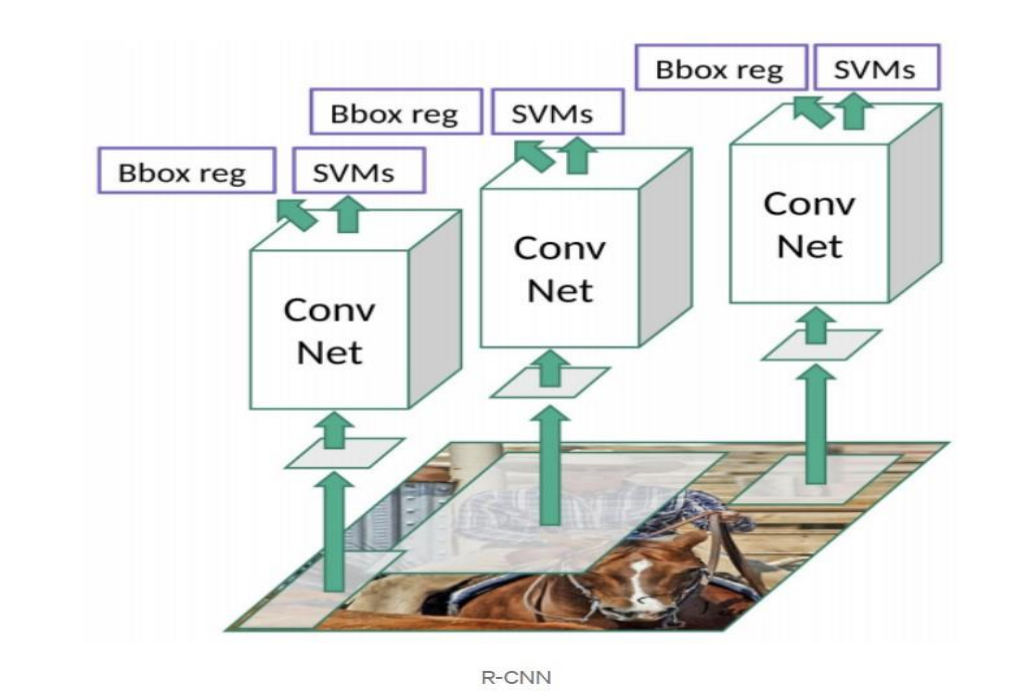




RCNN is two stage detector.

1st stage: Analyzes the input image and divides it into regions (bounding boxes). The algorithm that is used is called **selective search** and produces roughly 2000 regions. Then the images are getting warped for each region, until it gets a fixed size of pixels.

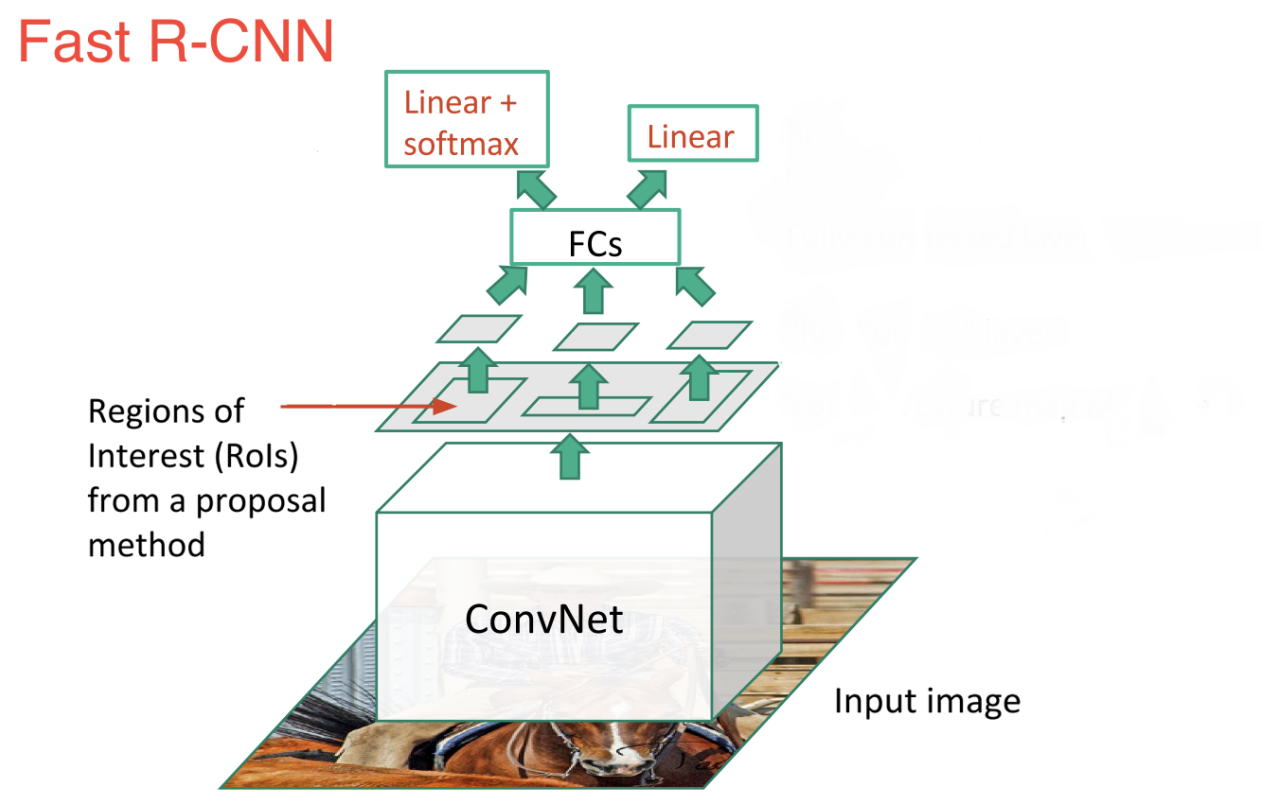
2nd stage: The new images are introduced into the CNN and the output dense layer consists of features that are fed into SVM. In addition, the model predicts four values, which are offset values to increase the precision of the bounding box.



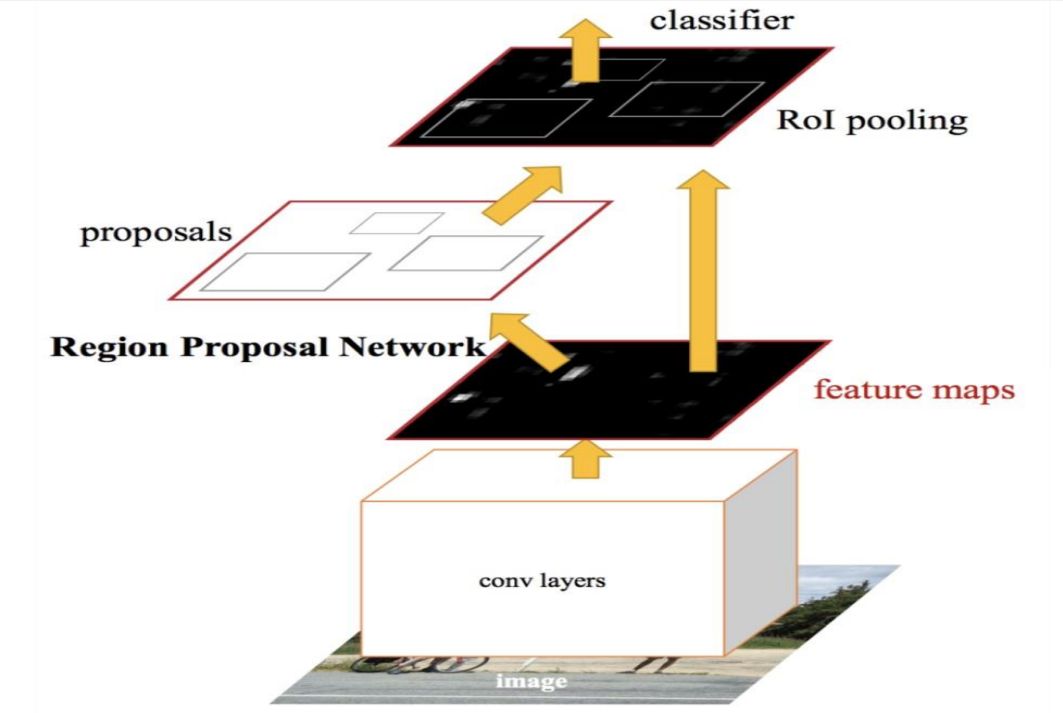
R-CNN

Fast RCNN an improvement of R-CNN.

First the image gets fed to a ConvNet. Then the selective search algorithm is used to generate RoIs from the feature map that was created. Afterwards the regions are warped and fed into FCs for classification and bounding box regression.

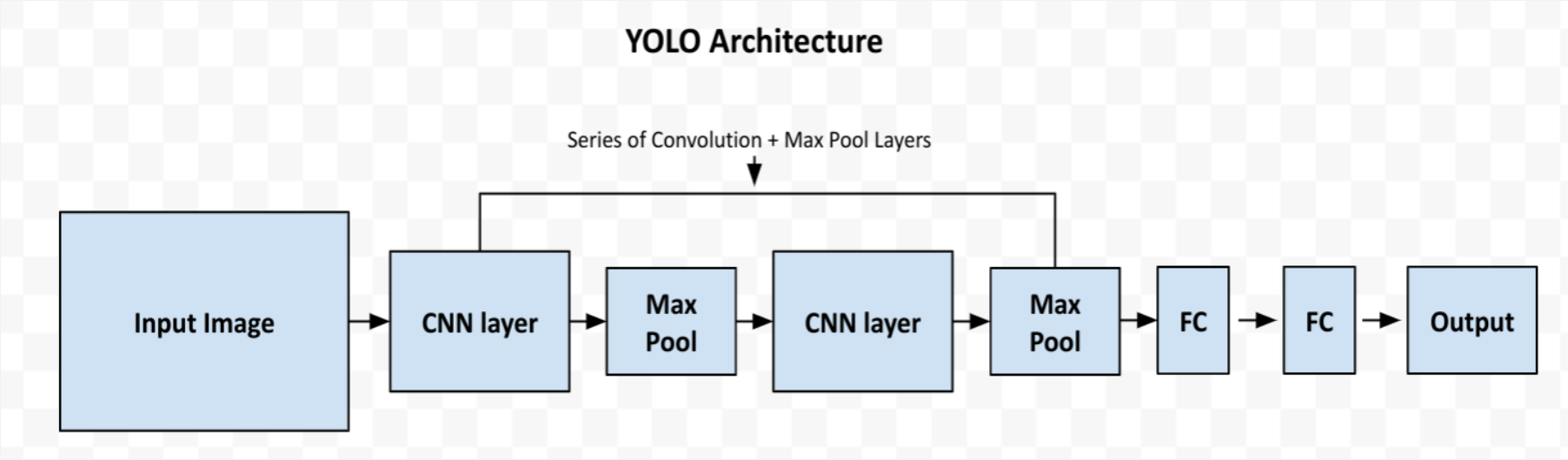


Faster RCNN. The input is fed to a ConvNet, then the feature map is fed in to another network called Region Proposal Network. RPN replaces selective search in Faster RCNN. The network gets trained and afterwards the proposals follow the same path as Fast RCNN.

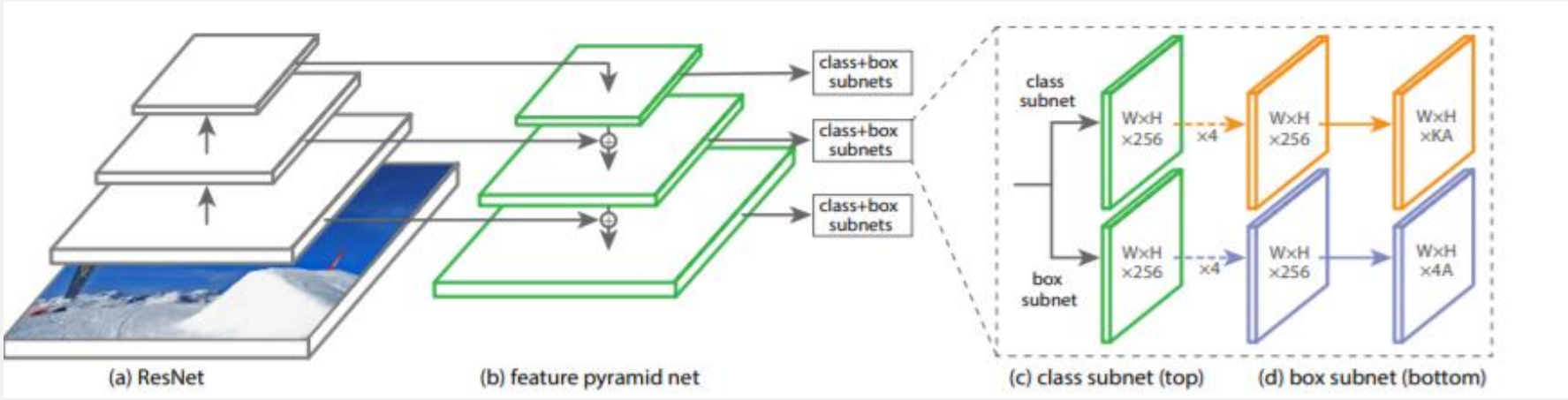


YOLO general Pipeline

General Architecture of YOLO (Fast, Tiny, v1, v2, etc).



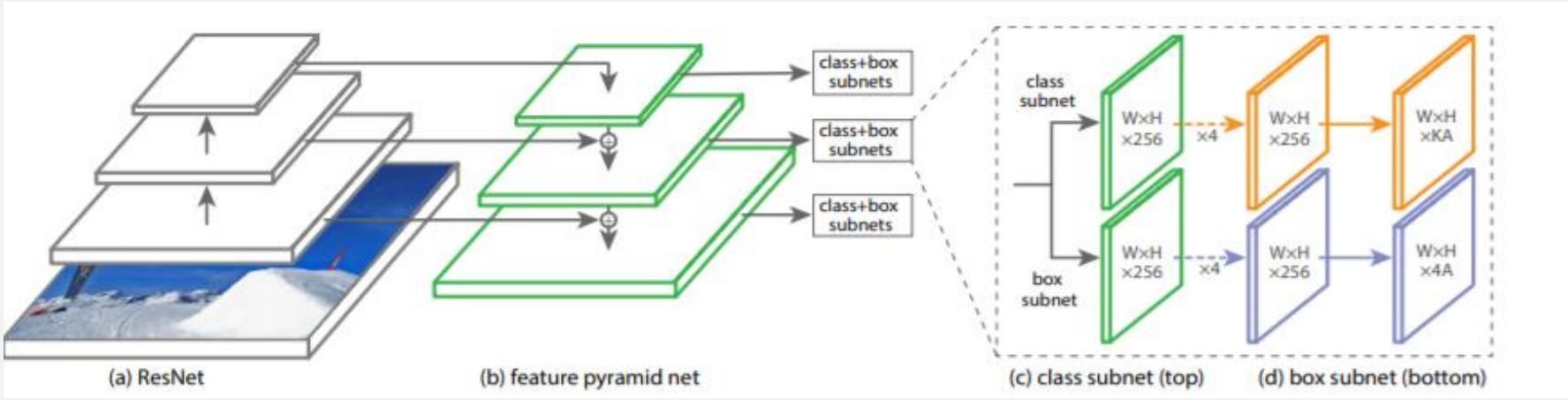
RetinaNet Architecture



The one stage RetinaNetwork architecture, uses a Feature Pyramid Network on top of a feedforward ResNet architecture as backbone, in order to generate a rich, multi-scale convolutional feature pyramid.

Two Fully Convolutional Network (top for classification, bottom for box regression) are attached to each FPN level and the parameters of this subnet are shared across all pyramid levels.

Backbone Network

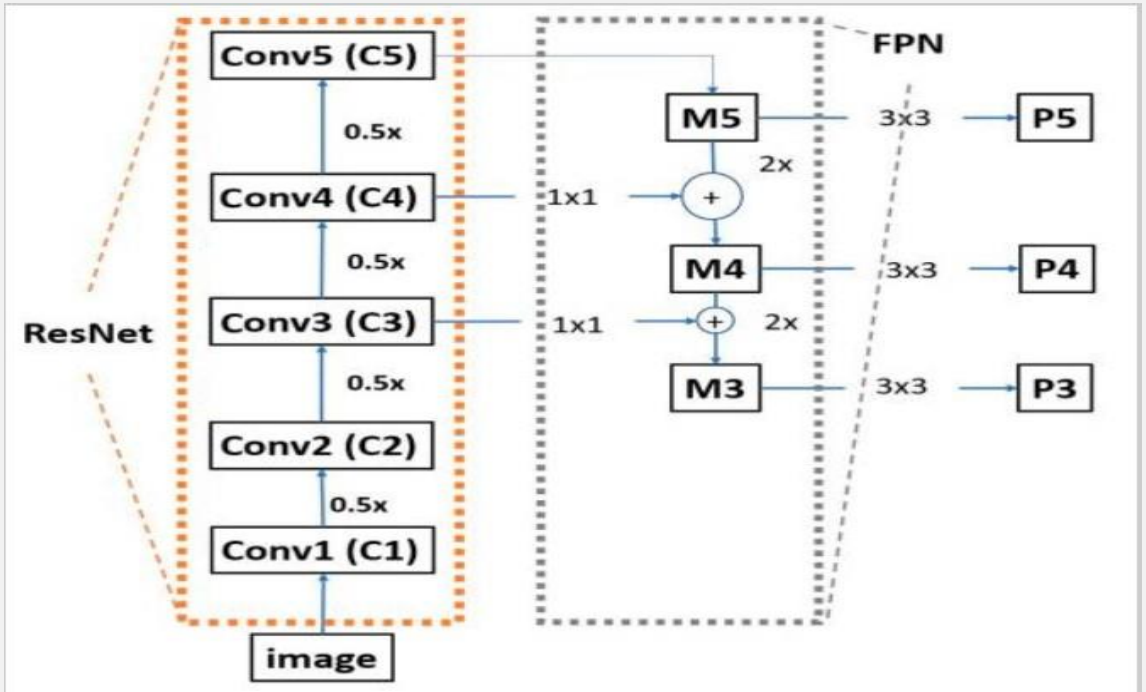


The backbone is necessary for computing a convolutional feature map over the entire image. It consists of an encoder (a residual network e.g. ResNet50) and a Feature Pyramid Network.

The original image is applied as an input to the encoder, which processes the image through convolutional kernels and generates deep features.

Our case a ResNet50 is used as encoder(bottom-up pathway).

ResNet50 - FPN

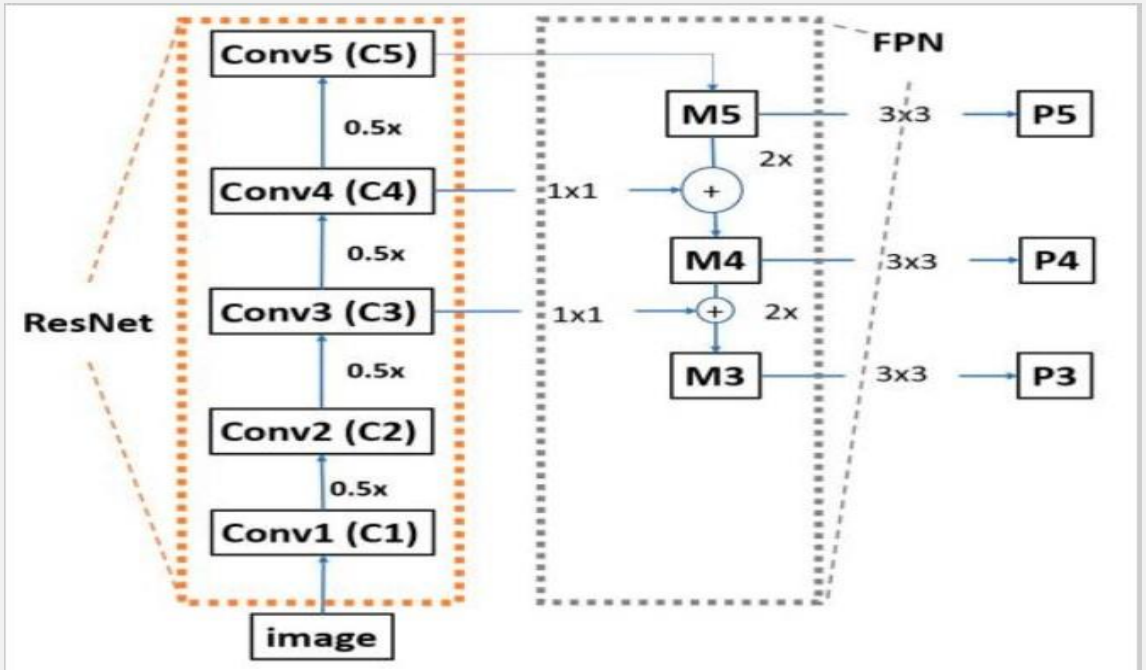


Composed of many convolutional layers.

The ResNet50 that RetinaNet utilizes does not contain the last average pooling layer, the fully connected layer and the softmax layer.

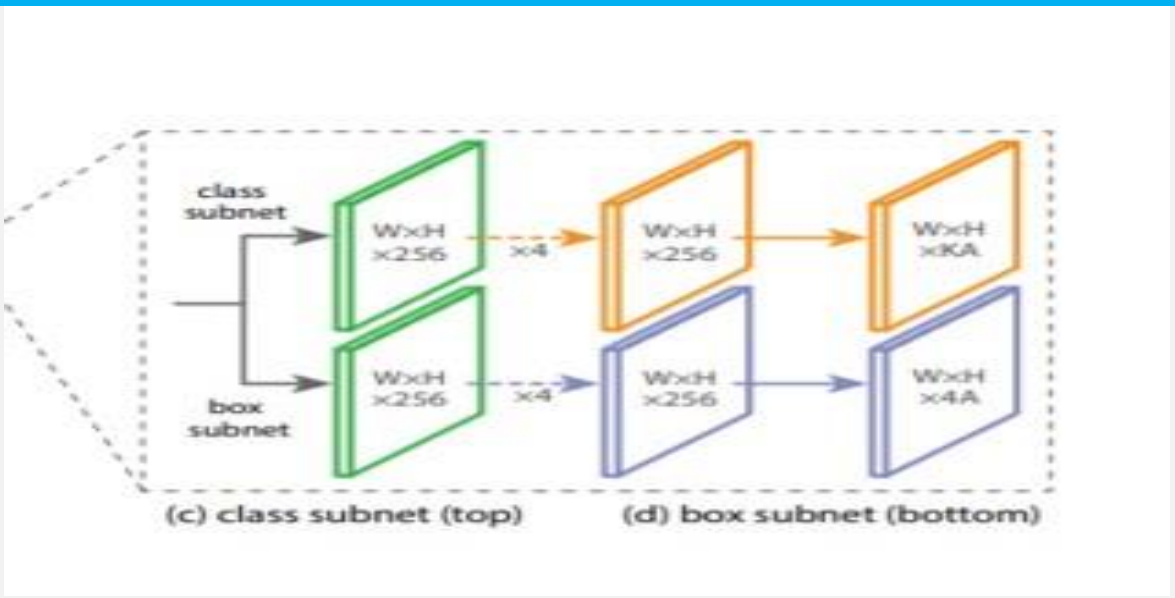
Moving from the bottom to the top, the spatial dimensions are reduced by half.

ResNet50 - FPN



For each feature map, the FPN up-samples the spatial resolution of the input feature map by a factor of two, and the up-sampled map is then merged with the corresponding bottom-up map, which undergoes a 1x1 convolution to reduce channel dimension by element-wise addition.

Classification

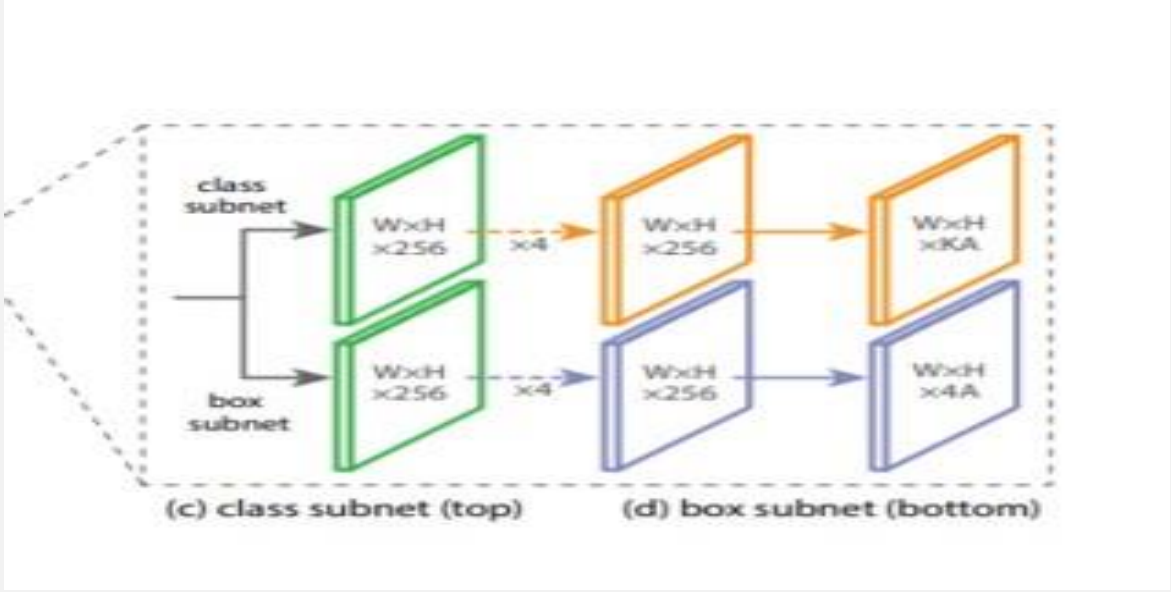


Predicts the probability of object presence at each spatial position for each of the A anchors and K object classes.

A small FCN attached to each FPN level. Parameters are shared across all pyramids levels.

Design: As input a feature map with C(256) channels from a pyramid lvl. Applies four 3x3 conv layers, each with C filters and followed by ReLU activations, followed by a 3x3 conv layer with KA filters. Last layer, sigmoid activation attached to output the KA predictions per spatial location.

Box Regression



Identical design, ends in 4A linear outputs per spatial location.

Predicts the relative offset between anchor and the ground truth box.

Anchor Boxes

A set of predefined bounding boxes of a certain height and width.

Defined to capture the scale and aspect ratio of specific object classes you want to detect and are typically chosen based on object sizes in the training datasets.

Anchor Boxes Parameters:

sizes = {16, 32, 64, 128, 256}

aspect ratios = {1:2,1:1,2:1,3:1}

(16x32, 16x16, 32x16, 64x16

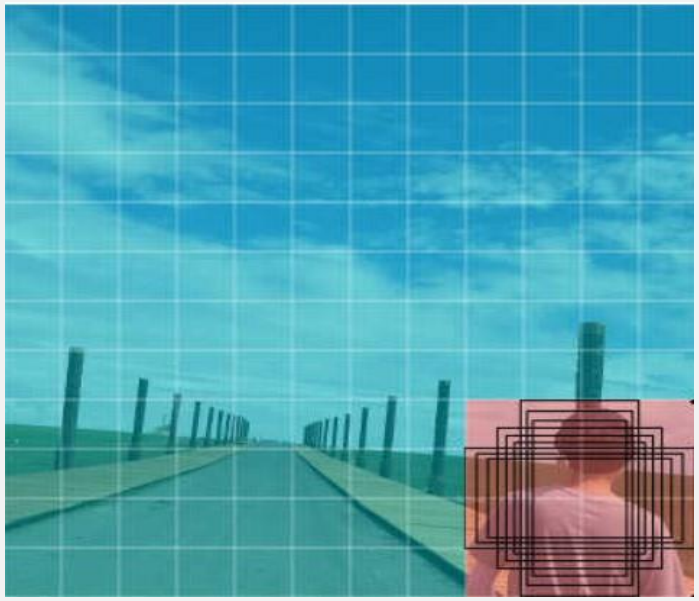
32x64, 32x32, 64x32, 128x32

64x128, 64x64, 128x64, 256x64

128x256, 128x128, 256x128

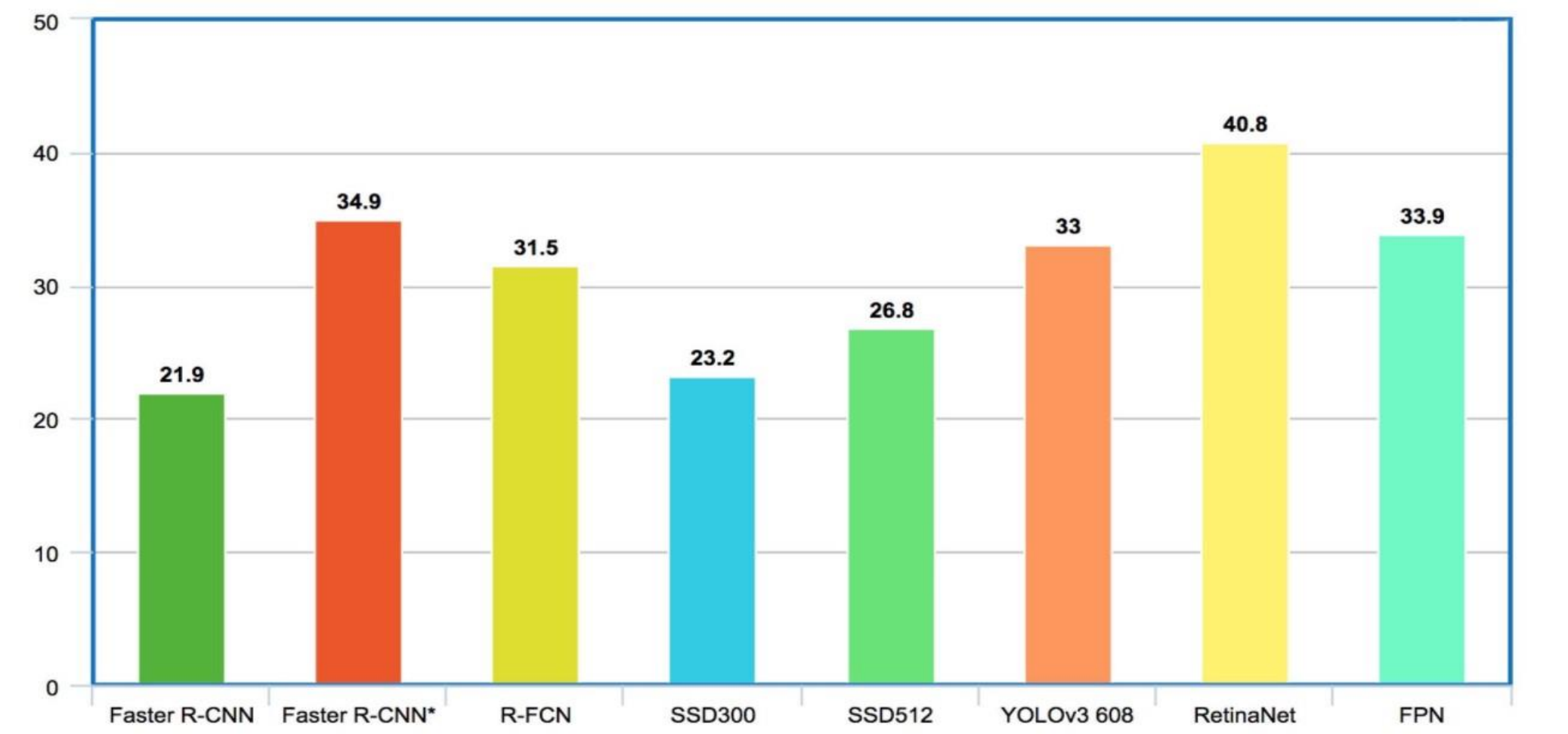
256x256)

Total 16 possible anchor boxes.



Non Maximum Suppression is applied to remove overlapping predictions and to get the box with the highest IoU.

Why Retina?



Mean Average Precision in the COCO dataset

Data description

Name: *Stanford Drone Dataset.*

Training annotations(35.765):

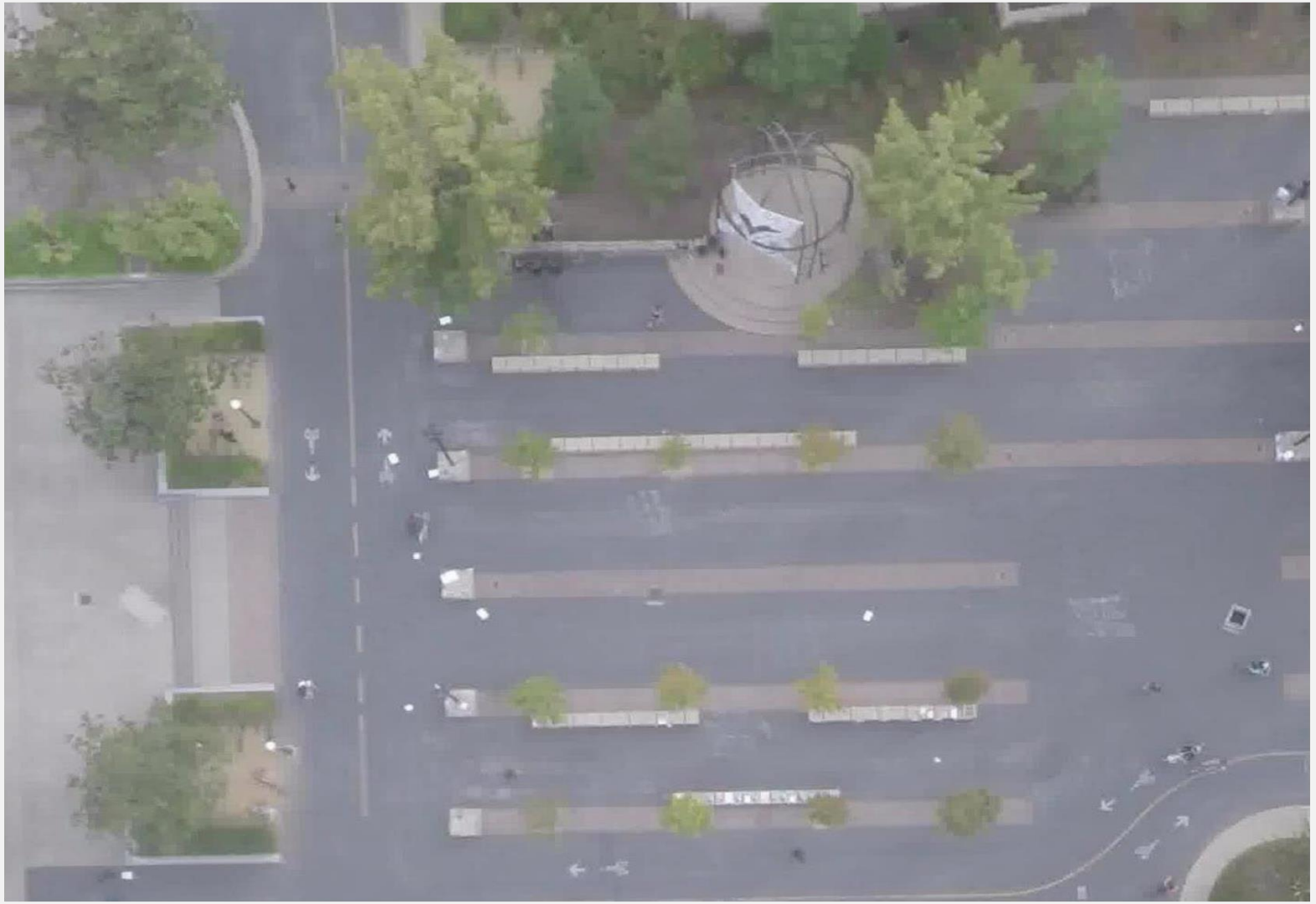
1) Person (22673)
2) Biker (11479)
3) Car (1512)
4) Bus (101)

Dataset Format: Pascal VOC (path/to/image/, x1, y1, x2, y2, class_name).

Test annotations(6.816):

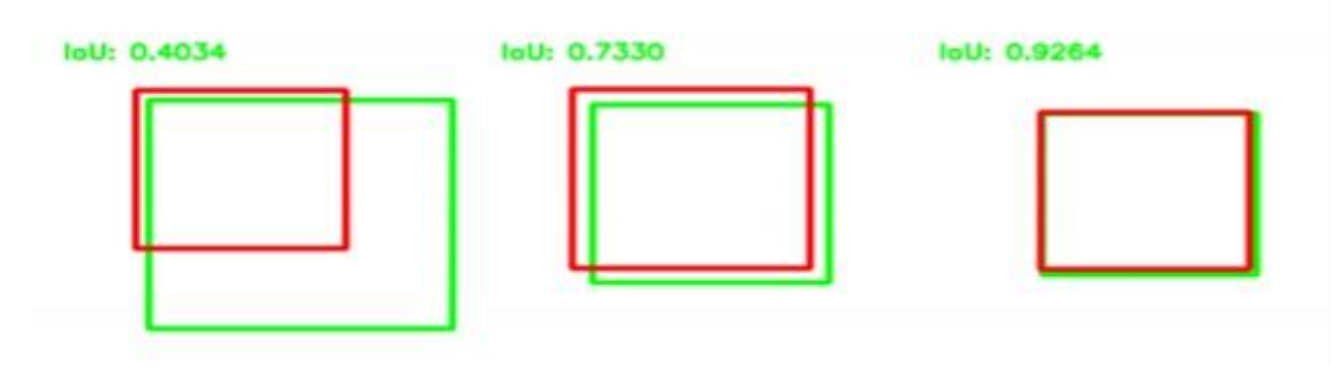
1) Person (5558)
2) Biker (1204)
3) Car (23)
4) Bus (31)

Data description



$$\boldsymbol{IoU} = \frac{area(B_p \cap B_{gt})}{area(B_p \cup B_{gt})}$$

True Positive: A correct detection with, IoU ≥ threshold.
False Positive: A wrong detection with, IoU < threshold.
False Negative: A ground truth that was not detected.
True Negative: Correct misdetections. It has no meaning in object detection.



$$\boldsymbol{Accuracy} = \frac{TP}{TP + FP + FN}$$

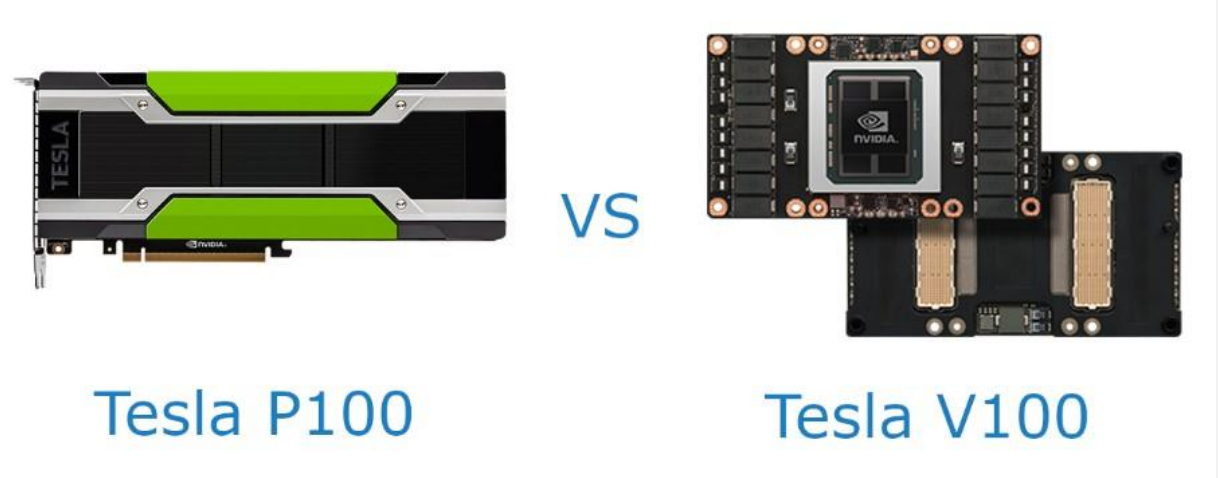
$$\boldsymbol{Precision} = \frac{TP}{TP + FP}$$

$$\boldsymbol{Recall} = \boldsymbol{Sensitivity} = \frac{TP}{TP + FN}$$

$$\boldsymbol{F1_{score}} = \frac{\boldsymbol{2 * precision * recall}}{\boldsymbol{precision + recall}}$$

$$\boldsymbol{mAP} = \frac{\boldsymbol{1}}{\boldsymbol{n}} \sum_{\boldsymbol{k=1}}^{\boldsymbol{k=n}} \boldsymbol{AP_k}$$

The training and testing took place in Google **Colab**. **Colab** allows anybody to write and execute arbitrary python code through the browser, and is especially well suited to machine learning, data analysis.



- NVIDIA Pascal architecture.
- 3.584 CUDA core.
- Performs at 4.7 TeraFLOPS.
- Memory: 16GB.
- Power Consumption: 250W.

Training Enviroment

- NVIDIA Volta architecture.
- 640 Tensor cores / 5.120 CUDA core.
- Performs at 7 TeraFLOPS.
- Memory: 32GB.
- Power Consumption: 250W.

Retina Training/Evaluation

Data Split (Training/Test). → Feed Training annotations to Network. → Training for 100 epochs on NVIDIA Tesla P100. → Test network.

Metric	Simple split (0.6 threshold)
Accuracy	0.800
Precision	0.847
Recall	0.934
F_1 score	0.889
mAP/wmAP	0.611/0.870

Person	**4829**	582	0	0
Biker	415	**738**	0	0
Bus	0	0	**22**	0
Car	0	0	8	**14**
None	643	122	1	6
	Person	**Biker**	**Bus**	**Car**

$$person_{precision} = \frac{4829}{4829 + 415} = 0.920$$

$$biker_{precision} = \frac{738}{738 + 582} = 0.559$$

$$car_{precision} = \frac{14}{14} = 1$$

$$bus_{precision} = \frac{22}{22 + 8} = 0.7333$$

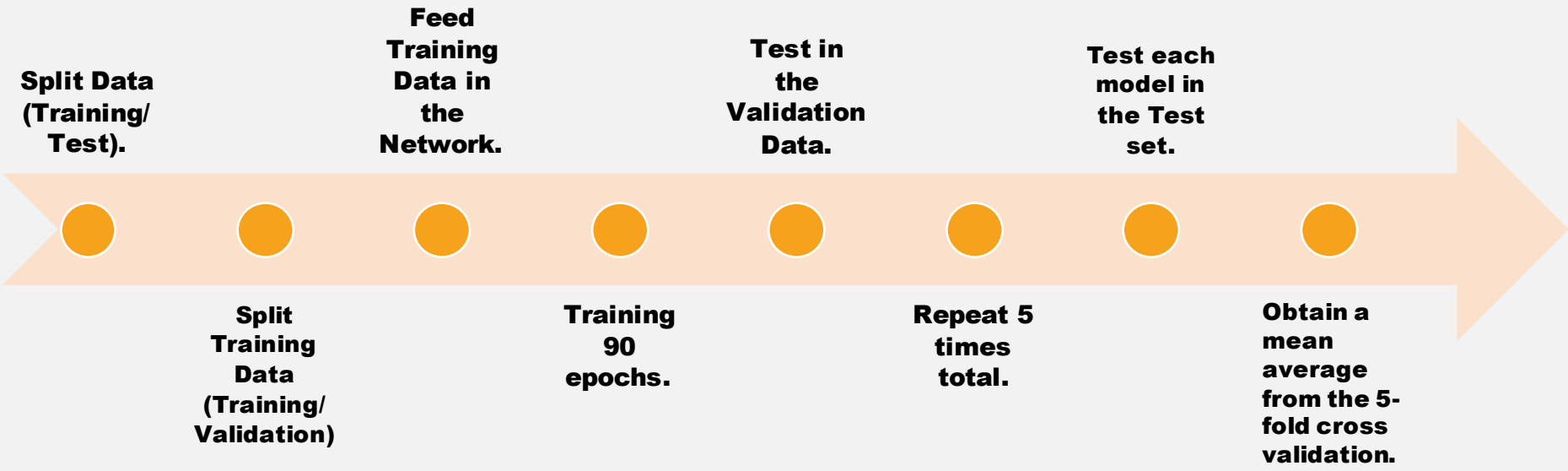
Retina Training/Evaluation

Metric	Simple split (0.7 threshold)
Accuracy	0.746
Precision	0.776
Recall	0.950
F_1 score	0.854
mAP/wmAP	0.596/0.847

Retina Evaluation

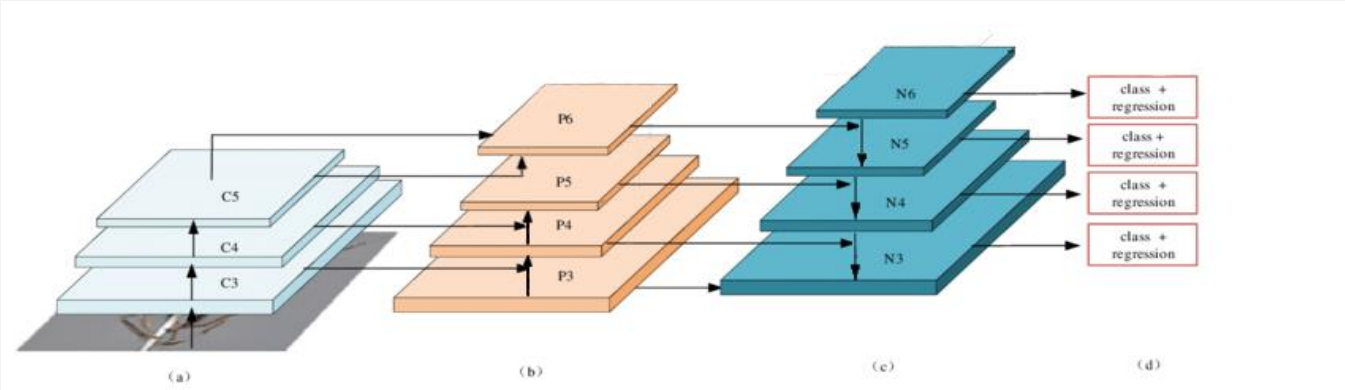
Person	**4661**	898	0	0
Biker	630	**661**	0	0
Bus	0	0	**25**	0
Car	0	0	6	**17**
None	207	67	1	6
	Person	**Biker**	**Bus**	**Car**

K-fold Cross Validation



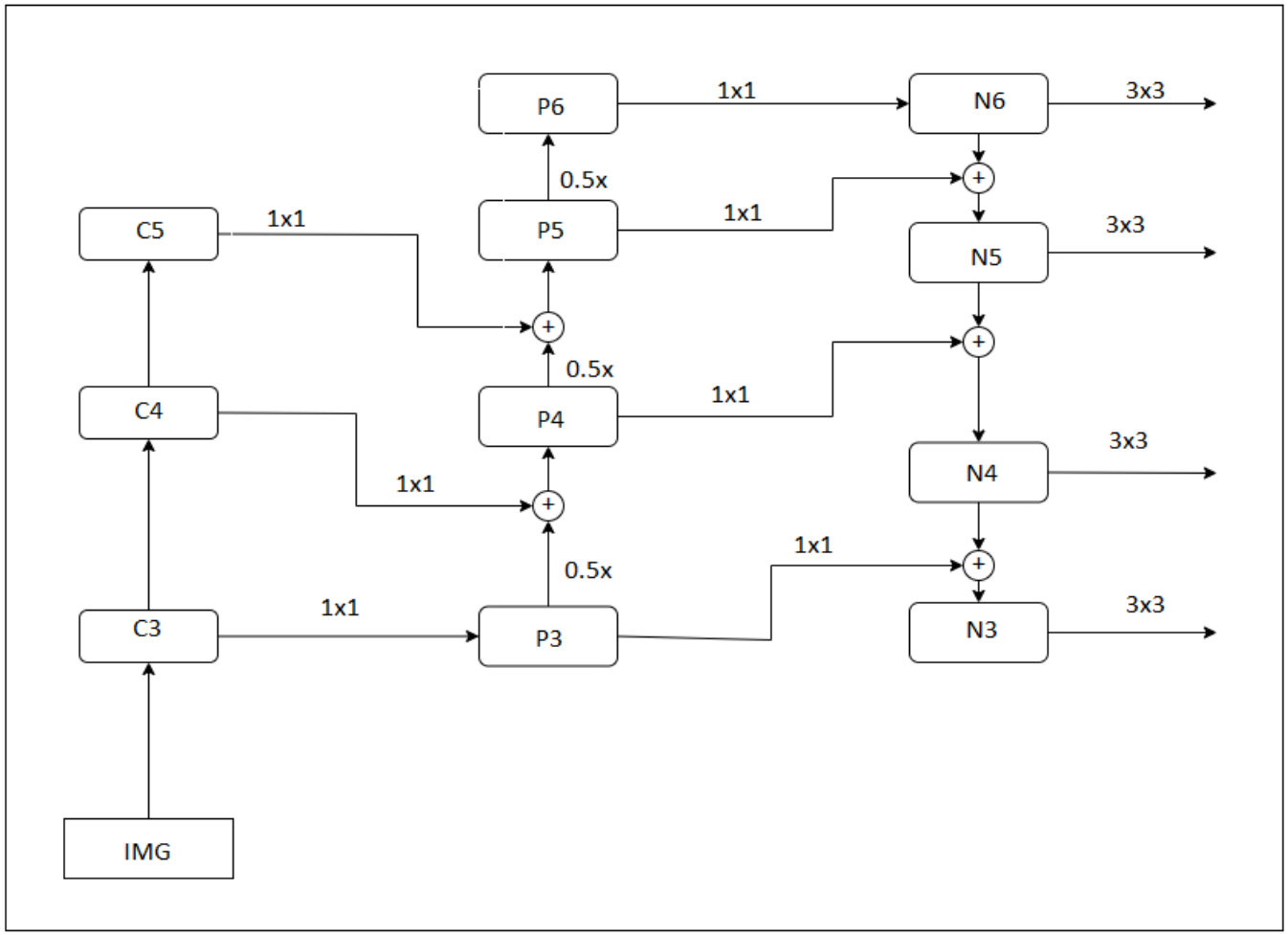
	1st fold	1st fold	2nd fold	2nd fold	3rd fold	3rd fold	4th fold	4th fold	5th fold	5th fold	mean	mean
Accuracy	0.795	0.789	0.792	0.748	0.584	0.657	0.689	0.742	0.821	0.796	0.736	0.746
Precision	0.865	0.857	0.866	0.853	0.818	0.849	0.841	0.859	0.853	0.852	0.848	0.854
Recall	0.908	0.899	0.902	0.900	0.670	0.655	0.792	0.788	0.955	0.943	0.845	0.837
F_1 score	0.886	0.888	0.883	0.886	0.737	0.711	0.816	0.817	0.897	0.901	0.843	0.840
	Val	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**

Modified RetinaNet Architecture



- Almost same architecture.
- Extra Convolution block.
- Extra convolution layers.

Modified RetinaNet Architecture



Modified RetinaNet Evaluation

Metric	Simple split (0.6 threshold)
Accuracy	0.849
Precision	0.887
Recall	0.943
F_1 score	0.914
mAP/wmAP	0.632/0.897

Modified Retina Evaluation

Person	**4972**	504	0	0
Biker	355	**842**	0	0
Bus	0	0	**22**	0
Car	0	0	8	**14**
None	397	192	1	6
	Person	**Biker**	**Bus**	**Car**

$$person_{precision} = \frac{4829}{4829 + 415} = 0.933$$

$$biker_{precision} = \frac{738}{738 + 582} = 0.625$$

$$car_{precision} = \frac{14}{14} = 1$$

$$bus_{precision} = \frac{22}{22 + 8} = 0.7333$$

Modified Retina Evaluation

	1st fold		2nd fold		3rd fold		4th fold		5th fold		Mean	
Accuracy	0.811	0.808	0.776	0.756	0.603	0.688	0.749	0.777	0.860	0.819	0.759	0.770
Precision	0.887	0.846	0.844	0.877	0.834	0.854	0.866	0.898	0.901	0.875	0.866	0.870
Recall	0.912	0.901	0.909	0.920	0.705	0.703	0.819	0.800	0.963	0.966	0.861	0.858
F_1 score	0.899	0.873	0.875	0.897	0.764	0.771	0.841	0.846	0.930	0.918	0.861	0.861
	Val	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**	**Val**	**Test**

Modified v Original Retina

- Significant more time to train the model.
- Accuracy 6% higher than original.
- Precision 4% higher than the original.
- Recall 1% higher than the original.
- F1 score 2% higher than the original.
- Mean Average Precision and weighted mAP higher than the original.

Comparison to other models

Model	mAP(%)
FasterRCNN	59.60
SSD	64.31
YOLOv3	57.42
RetinaNet	61.10
Modified Retina	63.27

Conclusions

- Modified Retina outperformed most of the detectors.
- Single Shot Detector performed slightly better in terms of mAP.
- Modified architecture performed better than baseline model.
- Ideal for detection of small objects that require high accuracy score.

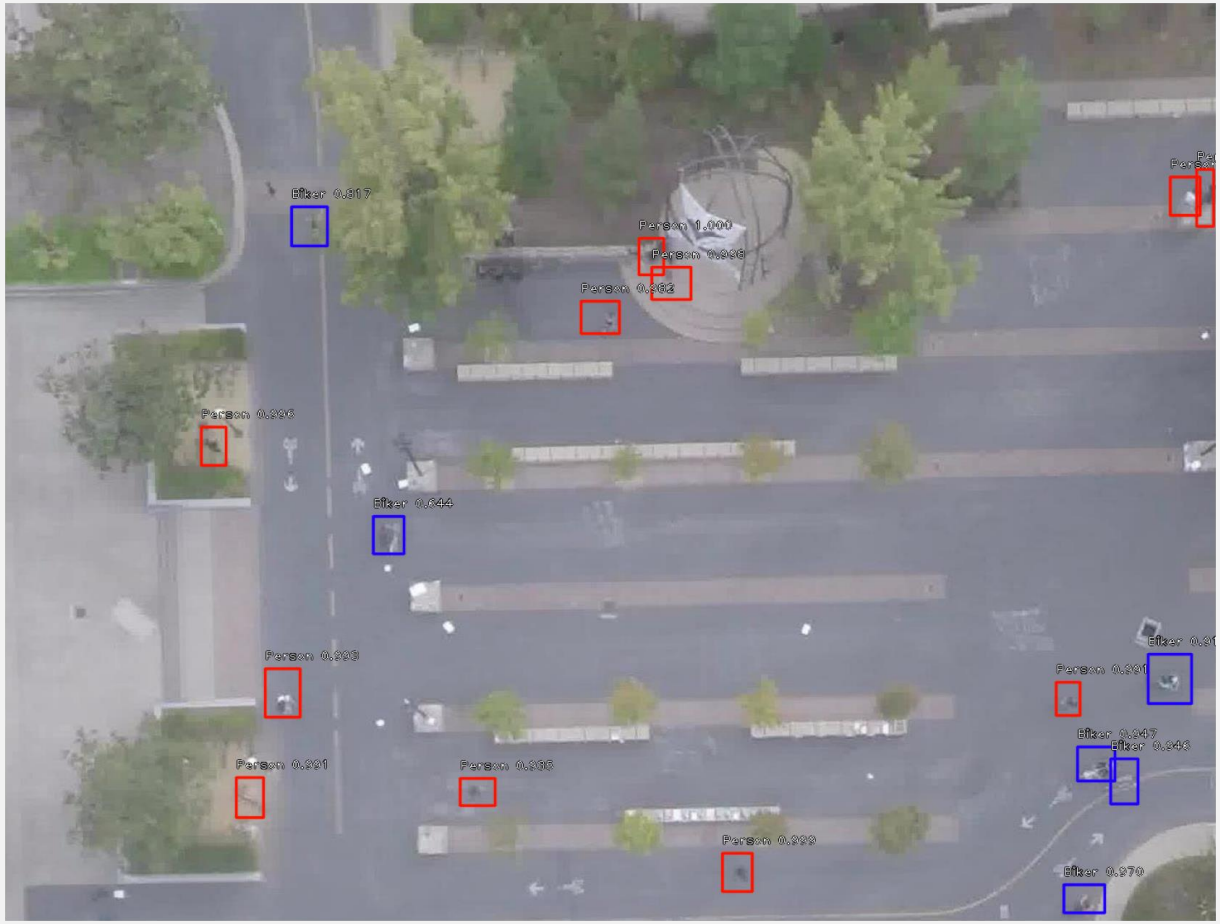
Limitations:

- Required high computational power.
- Limited Access to Colab resources.
- Long hours of training.

Future work:

- Train model with other loss functions.
- Better fine tuning of the network.
- Data fusion to enlarge the dataset.

Detections


Biker 0.817
Person 1.000
Person 0.998
Person 0.982
Person 0.946
Biker 0.844
Person 0.993
Person 0.991
Biker 0.947
Biker 0.948
Person 0.991
Person 0.986
Person 0.999
Biker 0.979