# Detecting Objects in Traffic Scenes

**Anıl Öztürk**
504181504

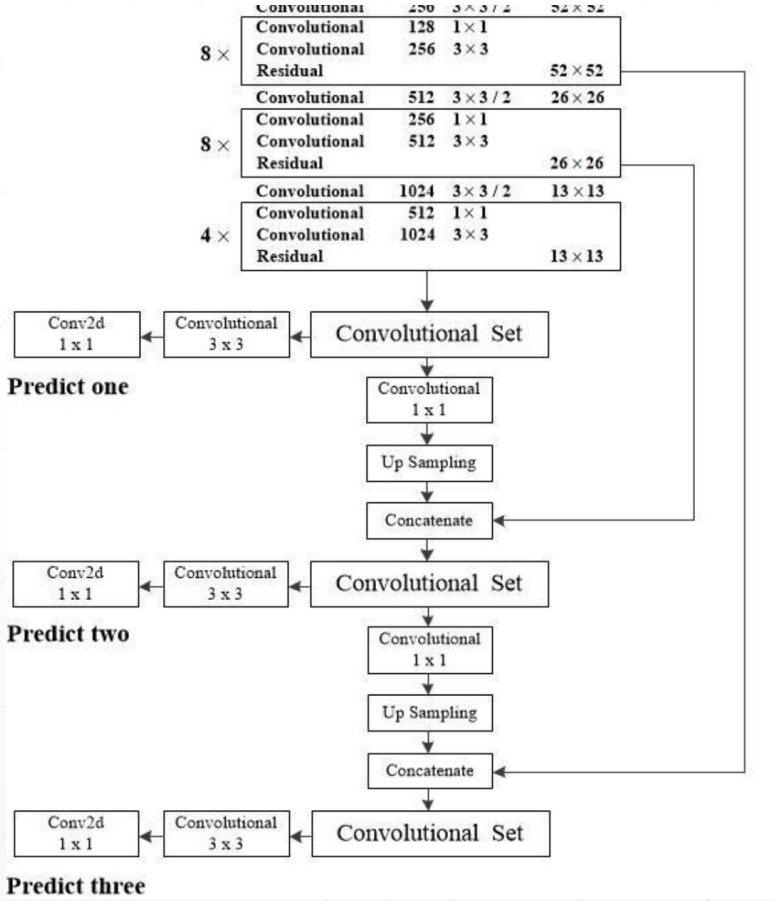
**Mustafa Günel**
511191124

**Onur Gürışık**
504191126

**Sertaç İkizoğlu**
504191128

# Meaning of each Prediction Layer

- **Predict Level 1 :** Large Objects
- **Predict Level 2 :** Medium Objects
- **Predict Level 3 :** Small Objects



# Modifications on Tiny YOLOv3

- Added **third detection layer** for **detecting small objects** and **increase mAP**
- Since third layer comes with a **computational load**, we **reduced the depth** of last two convolutional layers' of first detection phase
- To neutralize this reduction, we added extra **3x3** convolutional with the depth of 512 to the end of the feature extraction part and made the final 1024-depth convolutional **1x1**.

# Processing the Datasets

- Eatron data has **outlier** boxes (unlabeled, out of the frame etc.)
- BDD100K has **more** classes than we need
- Our train scripts want boxes with ***(cx, cy, w, h)*** format, both datasets was not in that format.

# Splitting the Datasets

- We **splitted** Eatron data with the ratios 70%, 30% for train and validation, respectively.
- BDD100K data were **already splitted** for train and validation with the sizes of 100K and 10K, respectively.

# Training Phases

We decided to execute **two** different training phase;

- **Phase One:** We train our model with BDD100K train set and validate it with BDD100K validation + Eatron validation set
- **Phase Two:** We train our model with BDD100K train + Eatron train sets and validate it with Eatron validation set with **lesser epochs** **(to fine-tune)**

# Setting Prior Boxes

The YOLO algorithm is based on regressing a set of pre-defined box sizes with a parametrised weights. These pre-defined sizes are called box priors or anchors. You should set them manually when you train on a custom dataset with your special classes.

We calculated **9** most frequent priors (3 for big sized object detection layer, 3 for small and 3 for medium sized object detection layers) with runnning **k-means clustering** on BDD100K training set.

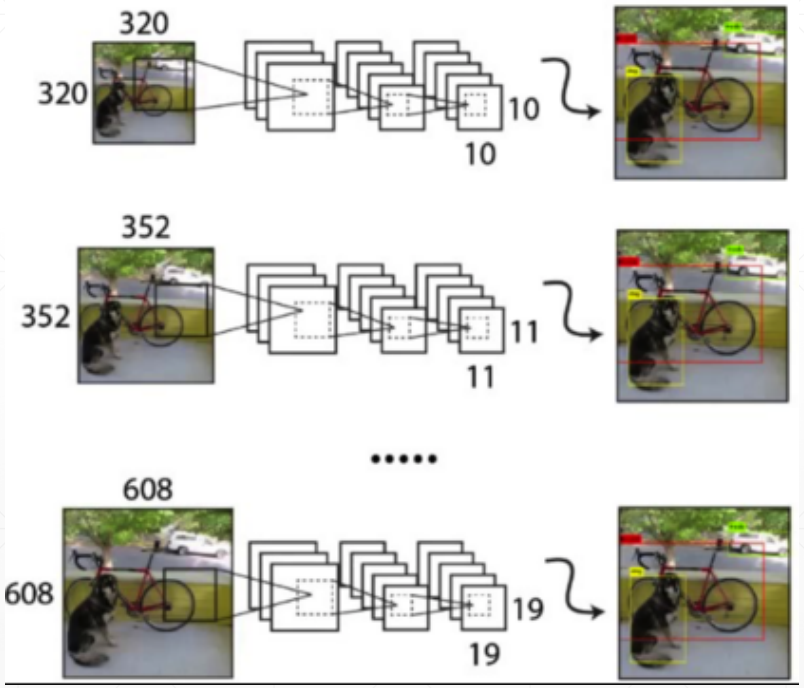
# Data Augmentation

We performed data-augmentation when getting batches dynamically for our training phase.

- **+/- 10%** vertical and horizontal translation
- **+/- 5%** rotation
- **+/- 10%** scale
- **50%** probability horizontal-reflection

# Multi-Scale Approach

We have used multi-scale approach while feeding images into our network in the training phase. We set the scale-up and scale-down multiplier as **1.5**.



# Weighting Images

- We weight the images based on the loss on each class of the model while training.
- If a class' classification loss is higher than the others, the images that contains objects from that class more likely to be shown to our network in the process.
- This will bring faster-converging model as result.

# Conclusion

- We compared our custom model with YOLOv3 that trained on the COCO set and YOLOv3 Tiny that trained on the BDD100K set.
- All of the YOLOs run at **608x608** input resolution. We calculated their relative score with respect to each other, like in this competition.
- Our model has the highest score with a very large gap. We calculated these results on **Eatron Validation Set** with **NVIDIA GTX1080TI** and **Intel i9 9900K**.

| Model | mAP | FPS | Parameter Count | Relative Score |
|---|---|---|---|---|
| YOLOv3 (COCO) | 0.499 | 39.11 | 140.69 Bn | 0.558 |
| Tiny YOLOv3 (BDD100K) | 0.299 | **171.66** | **5.56 Bn** | 0.765 |
| Our YOLO | **0.563** | 160.57 | 5.86 Bn | **0.968** |

# Cons of Our Model

- Since we have made our big-object detection layer more weak in the terms of convolutional depth, we have an issue with the **bounding boxes of large vehicles**.
- We set the values (kernel-sizes, depths) of third detection layer (a.k.a. Small object detector) with thumb's rule. We are having **false-positives** on small objects on inference. We should've apply NAS-like method.

# Thank You

any questions?