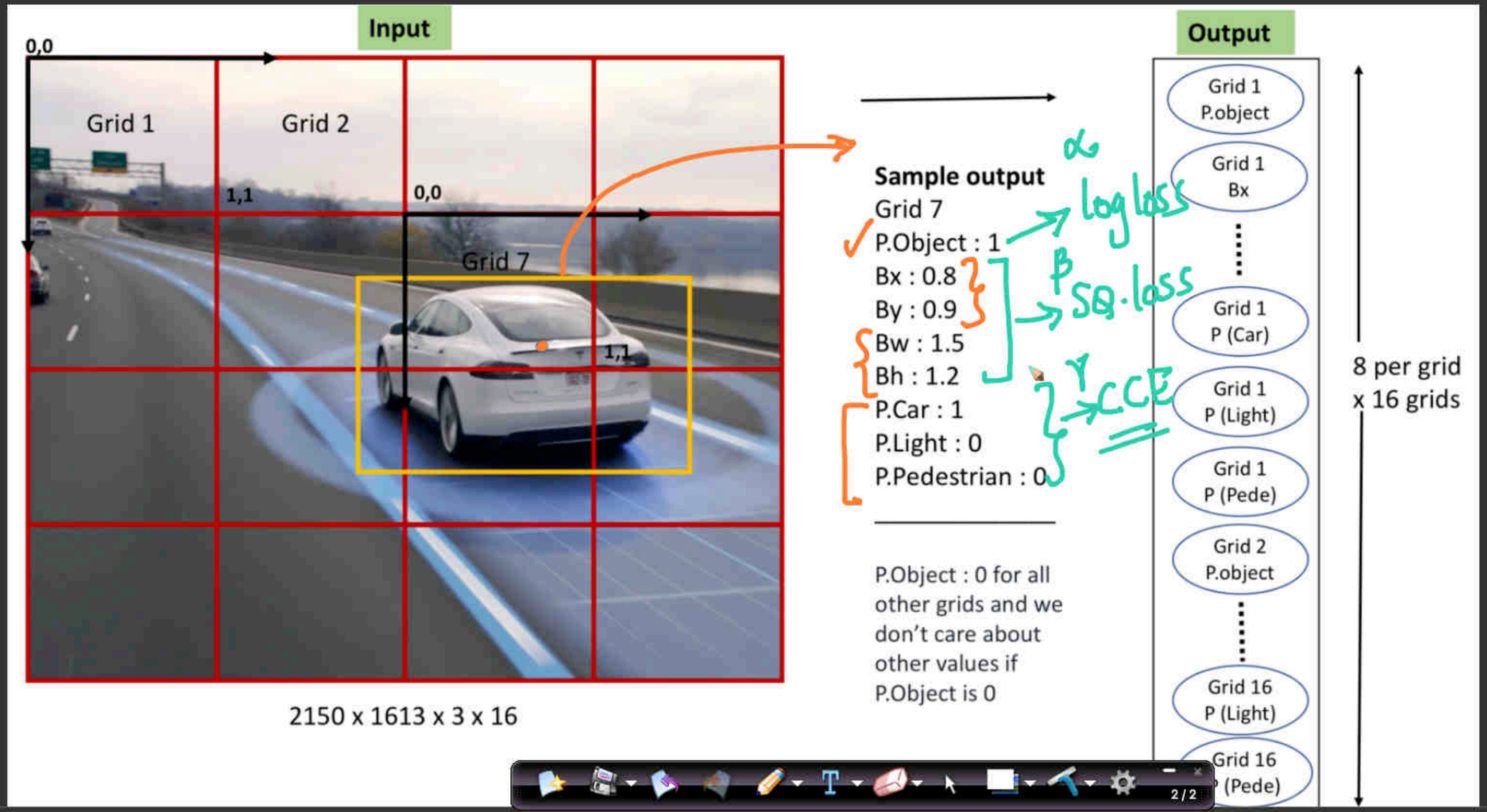
Agenda:

1. YOLO, YOLO V3
2. Overview of RetinaNet (Post read)
3. Segmentation & metrics
4. FCN + Transposed conv
5. U-net + Upsampling2D
6. Dilated Conv...


Input
0,0
Grid 1
Grid 2
1,1
0,0
Grid 7
1,1
2150 x 1613 x 3 x 16
Sample output
Grid 7
P.Object : 1
Bx : 0.8
By : 0.9
Bw : 1.5
Bh : 1.2
P.Car : 1
P.Light : 0
P.Pedestrian : 0
P.Object : 0 for all other grids and we don't care about other values if P.Object is 0
α
log loss
β
SQ. loss
γ
CCE
Output
Grid 1 P.object
Grid 1 Bx
Grid 1 P (Car)
Grid 1 P (Light)
Grid 1 P (Pede)
Grid 2 P.object
Grid 16 P (Light)
Grid 16 (Pede)
8 per grid x 16 grids

Network Architecture

The base model is similar to GoogLeNet with inception module replaced by 1x1 and 3x3 conv layers. The final prediction of shape $S \times S \times (5B + K)$ is produced by two fully connected layers over the whole conv feature map.

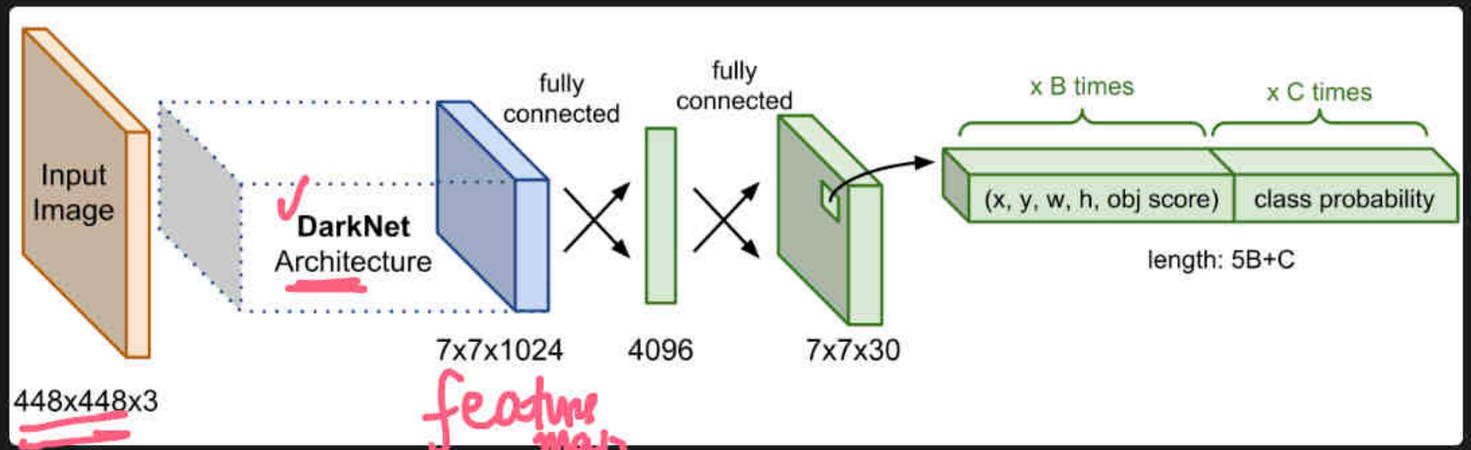


Fig. 2. The network architecture of YOLO.

Loss Function

The loss consists of two parts, the *localization loss* for bounding box offset prediction and the *classification loss* for conditional class probabilities. Both parts are computed as the sum of squared errors. Two scale parameters are used to control how much we want to increase the loss from bounding box coordinate predictions (λ_{coord}) and how much we want to decrease the loss of confidence score predictions for boxes without objects (λ_{noobj}). Down-weighting the loss contributed by back o instance. In the paper, the model sets λ = 5 and λ = 0.5

Network Architecture

The base model is similar to GoogLeNet with inception module replaced by 1x1 and 3x3 conv layers. The final prediction of shape $S \times S \times (5B + K)$ is produced by two fully connected layers over the whole conv feature map.

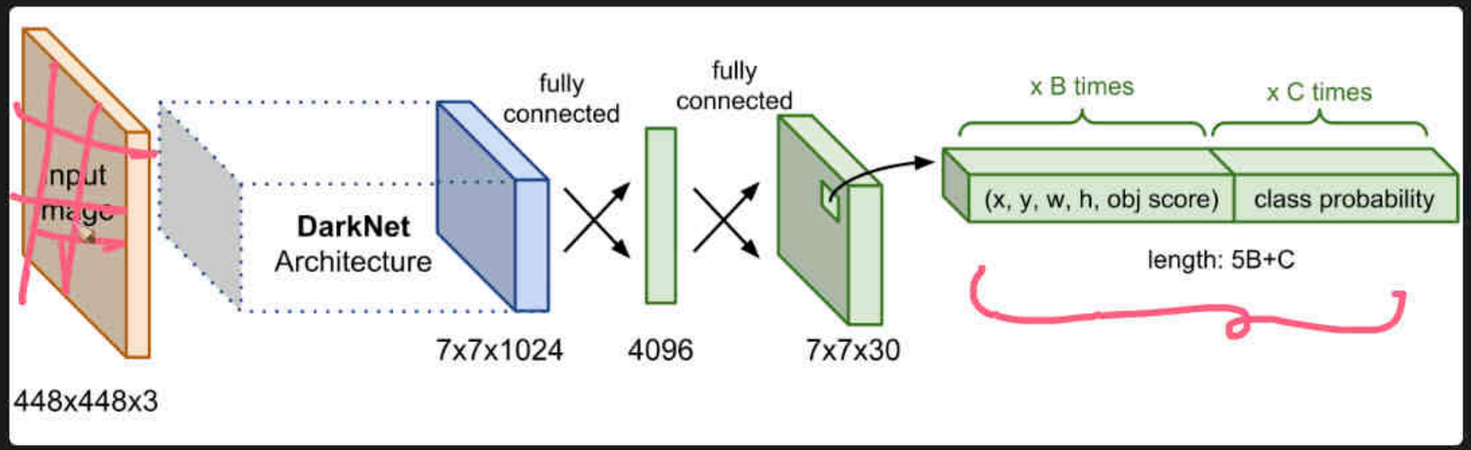


Fig. 2. The network architecture of YOLO.

Loss Function

The loss consists of two parts, the *localization loss* for bounding box offset prediction and the *classification loss* for conditional class probabilities. Both parts are computed as the sum of squared errors. Two scale parameters are used to control how much we want to increase the loss from bounding box coordinate predictions (λ_{coord}) and how much we want to decrease the loss of confidence score predictions for boxes without objects (λ_{noobj}). Down-weighting the loss contributed by back o instance. In the paper, the model sets λ = 5 and λ = 0.5

Network Architecture

The base model is similar to GoogLeNet with inception module replaced by 1x1 and 3x3 conv layers. The final prediction of shape $S \times S \times (5B + K)$ is produced by two fully connected layers over the whole conv feature map.

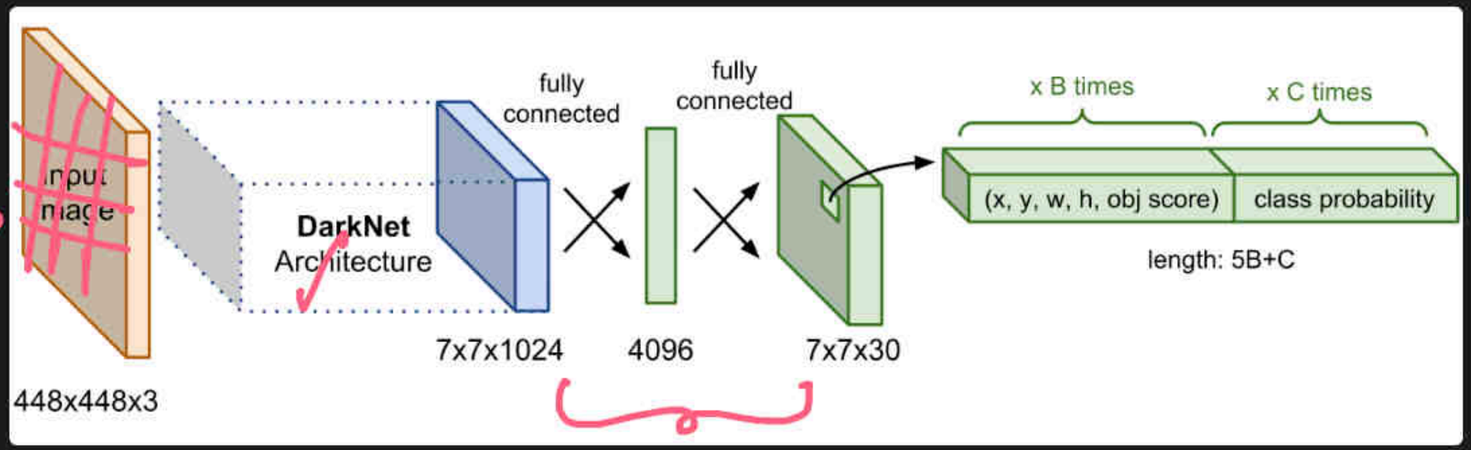


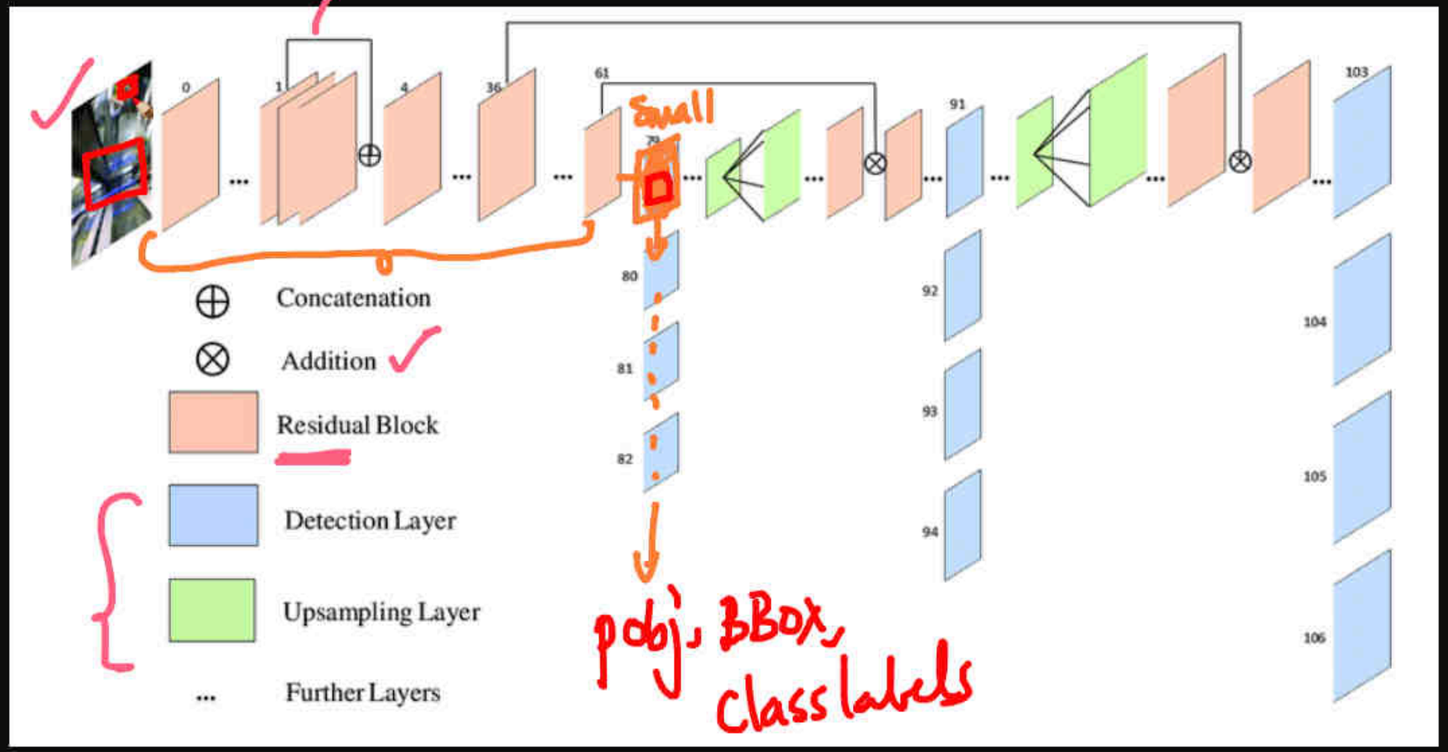
Fig. 2. The network architecture of YOLO.

Loss Function

The loss consists of two parts, the *localization loss* for bounding box offset prediction and the *classification loss* for conditional class probabilities. Both parts are computed as the sum of squared errors. Two scale parameters are used to control how much we want to increase the loss from bounding box coordinate predictions (λ_{coord}) and how much we want to decrease the loss of confidence score predictions for boxes without objects (λ_{noobj}). Down-weighting the loss contributed by back o instance. In the paper, the model sets λ = 5 and λ = 0.5

YoloV3: multi stage & multiscale detection

skip



Network Architecture

The base model is similar to GoogLeNet with inception module replaced by 1x1 and 3x3 conv layers. The final prediction of shape $S \times S \times (5B + K)$ is produced by two fully connected layers over the whole conv feature map.

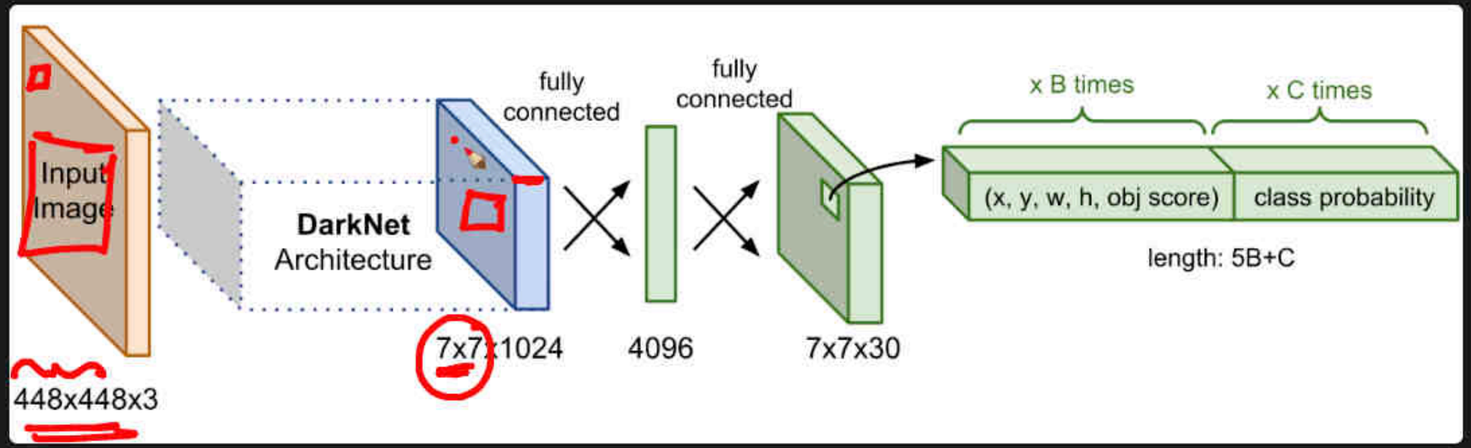
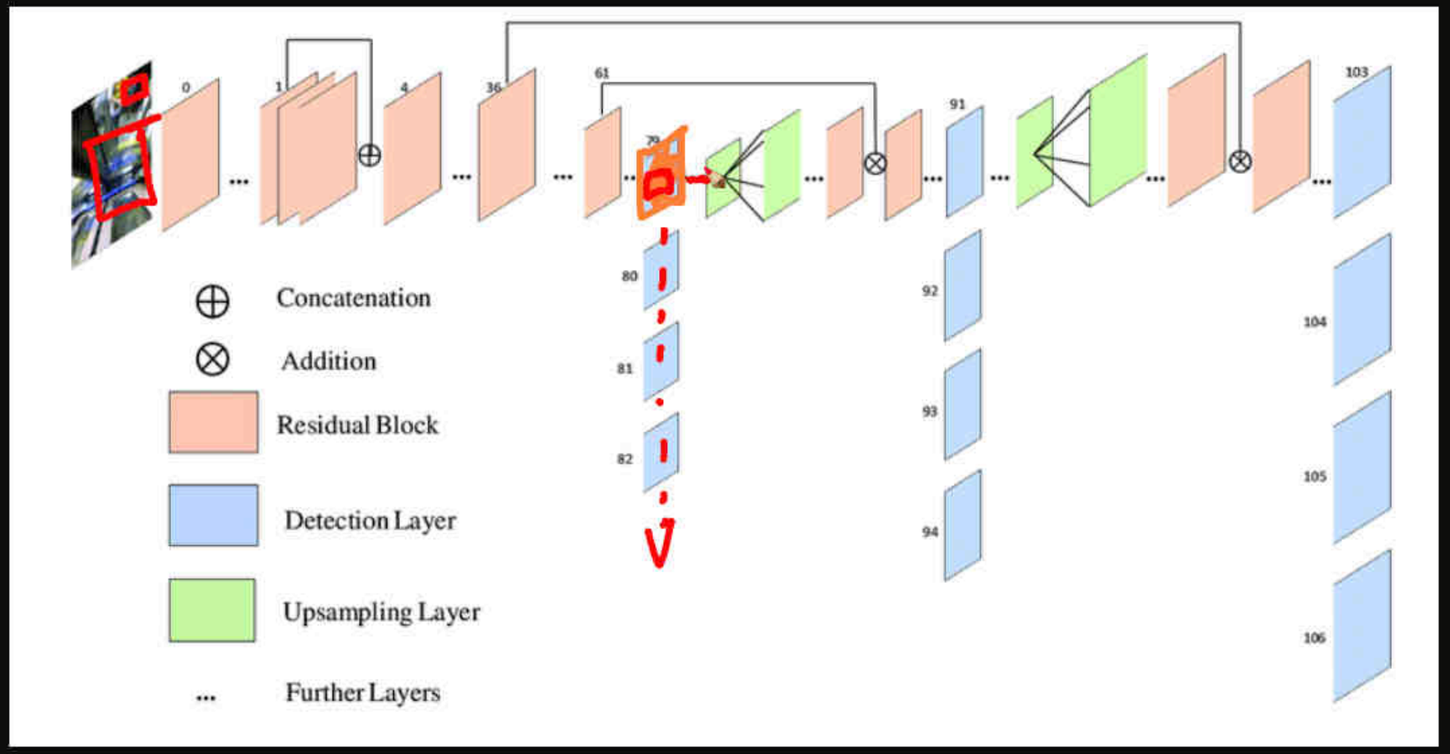


Fig. 2. The network architecture of YOLO.

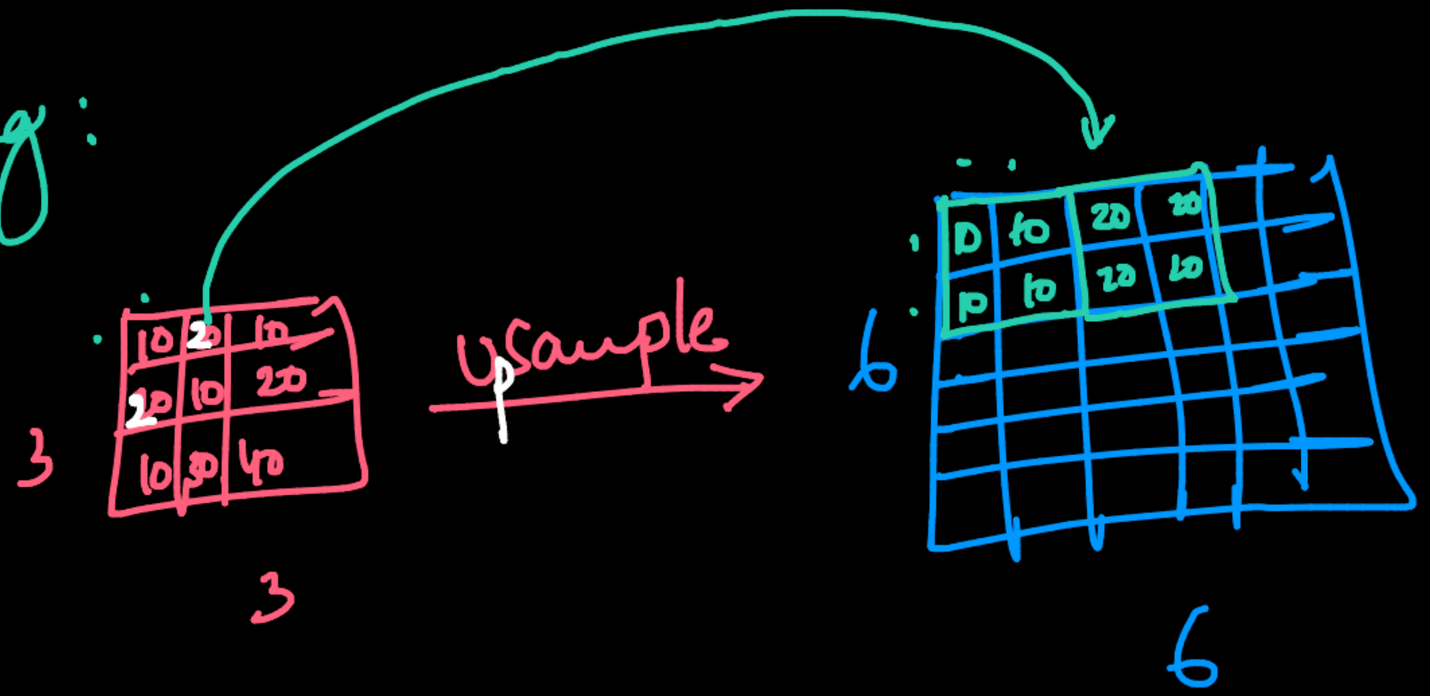
Loss Function

The loss consists of two parts, the *localization loss* for bounding box offset prediction and the *classification loss* for conditional class probabilities. Both parts are computed as the sum of squared errors. Two scale parameters are used to control how much we want to increase the loss from bounding box coordinate predictions (λ_{coord}) and how much we want to decrease the loss of confidence score pr[illegible] (λ[illegible]). Down-weighting the loss contributed by back[illegible] instance.

multi-scale


0
1
4
36
61
79
91
103
80
81
82
92
93
94
104
105
106
Concatenation
Addition
Residual Block
Detection Layer
Upsampling Layer
...
Further Layers

Usampling :

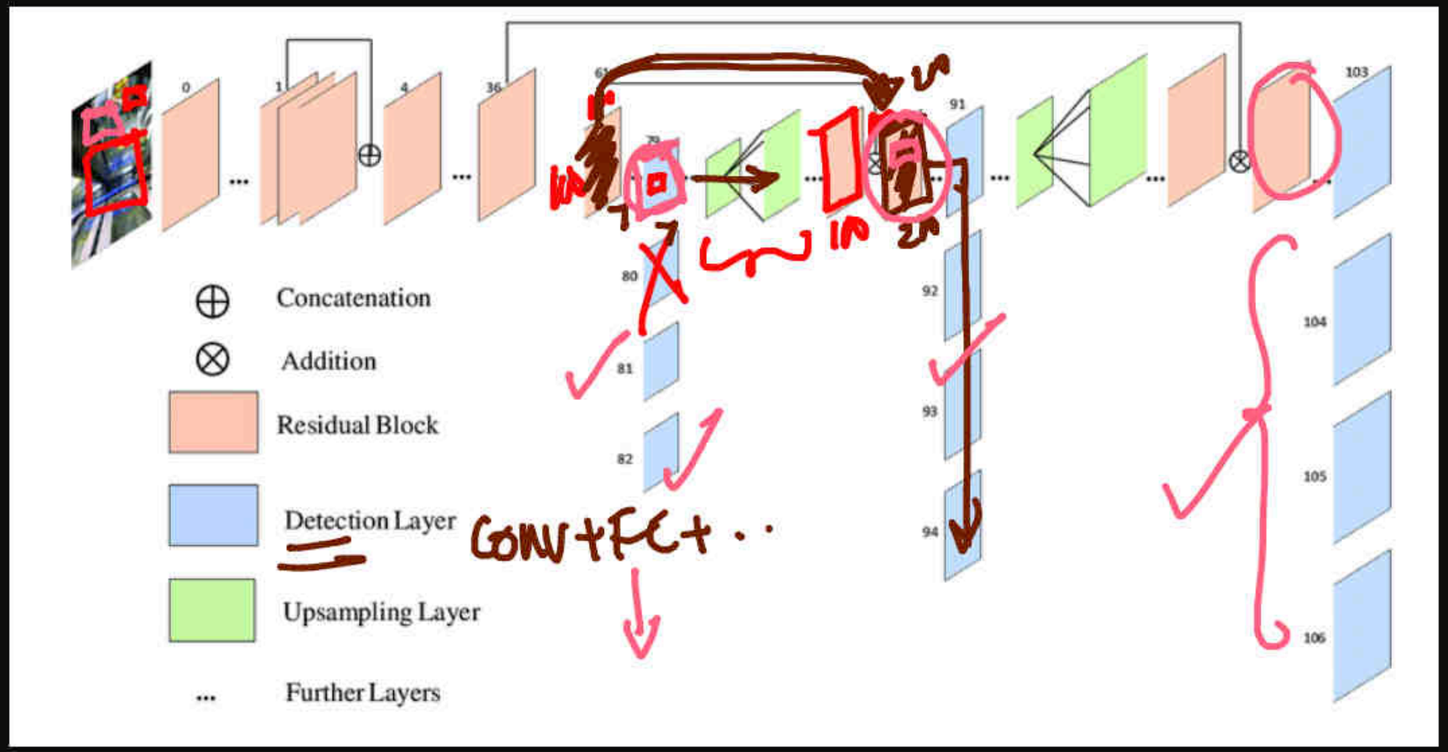


128x128 —US→ 256x256

V3

3 scale

3 stage



Let's go back to our problem statement and see how we can using Pre-Trained YoloV5 to solve it:

Why YOLO V5:

- We are going to use pre-trained Yolo V5 models for our use case since they are trained using COCO-Dataset (https://cocodataset.org/#home) and already contain the Target-Classes which we want to Detect:

```
- 'Car' , 'motorbike','aeroplane', 'bus',  'truck'
- Traffic light
- Person
- so on..
```

- Due to it's choice of framework being pytorch it's highly popular in Applied CV Community
- Pytorch models can be easily migrated to other frameworks such as onnx, tensorflow.
- It's faster than YoloV4 and has 5 checkpoints for different memory and speed requirements.
- For more details Check this blog: https://blog.roboflow.com/yolov5-is-here/
- We will be using ONNX Format of YoloV5

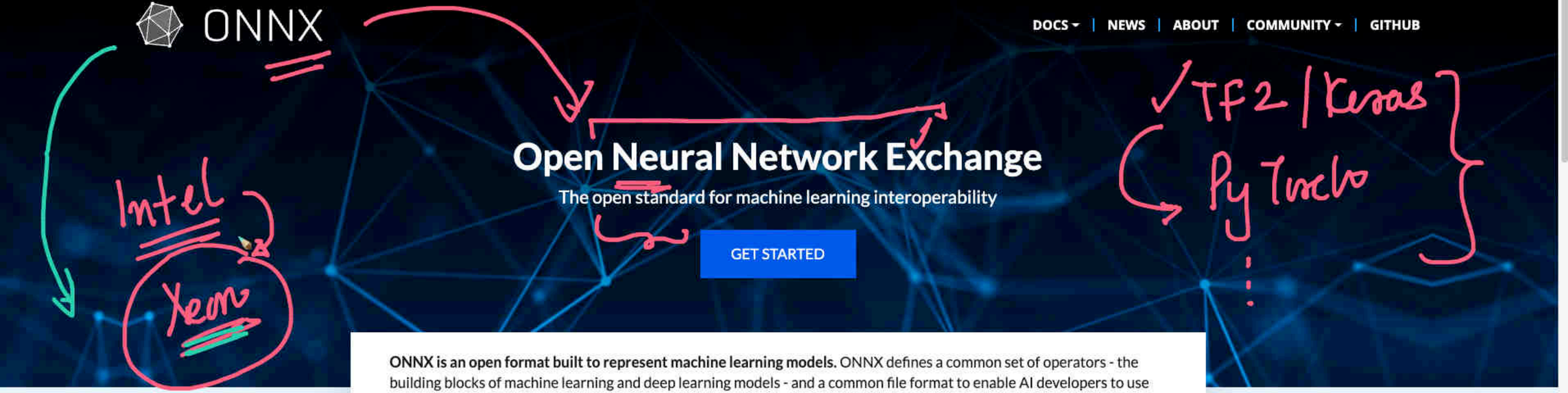
What is ONNX?

ONNX is an open format built to represent machine learning models. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a comm frameworks, tools, runtimes, and compilers

NEWS: Onnx Python Package Read more ›

ONNX

DOCS | NEWS | ABOUT | COMMUNITY | GITHUB



Open Neural Network Exchange

The open standard for machine learning interoperability

GET STARTED

ONNX is an open format built to represent machine learning models. ONNX defines a common set of operators - the building blocks of machine learning and deep learning models - and a common file format to enable AI developers to use models with a variety of frameworks, tools, runtimes, and compilers. LEARN MORE ›

KEY BENEFITS

Interoperability

Develop in your preferred framework without worrying about downstream inferencing implications. ONNX enables you to use your preferred framework with your chosen inference engine.

SUPPORTED FRAMEWORKS ›

Hardware Access

ONNX makes it easier to access hardware optimizations. Use ONNX-compatible runtimes and libraries designed to maximize performance across hardware.

SUPPORTED ACCELERATORS ›

```
INPUT_HEIGHT = 640

# probability threshold to filter boxes with object or no object
OBJECT_SCORE_THRESHOLD = 0.5

# # probability threshold to detect and Assign Class
CLASS_CONFIDENCE_THRESHOLD = 0.45

# IOU AREA Threshold to suppress redundant boxes using NMS
NMS_THRESHOLD = 0.45

# Text parameters used for annotating label on Image
FONT_FACE = cv2.FONT_HERSHEY_SIMPLEX
FONT_SCALE = 0.4
THICKNESS = 1
BOX_COLOR = (0,255,255)
FONT_COLOR= (0,0,0)
```

```
def draw_label(im, label, x, y):
    """Function used for Drawing text/label onto image at location."""

    # Get text size.
    text_size = cv2.getTextSize(label, FONT_FACE, FONT_SCALE, THICKNESS)
    dim, baseline = text_size[0], text_size[1]

    # Use text size to create a BLACK rectangle.
    cv2.rectangle(im, (x,y), (x + dim[0], y - (dim[1]+baseline) ), (255,255,255), cv2.FILLED);

    # Display text inside the rectang
```

```
BOX_COLOR = (0,255,255)
FONT_COLOR= (0,0,0)
```

```
def draw_label(im, label, x, y):
    """Function used for Drawing text/label onto image at location."""

    # Get text size.
    text_size = cv2.getTextSize(label, FONT_FACE, FONT_SCALE, THICKNESS)
    dim, baseline = text_size[0], text_size[1]

    # Use text size to create a BLACK rectangle.
    cv2.rectangle(im, (x,y), (x + dim[0], y - (dim[1]+baseline) ), (255,255,255), cv2.FILLED);

    # Display text inside the rectangle.
    cv2.putText(im, label, (x, y - dim[1]+baseline ), FONT_FACE, FONT_SCALE, FONT_COLOR, THICKNESS, cv2.LINE_AA)
```

```
def yolo_forward_pass(input_image, net):
    """Performs forward pass to generate prediction using image and Loaded Yolo Model as parameters """
    # Create a 4D blob from a frame
    blob = cv2.dnn.blobFromImage(input_image, 1/255,  (INPUT_WIDTH, INPUT_HEIGHT), [0,0,0], 1, crop=False)

    # Sets the input to the network.
    net.setInput(blob)

    # Run the forward pass to get output of the output layers.
    outputs = net.forward(net.getUnconnectedOutLayersNames())

    return outputs
```

```
# Use text size to create a BLACK rectangle.
cv2.rectangle(im, (x,y), (x + dim[0], y - (dim[1]+baseline) ), (255,255,255), cv2.FILLED);

# Display text inside the rectangle.
cv2.putText(im, label, (x, y - dim[1]+baseline ), FONT_FACE, FONT_SCALE, FONT_COLOR, THICKNESS, cv2.LINE_AA)
```

```
def yolo_forward_pass(input_image, net):
    """Performs forward pass to generate prediction using image and Loaded Yolo Model as parameters """
    # Create a 4D blob from a frame
    blob = cv2.dnn.blobFromImage(input_image, 1/255,  (INPUT_WIDTH, INPUT_HEIGHT), [0,0,0], 1, crop=False)

    # Sets the input to the network.
    net.setInput(blob)

    # Run the forward pass to get output of the output layers.
    outputs = net.forward(net.getUnconnectedOutLayersNames())

    return outputs
```

```
def post_process_outputs(input_image, outputs):
    """ Takes Model Prediction outputs from forward pass function and performs post-processing tasks:
    1. Removing predictions with Low Confidence score
    2. Filtering only classes required for our task such as Vehicles, Human, traffic light stop sgn etc
    3. Perform NMS to suppress redundant or Duplicate boxes
    4. Annotate/draw remaining boxes on the Image
    """
    # Lists to hold respective values while unwrapping.
    class_ids = []
```

```
[ ]     # Get text size.
        text_size = cv2.getTextSize(label, FONT_FACE, FONT_SCALE, THICKNESS)
        dim, baseline = text_size[0], text_size[1]

        # Use text size to create a BLACK rectangle.
        cv2.rectangle(im, (x,y), (x + dim[0], y - (dim[1]+baseline) ), (255,255,255), cv2.FILLED);

        # Display text inside the rectangle.
        cv2.putText(im, label, (x, y - dim[1]+baseline ), FONT_FACE, FONT_SCALE, FONT_COLOR, THICKNESS, cv2.LINE_AA)
```

```
[ ] def yolo_forward_pass(input_image, net):
        """Performs forward pass to generate prediction using image and Loaded Yolo Model as parameters """
        # Create a 4D blob from a frame
        blob = cv2.dnn.blobFromImage(input_image, 1/255, (INPUT_WIDTH, INPUT_HEIGHT), [0,0,0], 1, crop=False)

        # Sets the input to the network.
        net.setInput(blob)

        # Run the forward pass to get output of the output layers.
        outputs = net.forward(net.getUnconnectedOutLayersNames())

        return outputs
```

```
[ ] def post_process_outputs(input_image, outputs):
        """ Takes Model Prediction outputs from forward pass function and performs post-processing tasks:
        1. Removing predictions with Low Confidence score
        2. Filtering only classes required for our task such as Vehicles, Human, traffic light stop sgn etc
        3. Perform NMS to suppress redu
        4. Annotate/draw remaining boxe
```

```
# Get text size.
text_size = cv2.getTextSize(label, FONT_FACE, FONT_SCALE, THICKNESS)
dim, baseline = text_size[0], text_size[1]

# Use text size to create a BLACK rectangle.
cv2.rectangle(im, (x,y), (x + dim[0], y - (dim[1]+baseline) ), (255,255,255), cv2.FILLED);

# Display text inside the rectangle.
cv2.putText(im, label, (x, y - dim[1]+baseline ), FONT_FACE, FONT_SCALE, FONT_COLOR, THICKNESS, cv2.LINE_AA)
```

```
def yolo_forward_pass(input_image, net):
    """Performs forward pass to generate prediction using image and Loaded Yolo Model as parameters """
    # Create a 4D blob from a frame
    blob = cv2.dnn.blobFromImage(input_image, 1/255,  (INPUT_WIDTH, INPUT_HEIGHT), [0,0,0], 1, crop=False)

    # Sets the input to the network.
    net.setInput(blob)

    # Run the forward pass to get output of the output layers.
    outputs = net.forward(net.getUnconnectedOutLayersNames())

    return outputs
```

```
def post_process_outputs(input_image, outputs):
    """ Takes Model Prediction outputs from forward pass function and performs post-processing tasks:
    1. Removing predictions with Low Confidence score
    2. Filtering only classes required for our task such as Vehicles, Human, traffic light stop sgn etc
    3. Perform NMS to suppress redu
    4. Annotate/draw remaining boxe
```

```
[ ]         outputs = net.forward(net.getUnconnectedOutLayersNames())

            return outputs
```

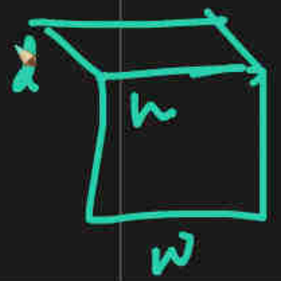
```
[ ] def post_process_outputs(input_image, outputs):
        """ Takes Model Prediction outputs from forward pass function and performs post-processing tasks:
        1. Removing predictions with Low Confidence score
        2. Filtering only classes required for our task such as Vehicles, Human, traffic light stop sgn etc
        3. Perform NMS to suppress redundant or Duplicate boxes
        4. Annotate/draw remaining boxes on the Image
        """
        # Lists to hold respective values while unwrapping.
        class_ids = []
        confidences = []
        boxes = []

        # Rows
        rows = outputs[0].shape[1]
        image_height, image_width = input_image.shape[:2]

        # Resizing factor.
        x_factor = image_width / INPUT_WIDTH
        y_factor =  image_height / INPUT_HEIGHT

        # Iterate through detections.
        for r in range(rows):
            row = outputs[0][0][r]
            confidence = row[4]

            # Discard bad detections
```

```
        # Run the forward pass to get output of the output layers.
        outputs = net.forward(net.getUnconnectedOutLayersNames())

        return outputs
```

```
def post_process_outputs(input_image, outputs):
    """ Takes Model Prediction outputs from forward pass function and performs post-processing tasks:
    1. Removing predictions with Low Confidence score
    2. Filtering only classes required for our task such as Vehicles, Human, traffic light stop sgn etc
    3. Perform NMS to suppress redundant or Duplicate boxes
    4. Annotate/draw remaining boxes on the Image
    """
    # Lists to hold respective values while unwrapping.
    class_ids = []
    confidences = []
    boxes = []

    # Rows
    rows = outputs[0].shape[1]
    image_height, image_width = input_image.shape[:2]

    # Resizing factor.
    x_factor = image_width / INPUT_WIDTH
    y_factor =  image_height / INPUT_HEIGHT

    # Iterate through detections.
    for r in range(rows):
        row = outputs[0][0][r]
        confidence = row[4]
```

```
# Use text size to create a BLACK rectangle.
cv2.rectangle(im, (x,y), (x + dim[0], y - (dim[1]+baseline) ), (255,255,255), cv2.FILLED);

# Display text inside the rectangle.
cv2.putText(im, label, (x, y - dim[1]+baseline ), FONT_FACE, FONT_SCALE, FONT_COLOR, THICKNESS, cv2.LINE_AA)
```

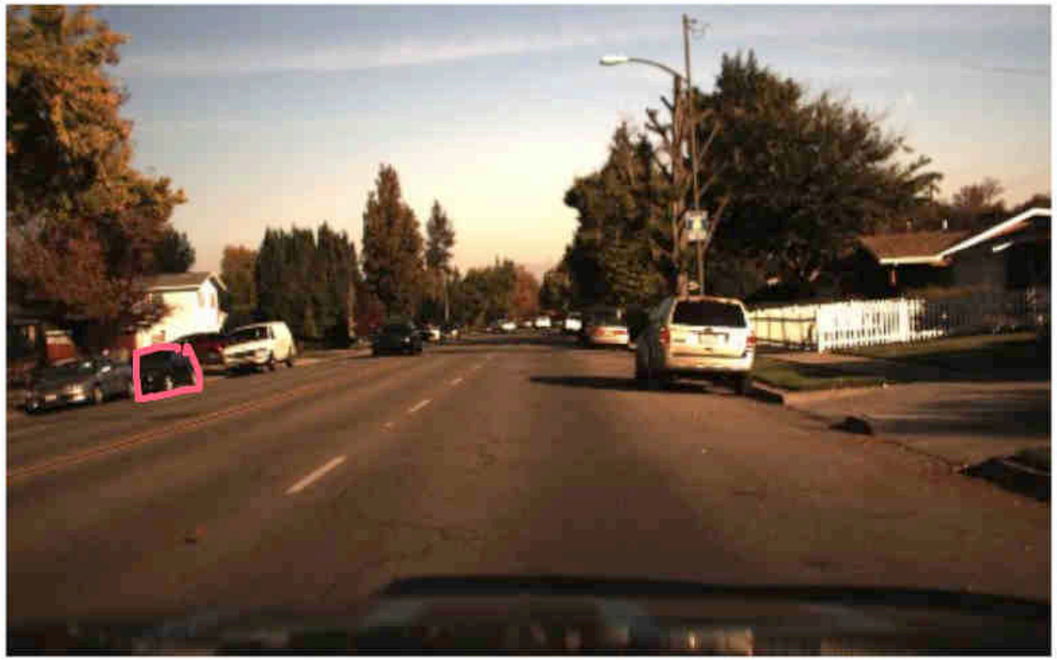
```
def yolo_forward_pass(input_image, net):
    """Performs forward pass to generate prediction using image and Loaded Yolo Model as parameters """
    # Create a 4D blob from a frame
    blob = cv2.dnn.blobFromImage(input_image, 1/255,  (INPUT_WIDTH, INPUT_HEIGHT), [0,0,0], 1, crop=False)

    # Sets the input to the network.
    net.setInput(blob)

    # Run the forward pass to get output of the output layers.
    outputs = net.forward(net.getUnconnectedOutLayersNames())

    return outputs
```

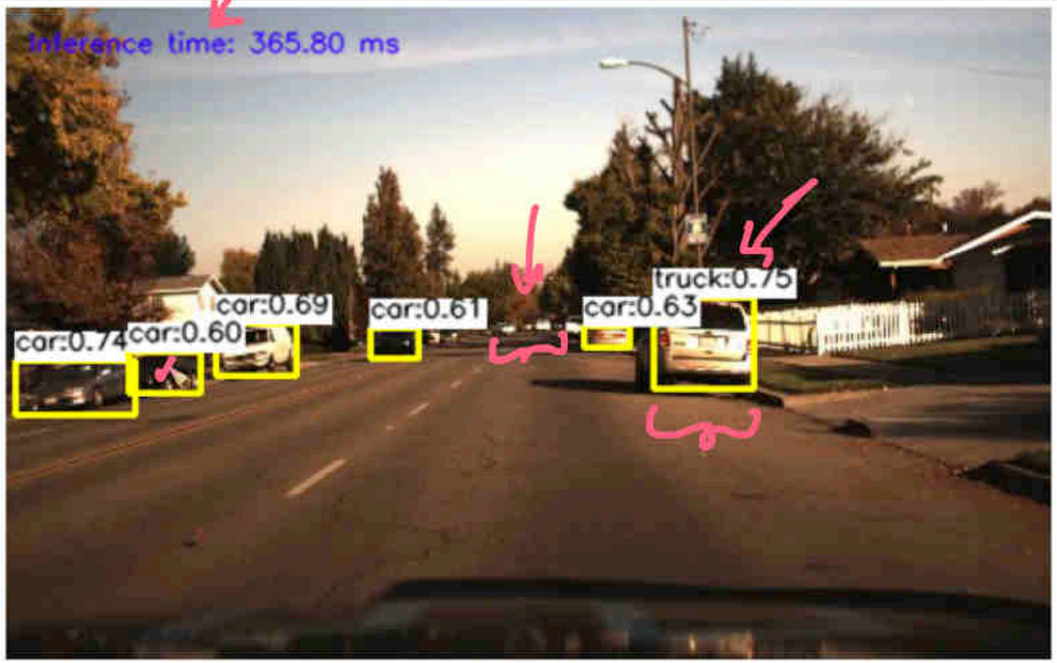



```
def post_process_outputs(input_image, outputs):
    """ Takes Model Prediction outputs from forward pass function and performs post-processing tasks:
    1. Removing predictions with Low Confidence score
    2. Filtering only classes required for our task such as Vehicles, Human, traffic light stop sgn etc
    3. Perform NMS to suppress redundant or Duplicate boxes
    4. Annotate/draw remaining boxes on the Image
    """
    # Lists to hold respective values while unwrapping.
    class_ids = []
    confidences = []
```

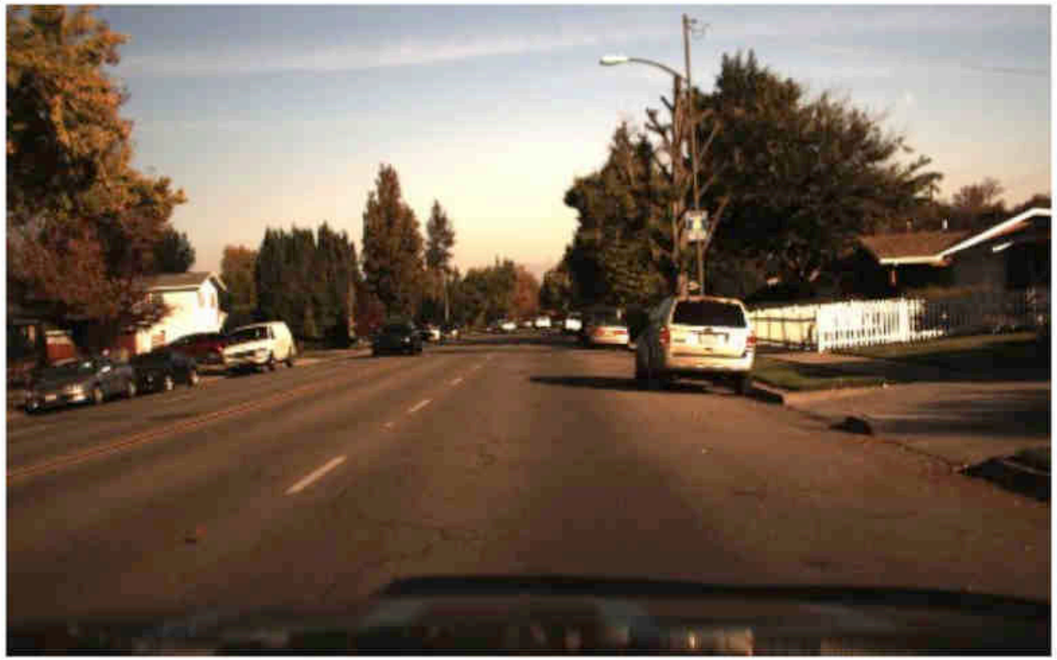
```
[ ]  from google.colab.patches import cv2_imshow
     import matplotlib.pyplot as plt


     # Load class names.
     classesFile = "/content/coco.names.txt"
     classes = None

     with open(classesFile, 'rt') as f:
           classes = f.read().rstrip('\n').split('\n')

     # Load the model using cv2.dnn.readNet: Yolo V5 Nano Model
     modelWeights = "/content/yolov5n.onnx"
     net = cv2.dnn.readNet(modelWeights)

     for image_path in sample_images:
       # Load image
       frame = cv2.imread(image_path)


       # Process image:
       detections = yolo_forward_pass(frame, net)
       pred_img = post_process_outputs(frame.copy(), detections)

       """
       Annotate efficiency information. The function getPerfProfile returns the overall time for inference(t)
       and the timings for each of the layers(in layersTimes).
       """
       t, _ = net.getPerfProfile()
       label = 'Inference time: %.2f ms' %
       cv2.putText(pred_img, label, (10, 2                                    .INE_AA)
```

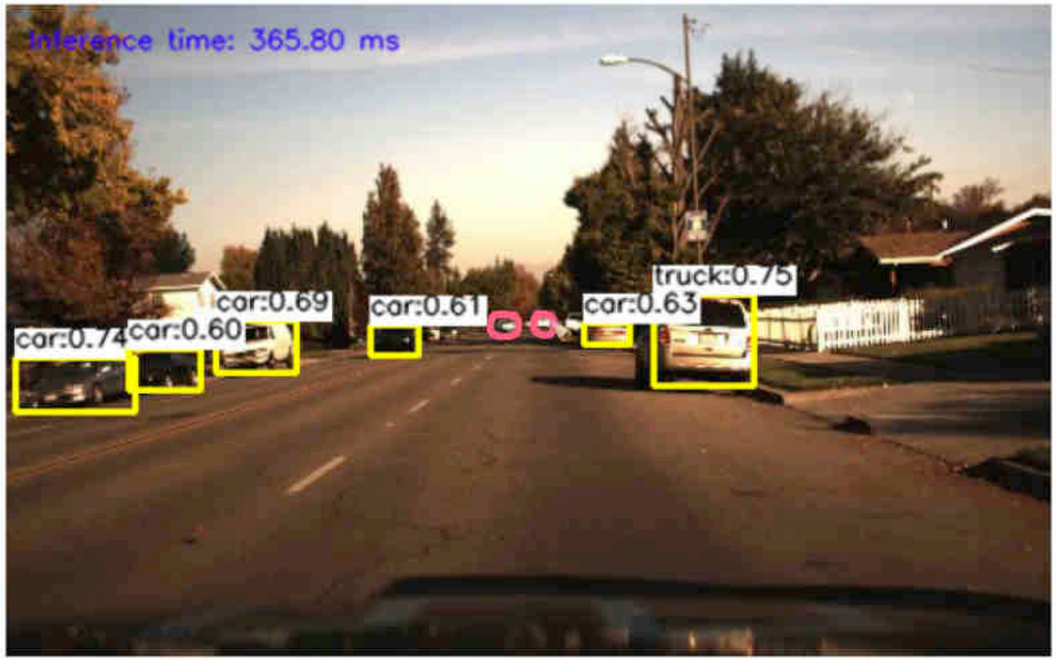
```
ax2 = fig.add_subplot(1,2,2)
pred_img = cv2.cvtColor(pred_img, cv2.COLOR_BGR2RGB )
ax2.imshow(pred_img)
ax2.axis('off')

plt.show()
```





```
ax2 = fig.add_subplot(1,2,2)
pred_img = cv2.cvtColor(pred_img, cv2.COLOR_BGR2RGB )
ax2.imshow(pred_img)
ax2.axis('off')

plt.show()
```





Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:36

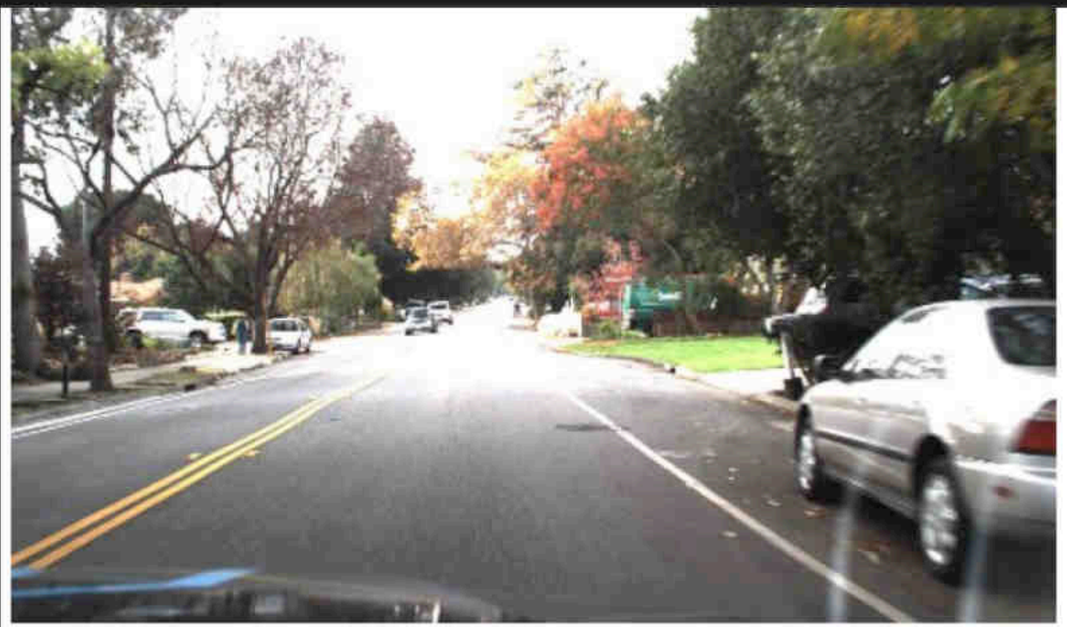
L8: ObjectDetectio | ONNX | Home | Object Detection F | Network-architect | Post_Read: Retina | L9: Object Segme | GitHub - vdumoul | ROI pooling vs. RC | An Introduction to | Lil'Log

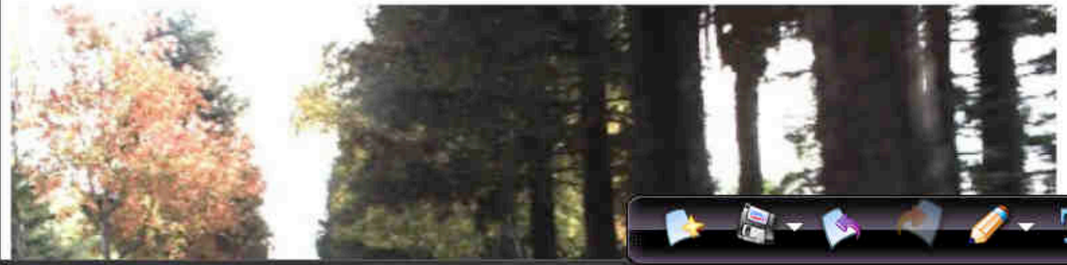
colab.research.google.com/drive/1XfHwVqTVMiw51yx1DdlqATyHjf-qybJS#scrollTo=Hk41mM0Q3aYW

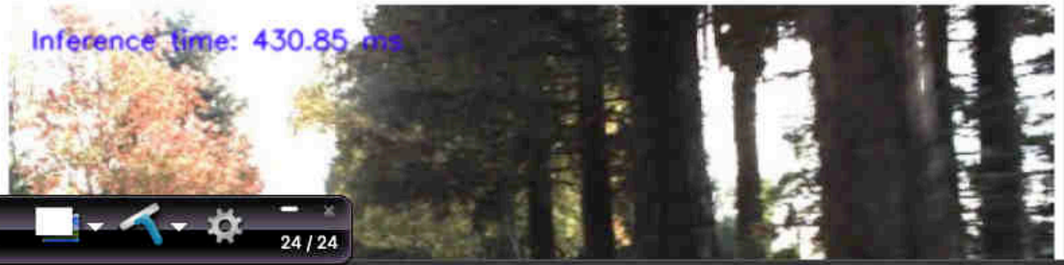
Update

+ Code + Text Connect

```
[ ]   ax2 = fig.add_subplot(1,2,2)
      pred_img = cv2.cvtColor(pred_img, cv2.COLOR_BGR2RGB )
      ax2.imshow(pred_img)
      ax2.axis('off')

      plt.show()
```

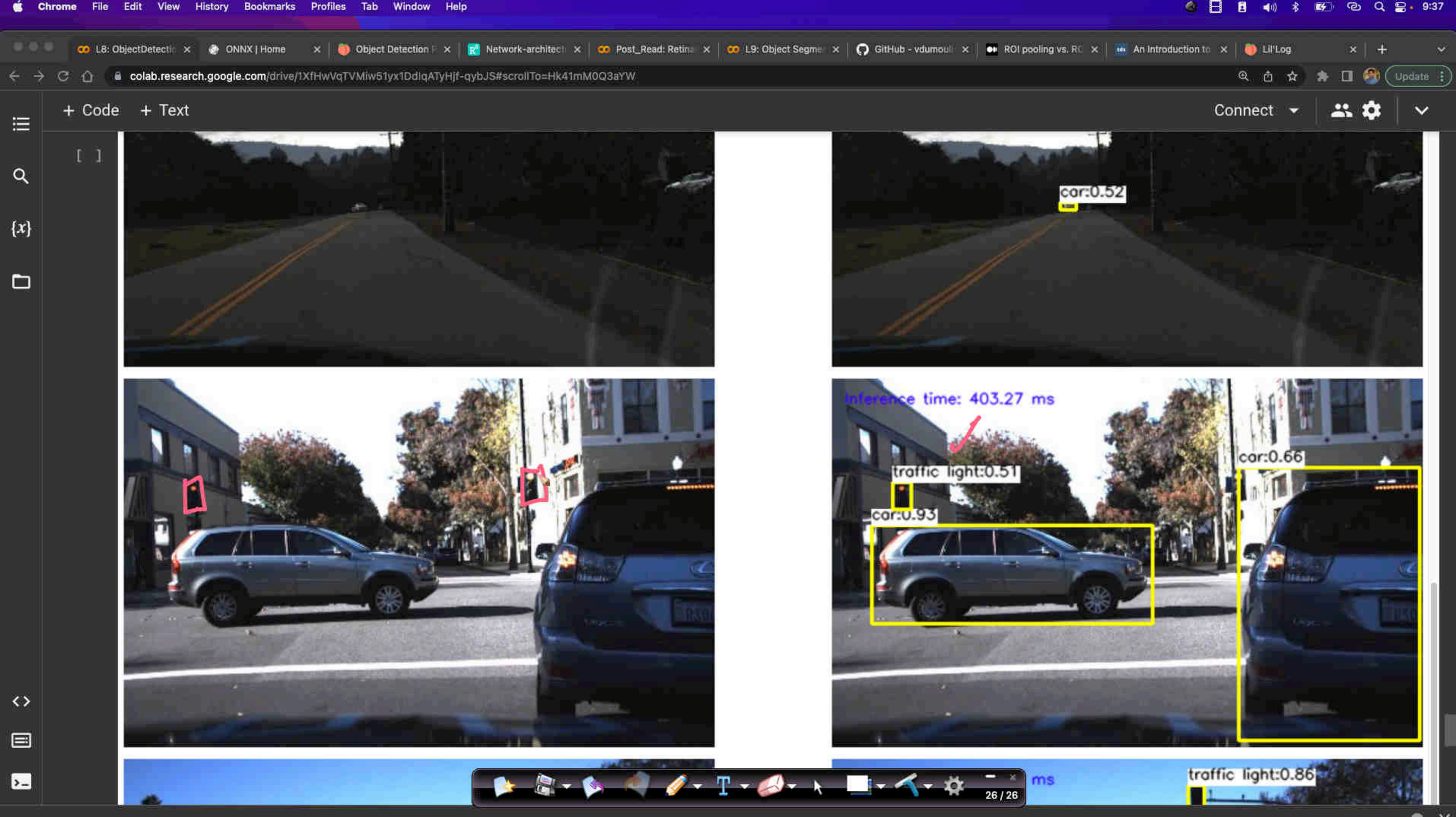




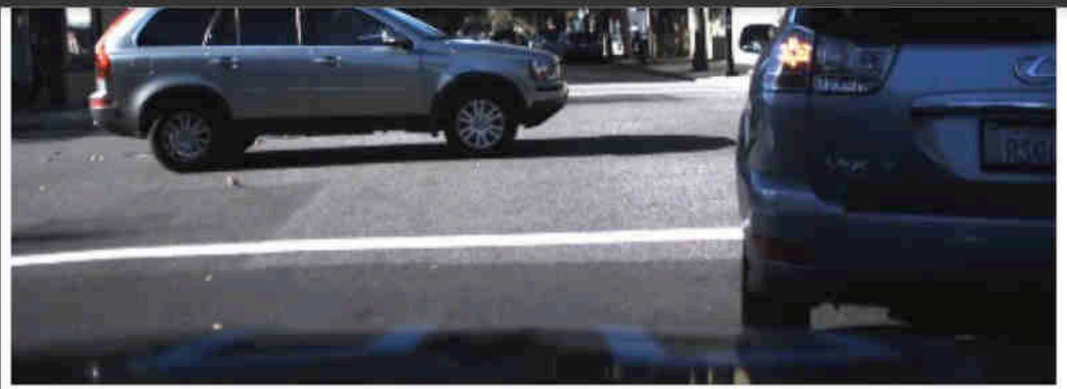


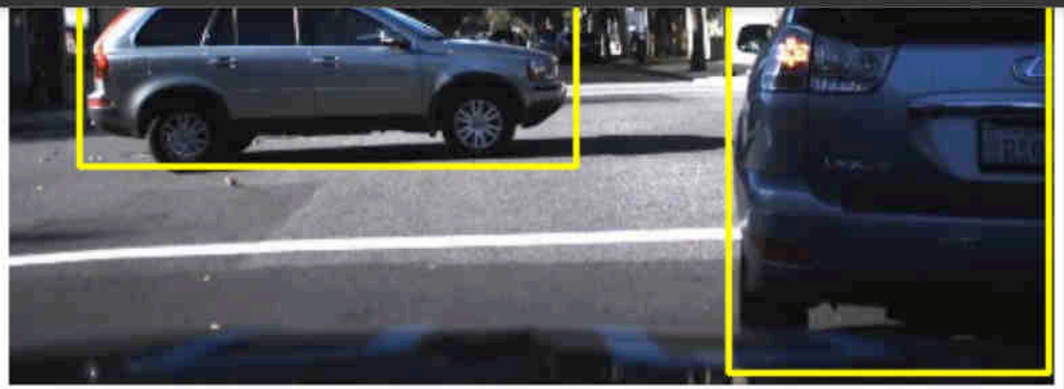
24 / 24

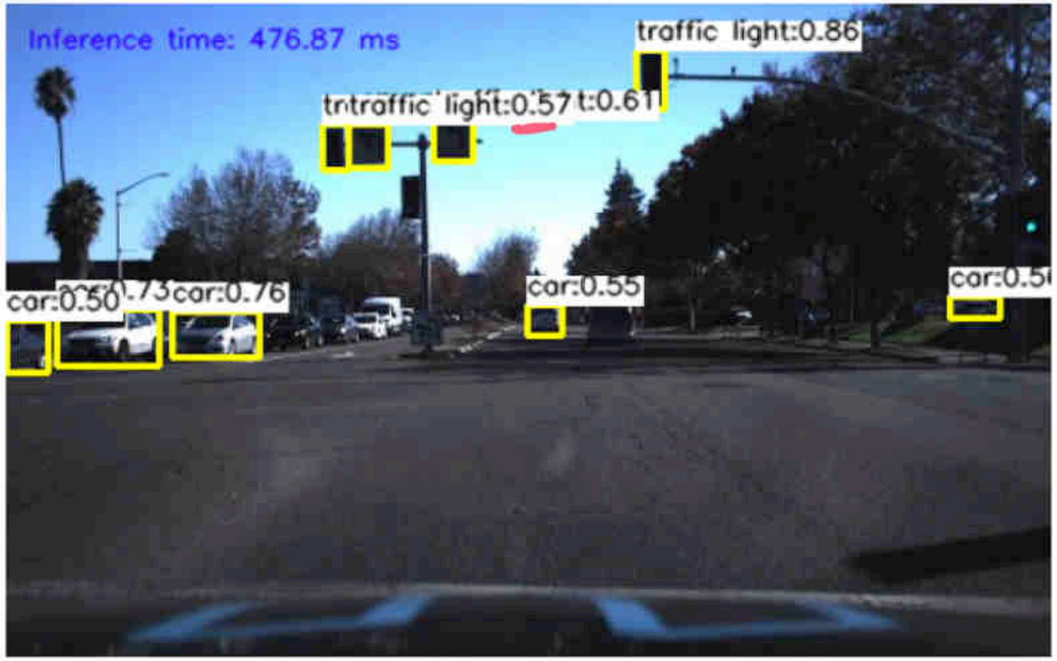
```
ax2 = fig.add_subplot(1,2,2)
pred_img = cv2.cvtColor(pred_img, cv2.COLOR_BGR2RGB )
ax2.imshow(pred_img)
ax2.axis('off')

plt.show()
```


Chrome File Edit View History Bookmarks Profiles Tab Window Help
9:37
colab.research.google.com/drive/1XfHwVqTVMiw51yx1DdlqATyHjf-qybJS#scrollTo=Hk41mM0Q3aYW
+ Code
+ Text
Connect
car:0.52
Inference time: 403.27 ms
traffic light:0.51
car:0.93
car:0.66
traffic light:0.86
26 / 26

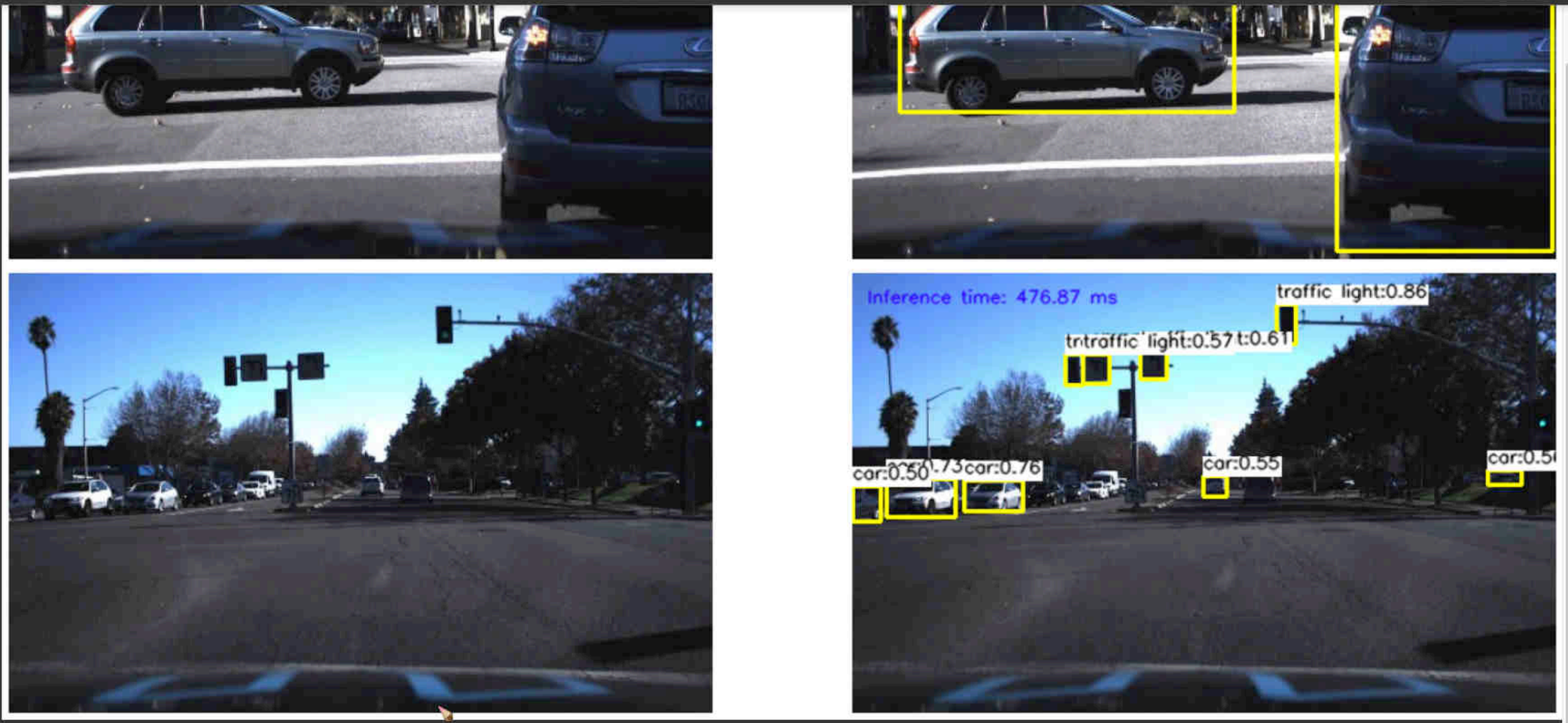
[ ]



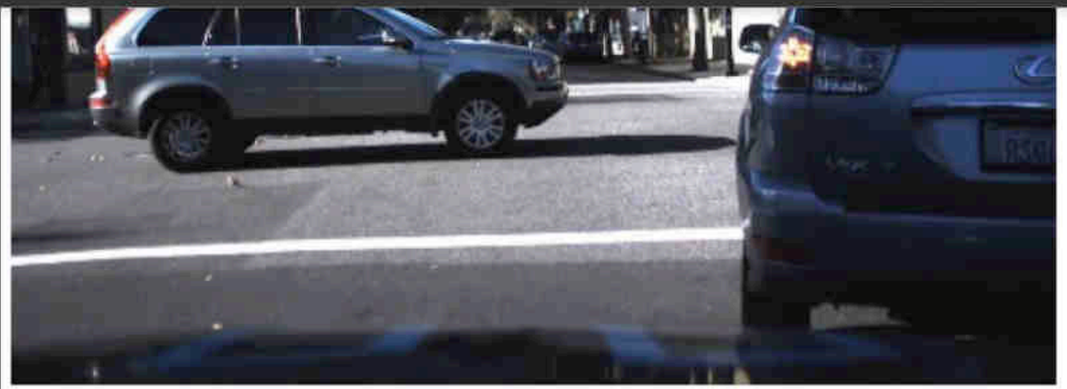


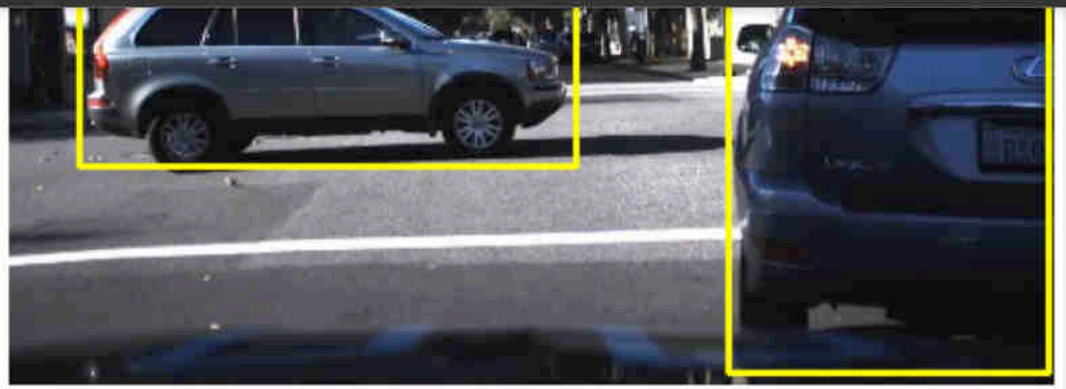


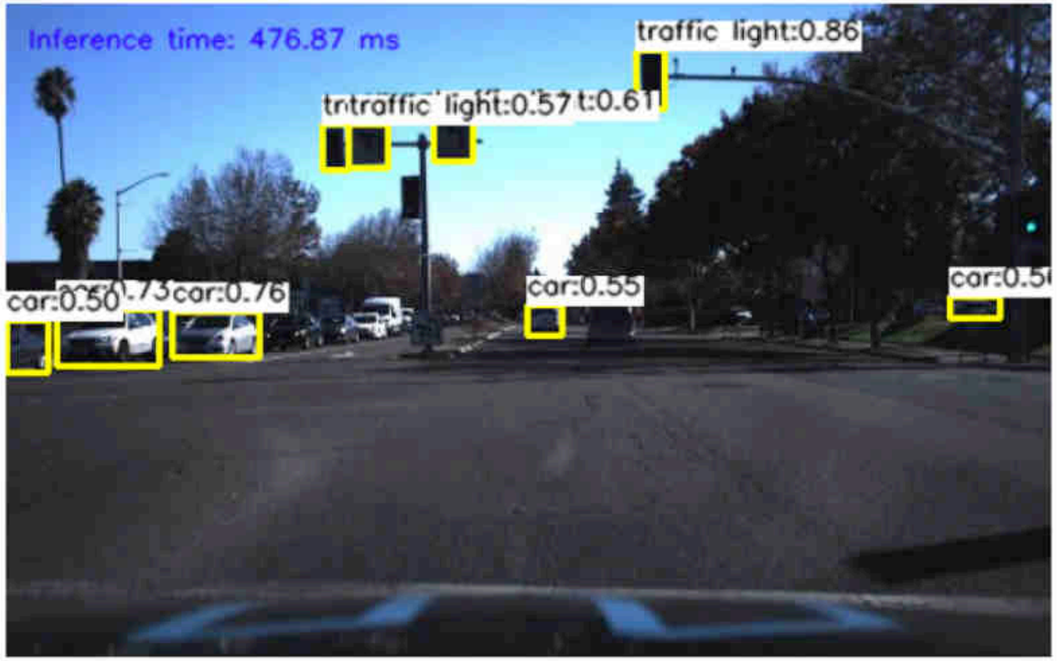
- It did a great job of predicting Vehicles and



- It did a great job of predicting Vehicles and







- It did a great job of predicting Vehicles and

[ ]

- It did a great job of predicting Vehicles and Traffic light with just the pre-trained model and
- Every image was processed in less than a Second on CPU without any specialized hardware(1000 ms = 1 second).
- To Enhance performance we can finetune the model on our custom dataset as well.

Question:

```
- Try tuning Score threshild and Class threshold probabilities and check impact on predictions.
- Evaluate MAP of the pre-trained model on full dataset
- Run inference using Other Variants of Yolo Model and compare performance in terms of Latency and Accuracy/MAP
- Fine tune the model on dataset and compare performance with Pre-Trained Model in terms of MAP.
```

Conclusion:

- Using Single Stage Detection method specifically pre-trained YOLOV5 Model, we developed a very fast and accurate detector for our use case which can run inferences in Real Time

```
# Load the model using cv2.dnn.readNet: Yolo V5 Nano Model
modelWeights = "/content/yolov5n.onnx"
net = cv2.dnn.readNet(modelWeights)

for image_path in sample_images:
  # Load image
  frame = cv2.imread(image_path)


  # Process image:
  detections = yolo_forward_pass(frame, net)
  pred_img = post_process_outputs(frame.copy(), detections)

  """
  Annotate efficiency information. The function getPerfProfile returns the overall time for inference(t)
  and the timings for each of the layers(in layersTimes).
  """
  t, _ = net.getPerfProfile()
  label = 'Inference time: %.2f ms' % (t * 1000.0 /  cv2.getTickFrequency())
  cv2.putText(pred_img, label, (10, 20), FONT_FACE, FONT_SCALE,  (255, 0, 50), THICKNESS, cv2.LINE_AA)

  fig = plt.figure(figsize=(28,16))
  ax1 = fig.add_subplot(1,2,1)
  frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB )
  ax1.imshow(frame)
  ax1.axis('off')


  ax2 = fig.add_subplot(1,2,2)
  pred_img = cv2.cvtColor(pred_img, c
```

24 fps

31 / 31

Q

LIDAR

Chrome File Edit View History Bookmarks Profiles Tab Window Help 9:41

L8: ObjectDete | ONNX | Home | Object Detectio | Network-archit | Post_Read: Ret | L9: Object Segn | GitHub - vdumo | ROI pooling vs. | An Introduction | Lil'Log | lidar - Google S

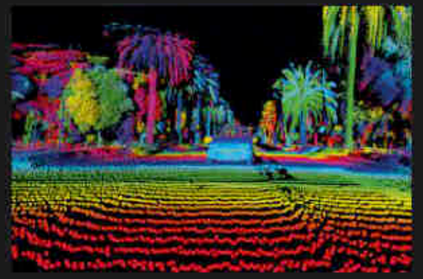
google.com/search?q=lidar&rlz=1C5CHFA_enIN958IN958&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjKno_ojeD6AhXITWwGHQIHBzAQ_AUoAXoECAEQAw&biw=1745&bih=887&dpr=1.1

Update

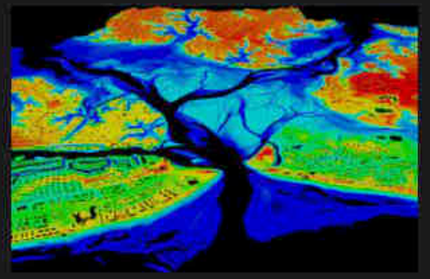
Google lidar

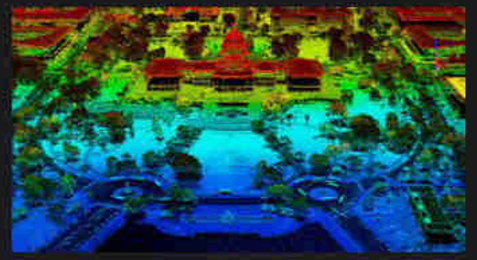
All Images News Videos Books More Tools Collections SafeSearch

scanning | sensor | autonomous | point cloud | drone | velodyne | mapping | radar | self driving | laser | forest | remote sensing

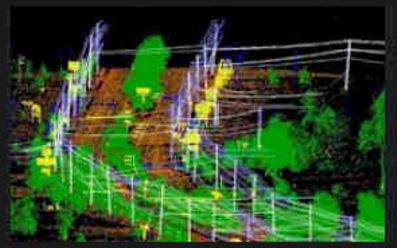

What is LiDAR technology and how does ...
geospatialworld.net

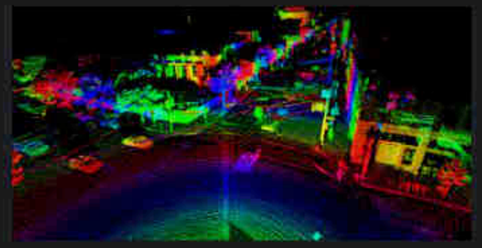

Lidar Wavelengths | Velodyne Lidar
velodynelidar.com

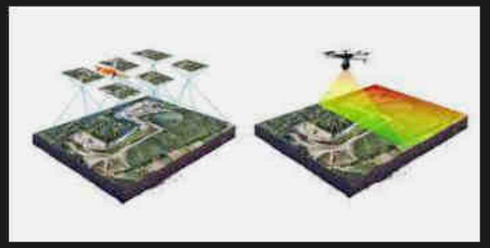

What is lidar?
oceanservice.noaa.gov

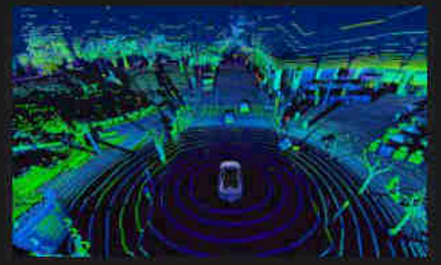

LiDAR: Light Detection and Ranging ...
gisgeography.com

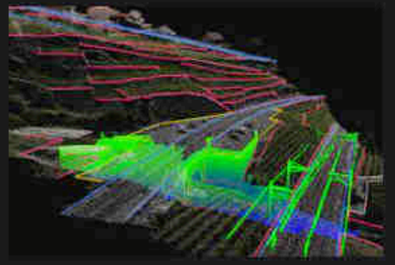

Lidar Explained - Geography Realm
geographyrealm.com

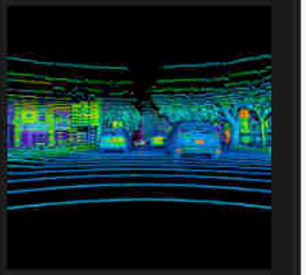

What Is LiDAR and How Can I Use It ...
allaboutcircuits.com

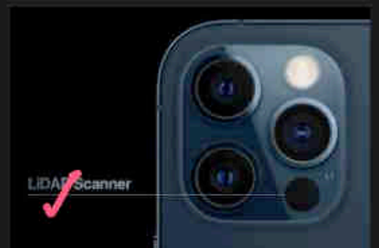

Sensing a bright future for lidar
spie.org

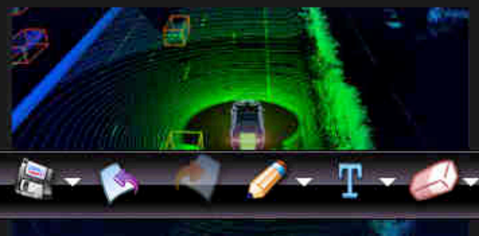

Drone photogrammetry vs. LIDAR: what ...
wingtra.com

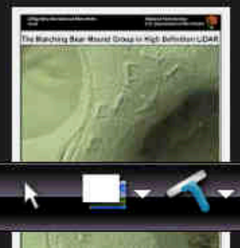

Multibeam Lidar for Mobile Mapping ...
gim-international.com

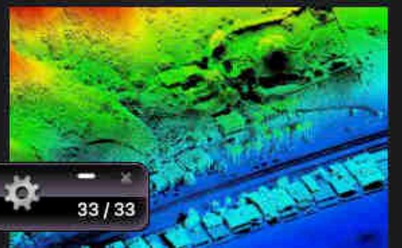

photogrammetry | Pix4D
pix4d.com


LiDARs for self-driving vehicl...
automotiveworld.com









33 / 33

Chrome File Edit View History Bookmarks Profiles Tab Window Help

L8: ObjectDete | ONNX | Home | Object Detectio | Network-archit | Post_Read: Ret | L9: Object Segn | GitHub - vdumo | ROI pooling vs. | An Introduction | Lil'Log | lidar - Google S

google.com/search?q=lidar&rlz=1C5CHFA_enIN958IN958&source=lnms&tbm=isch&sa=X&ved=2ahUKEwjKno_ojeD6AhXITWwGHQIHBzAQ_AUoAXoECAEQAw&biw=1745&bih=887&dpr=1.1

Update

Google

lidar

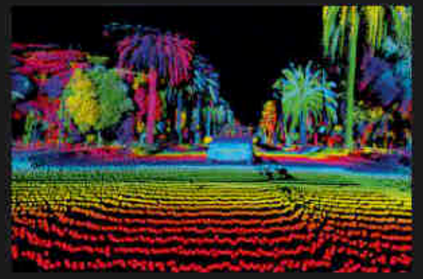
All | Images | News | Videos | Books | More | Tools

RGBD

4D bldg

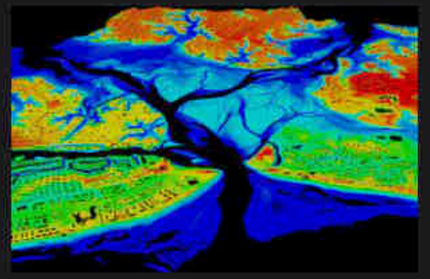
Collections | SafeSearch

scanning | sensor | autonomous | point cloud | drone | velodyne | mapping | radar | self driving | laser | forest | remote sensing

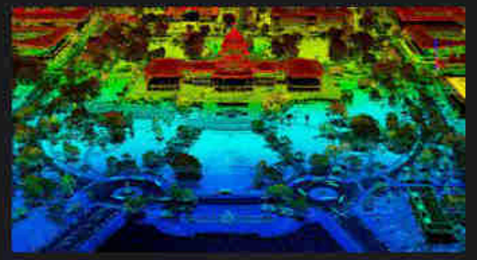


What is LiDAR technology and how does ...
geospatialworld.net

Lidar Wavelengths | Velodyne Lidar
velodynelidar.com

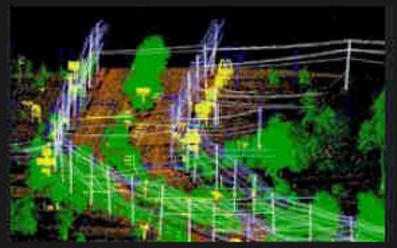


What is lidar?
oceanservice.noaa.gov

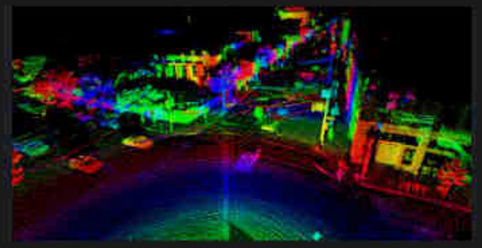


LiDAR: Light Detection and Ranging ...
gisgeography.com

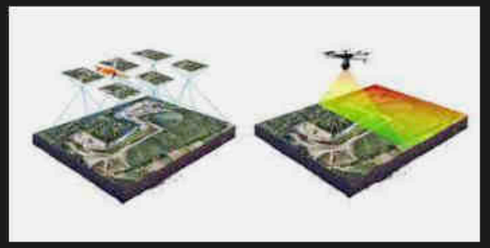
Lidar Explained - Geography Realm
geographyrealm.com



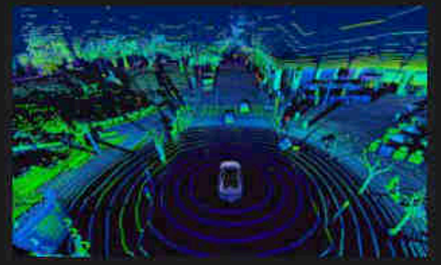
What Is LiDAR and How Can I Use It ...
allaboutcircuits.com



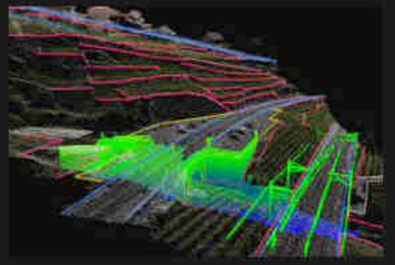
Sensing a bright future for lidar
spie.org



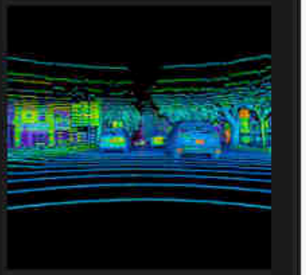
Drone photogrammetry vs. LIDAR: what ...
wingtra.com



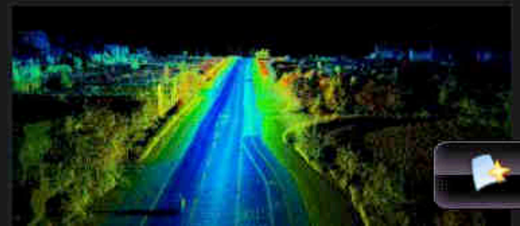
Multibeam Lidar for Mobile Mapping ...
gim-international.com

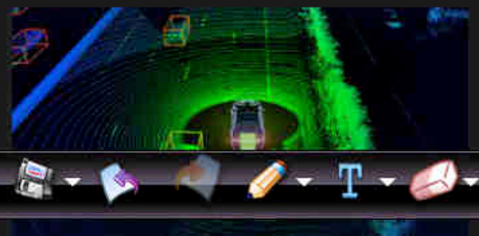


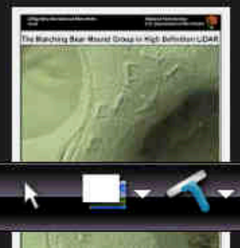
photogrammetry | Pix4D
pix4d.com

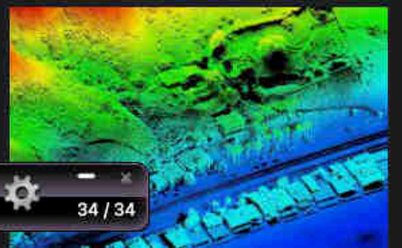


LiDARs for self-driving vehicl...
automotiveworld.com









34 / 34

NEWS

YOLOv5 is Here: State-of-the-Art Object Detection at 140 FPS

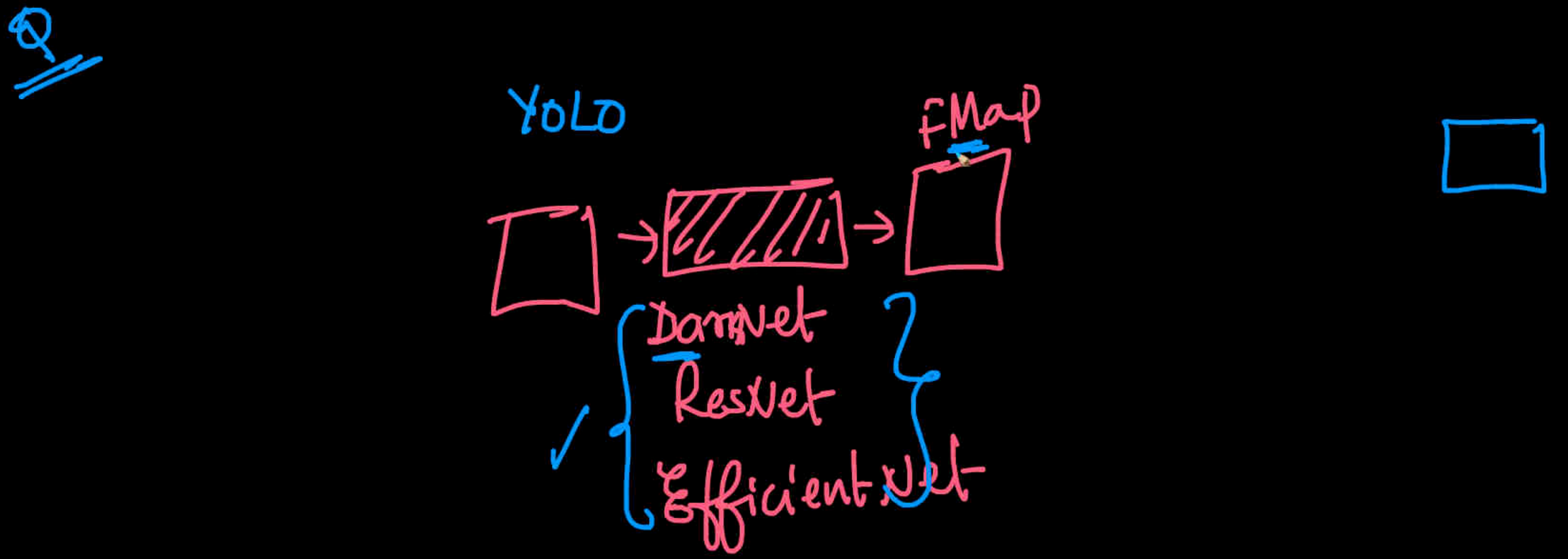
Joseph Nelson, Jacob Solawetz
JUN 10, 2020 | 4 MIN READ

Less than 50 days after the release YOLOv4, YOLOv5 improves accessibility for realtime object detection.

June 29, YOLOv5 has released the first official version of the repository. We wrote a new deep dive on YOLOv5.

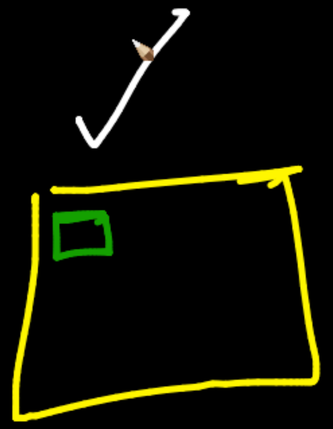
June 12, 8:08 AM CDT Update: In response to to community feedback, we have written a more

TABLE OF CONTENTS

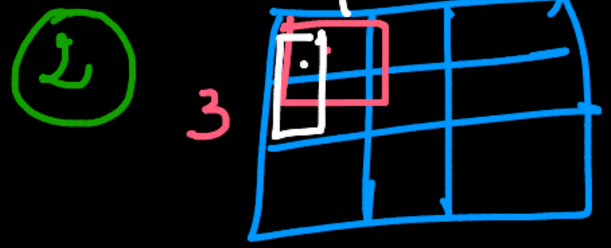


YOLO Limitations

1. Small-objects



2. ~~One~~ num 5 bounding boxes per cell [3 typically]

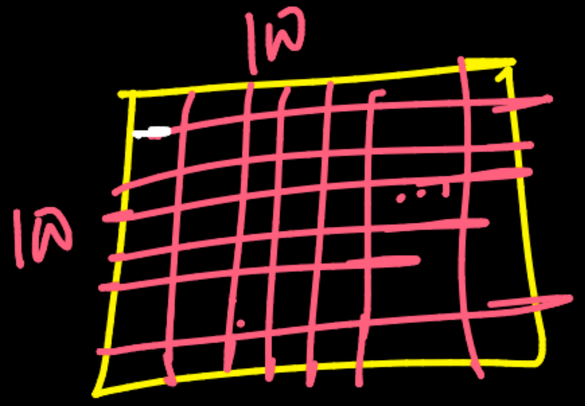


(let)

100 x 10
19 x 19
56 x 56
⋮

Centroids close to one another

3

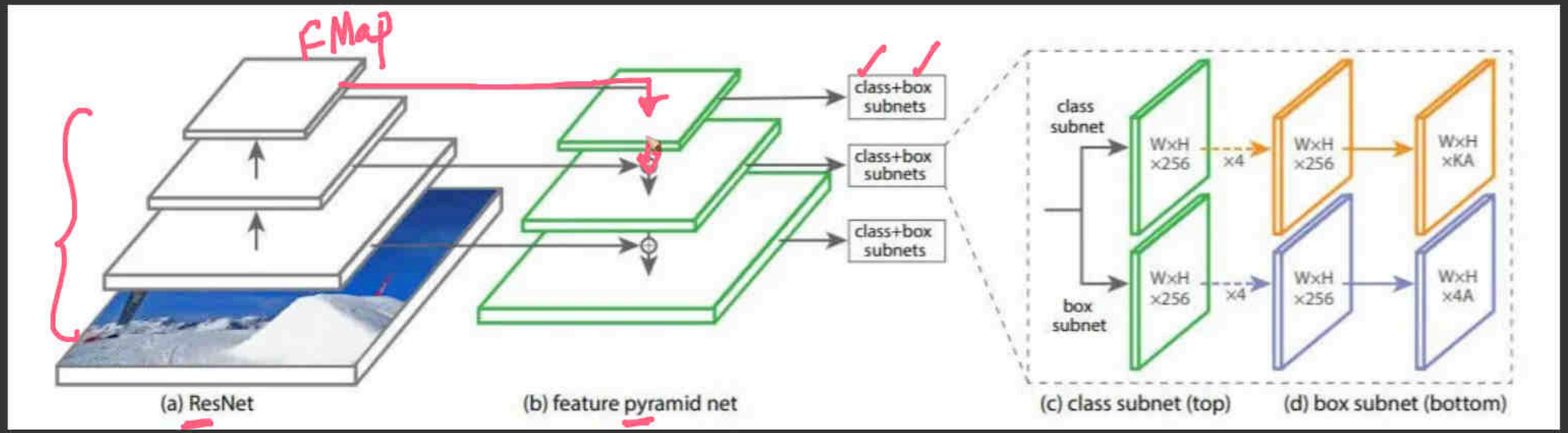


lots of cells with no centroid

→ 10 boxes
no centroid

output sparse

imbalanced classifi



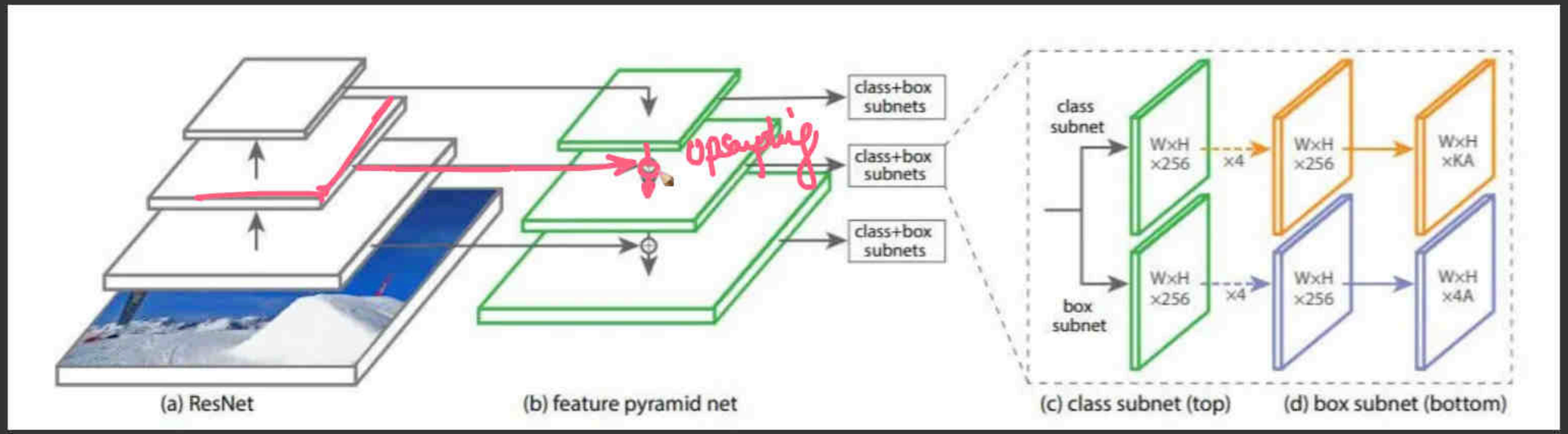
For better understanding, Let's understand each component of architecture separately

The backbone Network

Bottom up pathway:

- Bottom up pathway (eg. ResNet) is used for **feature extraction**.
- So, It calculates `the feature maps at different scales` , **irrespective of the input image size**.

Top down pathway with lateral connections:



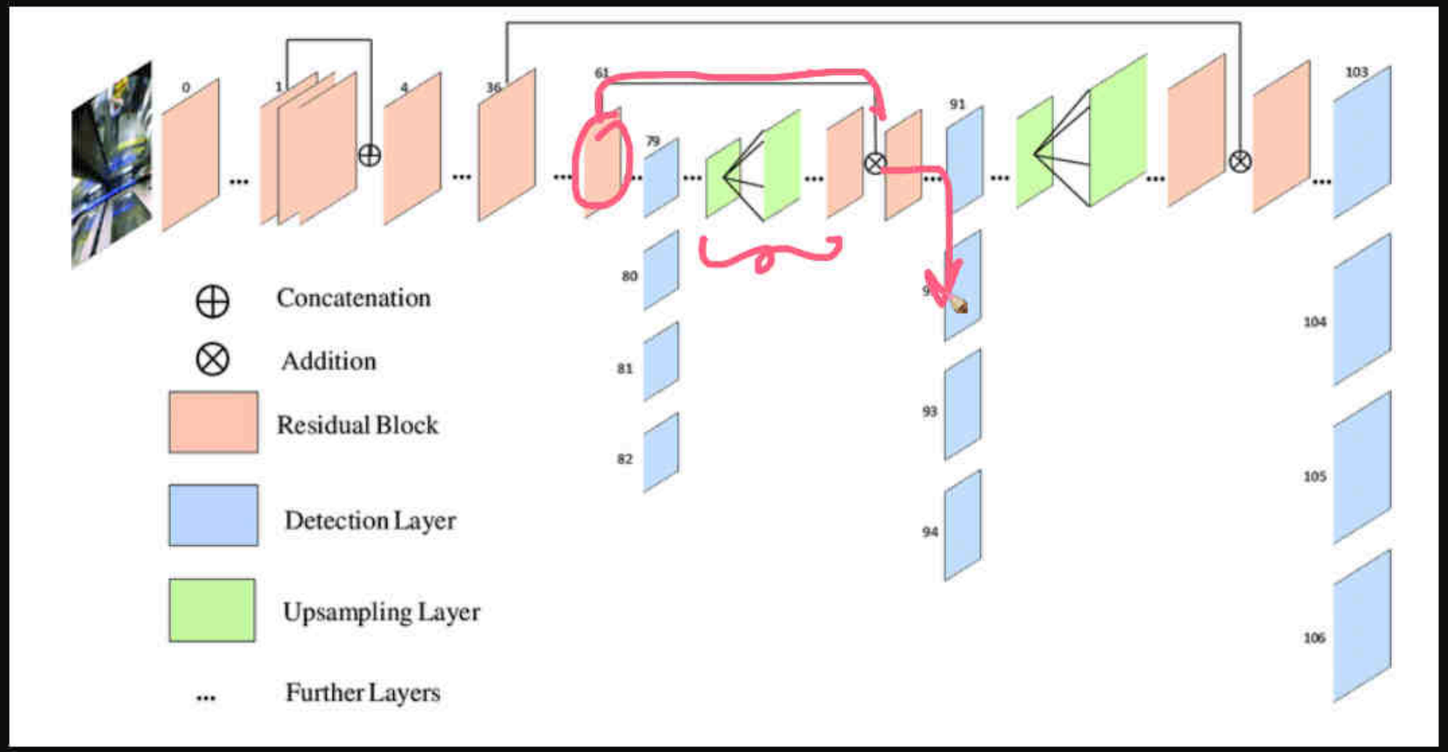
For better understanding, Let's understand each component of architecture separately

The backbone Network

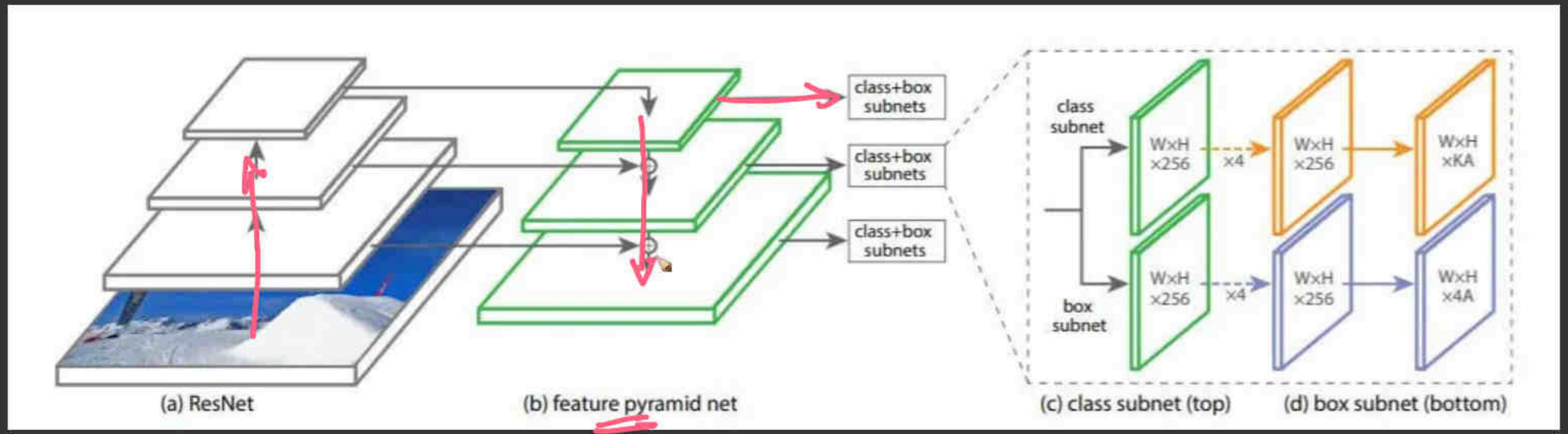
Bottom up pathway:

- Bottom up pathway (eg. ResNet) is used for **feature extraction**.
- So, It calculates `the feature maps at different scales` , **irrespective of the input image size**.

Top down pathway with lateral connections:


0
1
4
36
61
79
91
103
80
81
82
93
94
104
105
106
Concatenation
Addition
Residual Block
Detection Layer
Upsampling Layer
...
Further Layers

+ Code + Text Last edited on 12 September Connect



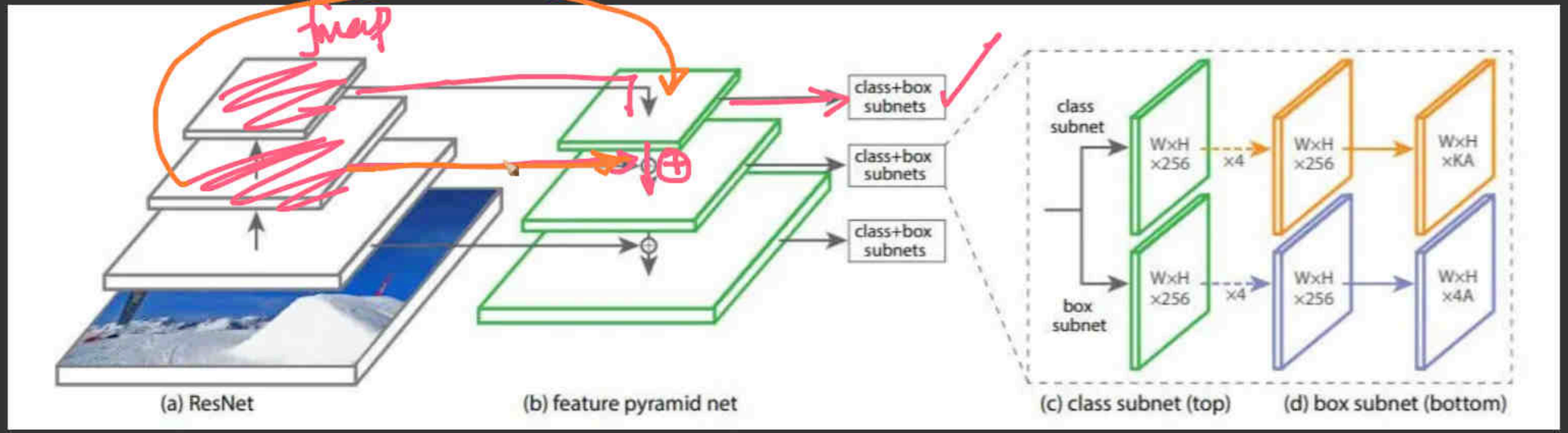
For better understanding, Let's understand each component of architecture separately

The backbone Network

Bottom up pathway:

- Bottom up pathway (eg. ResNet) is used for **feature extraction**.
- So, It calculates `the feature maps at different scales` , **irrespective of the input image size**.

Top down pathway with lateral connections:



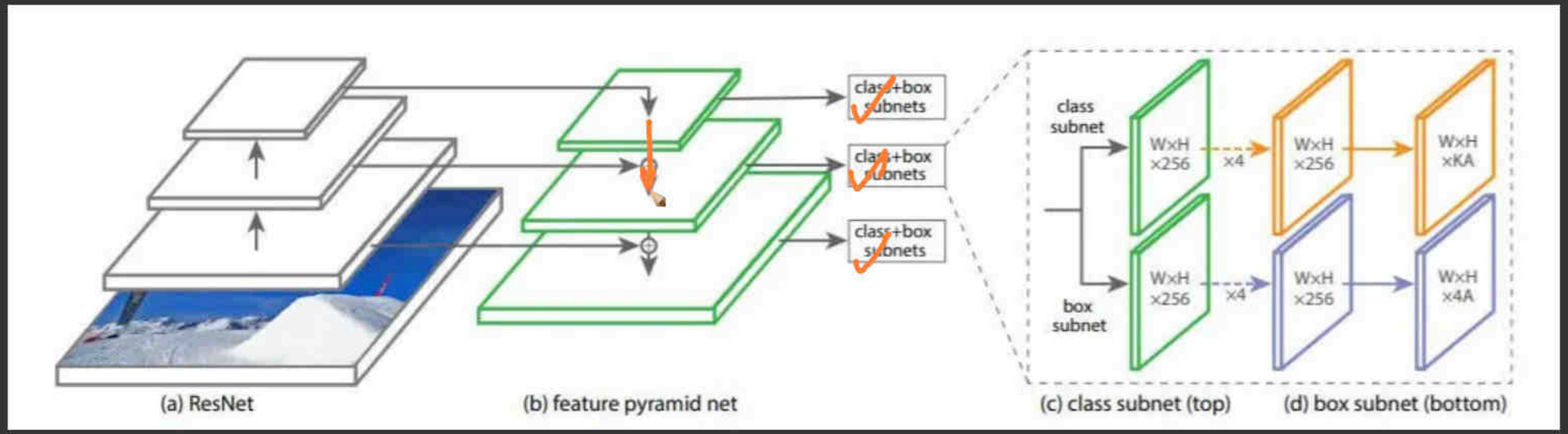
For better understanding, Let's understand each component of architecture separately

The backbone Network

Bottom up pathway:

- Bottom up pathway (eg. ResNet) is used for **feature extraction**.
- So, It calculates `the feature maps at different scales` , **irrespective of the input image size**.

Top down pathway with lateral connections:



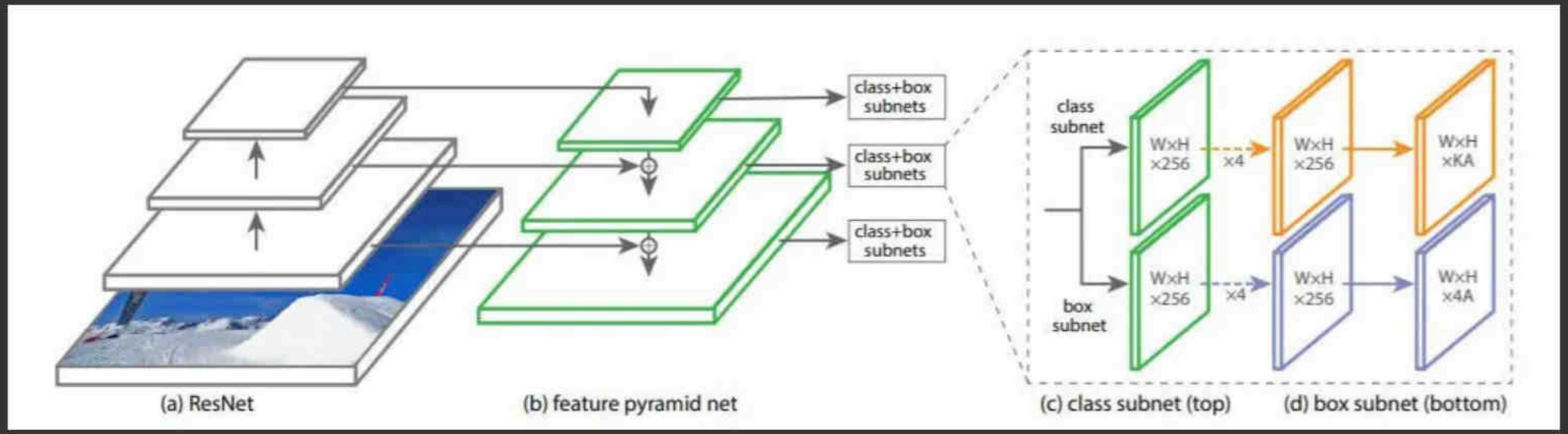
For better understanding, Let's understand each component of architecture separately

The backbone Network

Bottom up pathway:

- Bottom up pathway (eg. ResNet) is used for **feature extraction**.
- So, It calculates `the feature maps at different scales` , **irrespective of the input image size**.

Top down pathway with lateral connections:



For better understanding, Let's understand each component of architecture separately

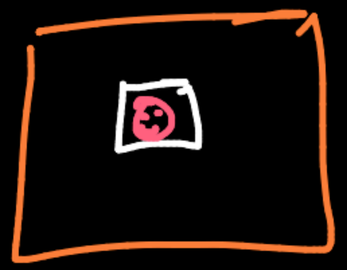
The backbone Network

Bottom up pathway:

- Bottom up pathway (eg. ResNet) is used for **feature extraction**.
- So, It calculates `the feature maps at different scales` , **irrespective of the input image size**.

Top down pathway with lateral connections:

Face → detection (YOLO) ✓



→ recognition

100 → 2-3 images

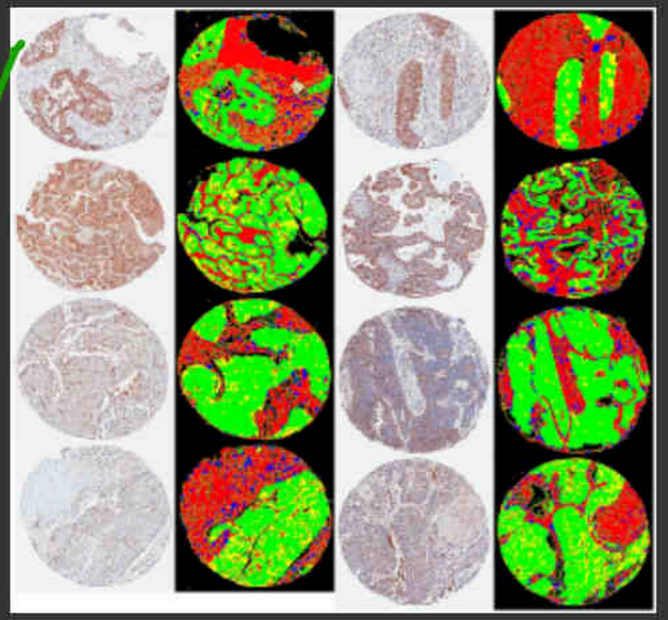
FaceNet

[Siamese n/ws

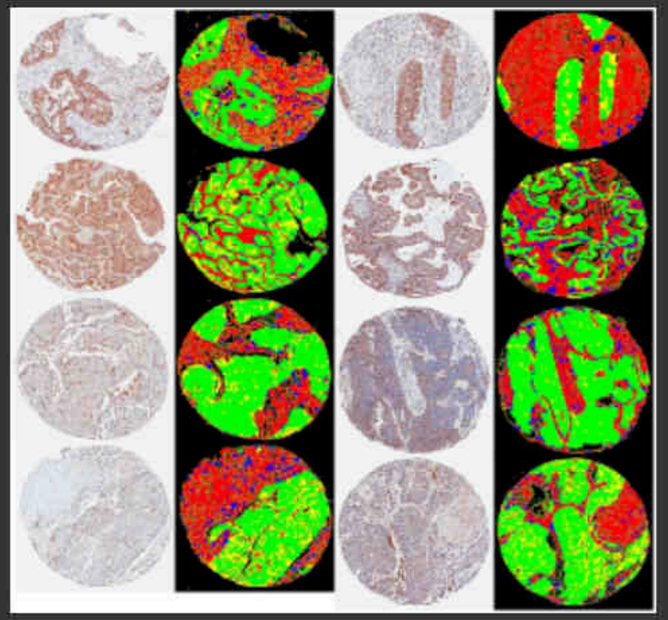
&

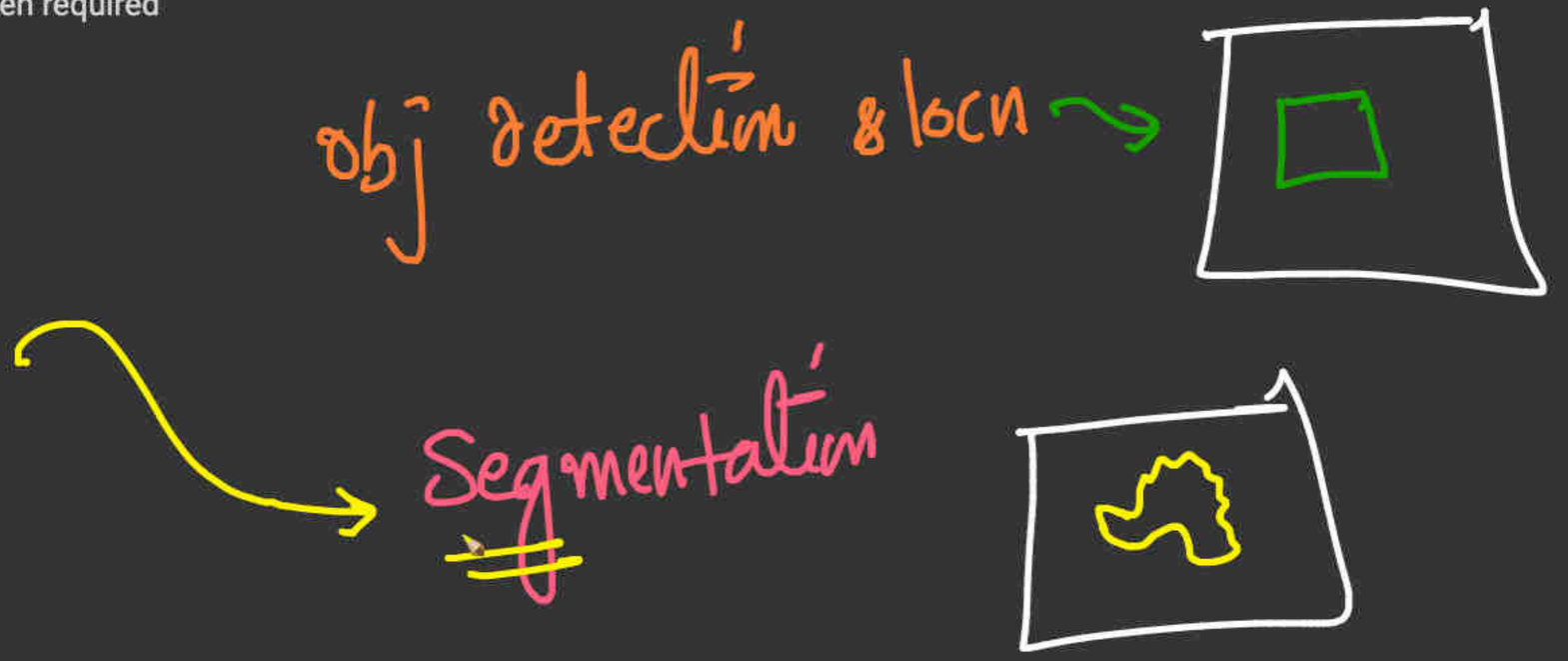
[contrastive learning

- We started with image classification, which is very coarse and high level : whether an object is present in the image or not
- Then in Object Detection we learnt about object localization : where exactly the object is present in the image and we drew a rectangular bounding box around it
- But sometimes bounding boxes are not enough, we need more precision, extract boundary of the object
- For Example Tumor tissue dectection, tissue in the red have quite complex structure and we cannot draw bounding boxes around them
- Today we are going to learn about image segmentation, where we do pixel level analysis of the object
- We will learn to form very complex boundary when required

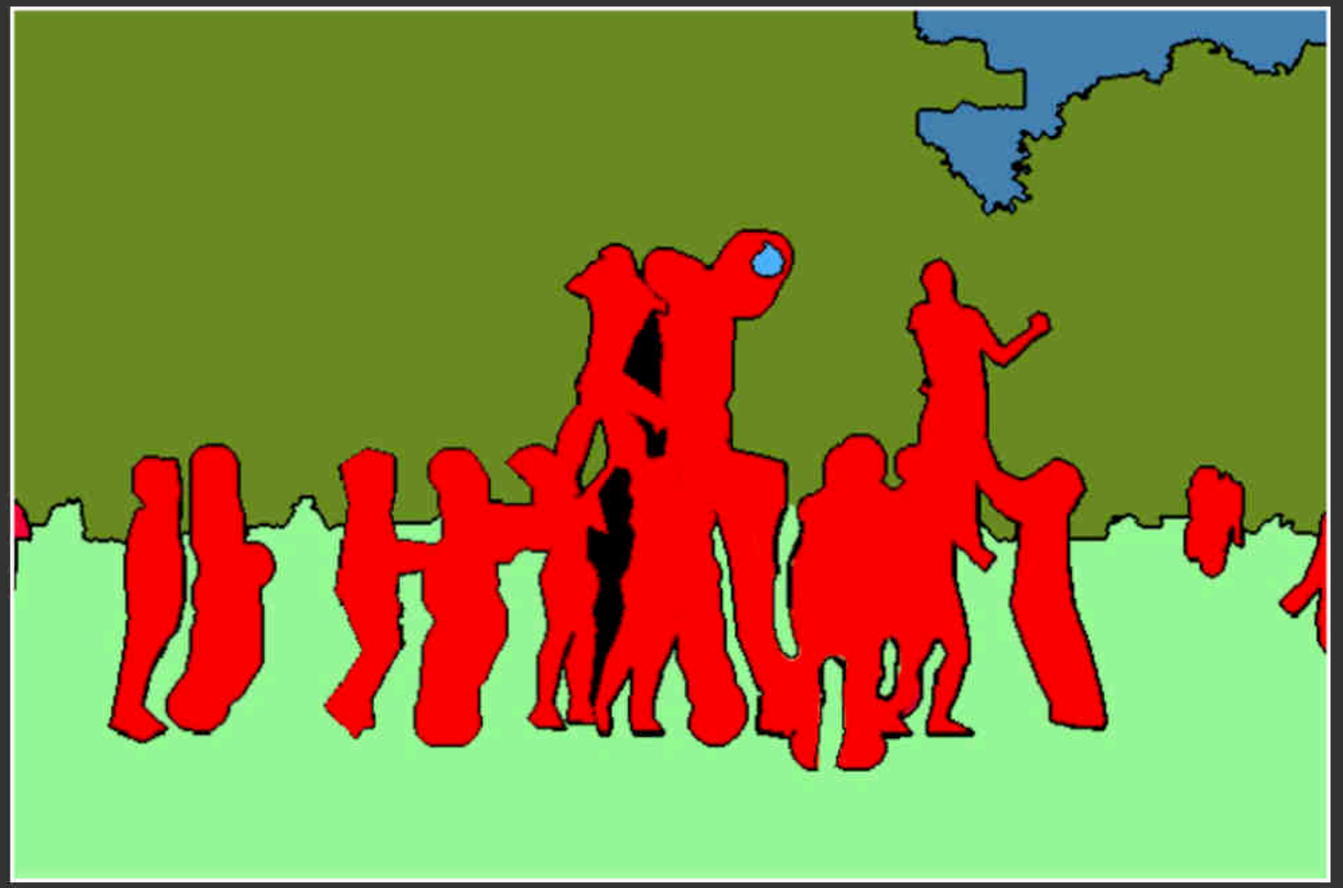


- We started with image classification, which is very coarse and high level : whether an object is present in the image or not
- Then in Object Detection we learnt about object localization : where exactly the object is present in the image and we drew a rectangular bounding box around it
- But sometimes bounding boxes are not enough, we need more precision, extract boundary of the object
- For Example Tumor tissue dectection, tissue in the red have quite complex structure and we cannot draw bounding boxes around them
- Today we are going to learn about image segmentation, where we do pixel level analysis of the object
- We will learn to form very complex boundary when required





- **Semantic Segmentation** : Each pixel is labeled in the image in categories. As shown in the following figure, the image is divided into human (red), tree (dark green), grass (light green), sky (blue).



- **Instance Segmentation**

How does the data looks?

```
[ ] import os
    import cv2
    import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np
    import tensorflow as tf
```

- Now lets look at the dataset of our bussniess case

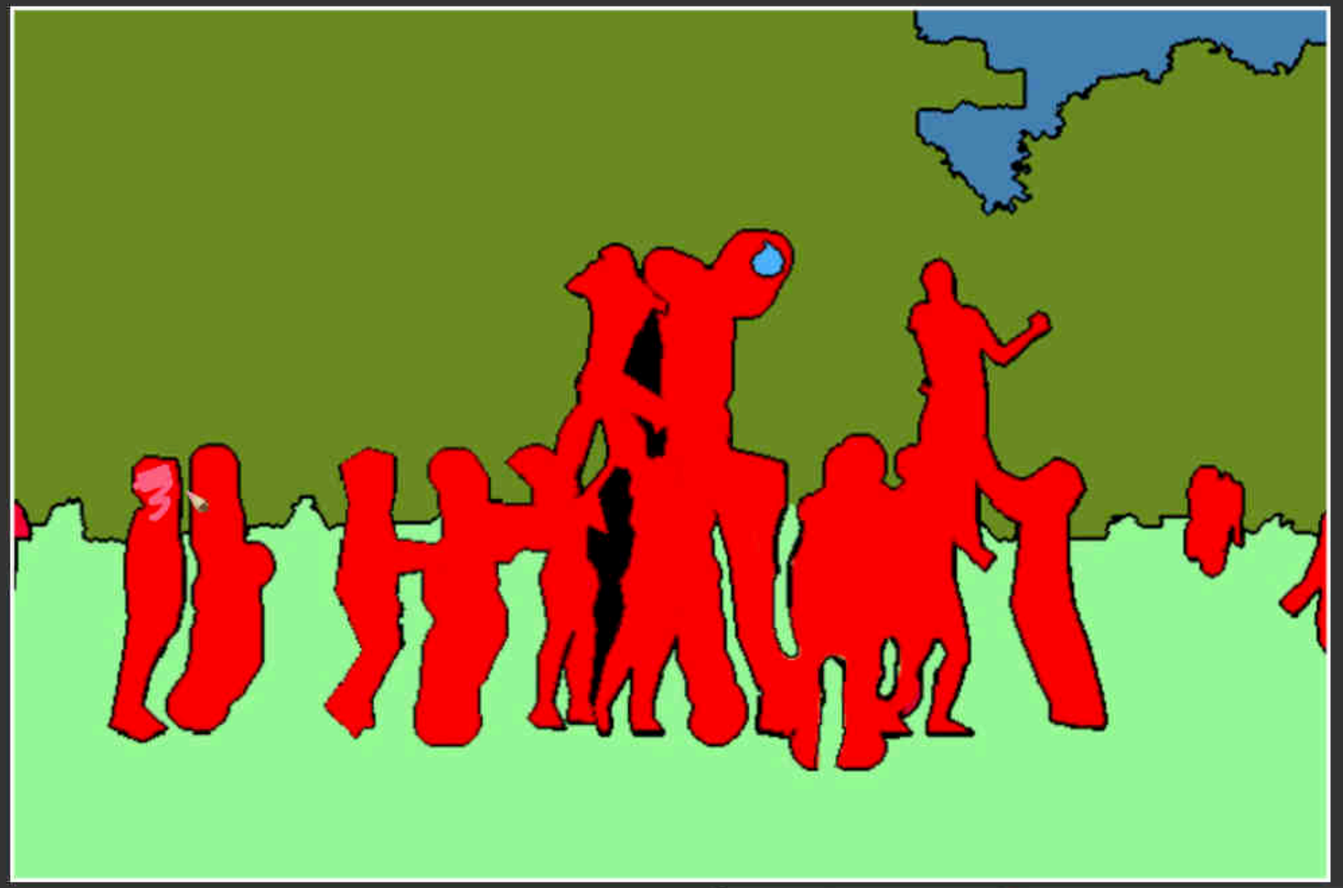
```
[ ] from google.colab import drive
    drive.mount('/content/gdrive/',force_remount=True)
    path ='/content/gdrive/My Drive/Segmentation/'
```

```
Mounted at /content/gdrive/
```

```
[ ] images = np.load(path+'data/img_uint8.npy')

    mask =  np.load(path+'data/msk_uint8.npy')

    print(f'Training X data: {images.shape}, Ground Truth: {mask.shape}')

    print(f'max pixel value: {mask.max()} , min pixel value: {mask.min()}')
```

```
Training X data: (18698, 128, 128, 3)
```

- **Semantic Segmentation** : Each pixel is labeled in the image in categories. As shown in the following figure, the image is divided into human (red), tree (dark green), grass (light green), sky (blue).



How does the data looks?

```
[ ] import os
    import cv2
    import pandas as pd
    import matplotlib.pyplot as plt
    import numpy as np
    import tensorflow as tf
```

- Now lets look at the dataset of our bussniess case

```
[ ] from google.colab import drive
    drive.mount('/content/gdrive/',force_remount=True)
    path ='/content/gdrive/My Drive/Segmentation/'
```

```
Mounted at /content/gdrive/
```

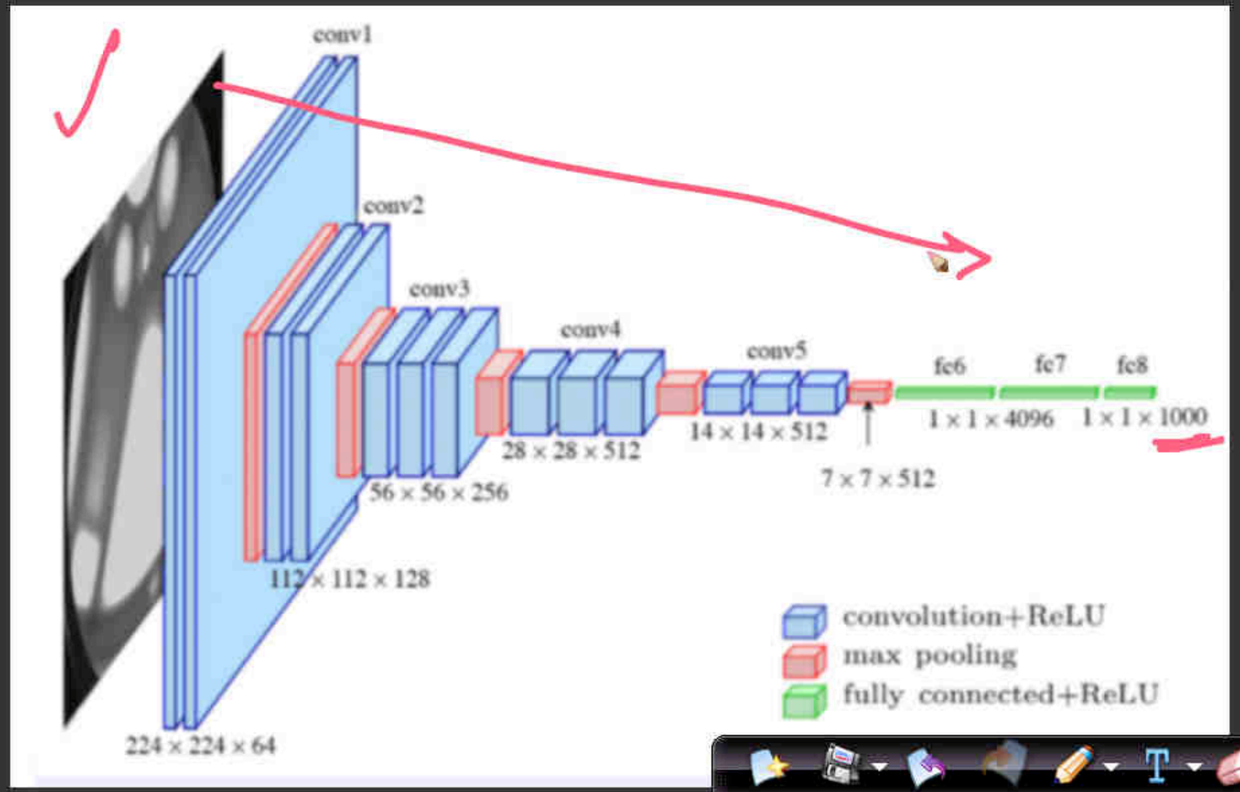
```
[ ] images = np.load(path+'data/img_uint8.npy')

    mask =  np.load(path+'data/msk_uint8.npy')

    print(f'Training X data: {images.shape}, Ground Truth: {mask.shape}')

    print(f'max pixel value: {mask.max()} , min pixel value: {mask.min()}')
```

```
Training X data: (18698, 128, 128, 3)
```

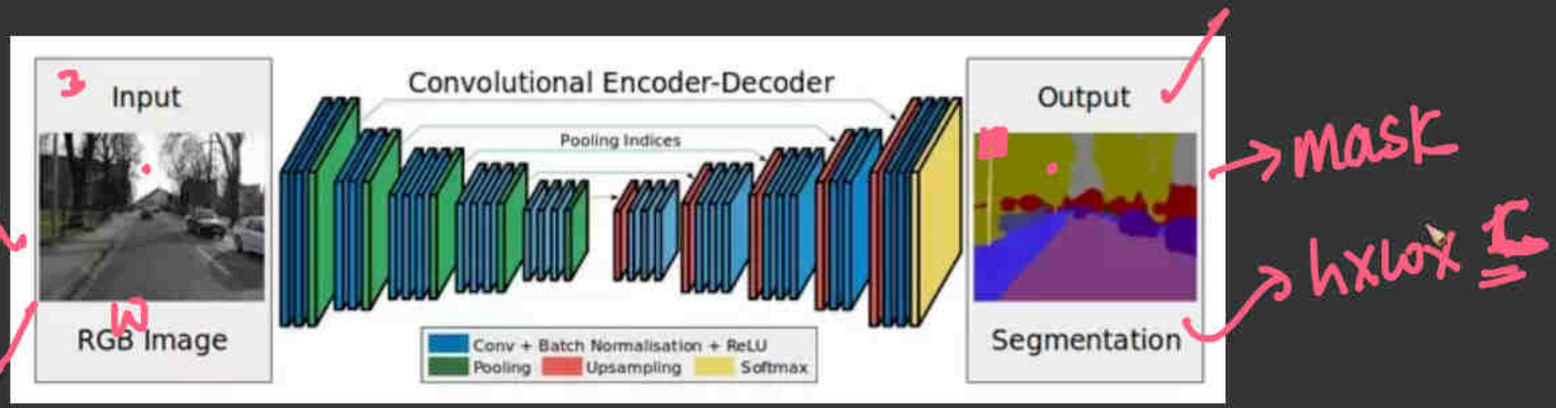
- Instead we can go through every pixel of the image and classify if it is background or not

How should we design the architecture ?

- Here we are predicting labels for each pixel, therefore we need to use a CNN layer with same HxW of the input img shape
- Each class will require one channel, therefore output layer will have dim (bs, H, W, #oflabels)

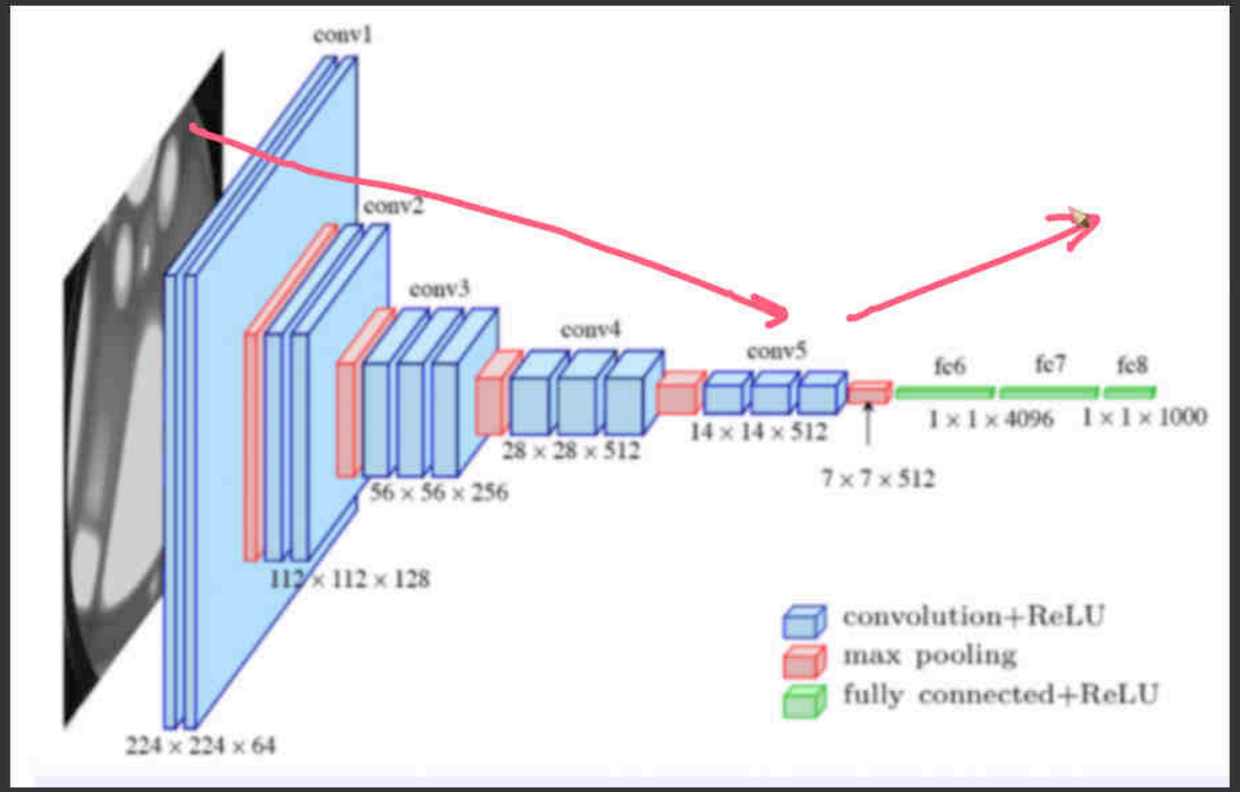


- Final CNN layers conveys Deep, coarse, semantic information: which Answers the “What”
- First few layers conveys Shallow, fine, appearance information: which Answers the “Where”
- In image segmentation we need both "What" and "Where", "What" helps in detecting the label(face/background)
- "Where" helps in dectecting the boundary of the "What", in image classification models we lack the "Where" information
- Therfore we skip connect initial layers with final layer (to infuse "Where")



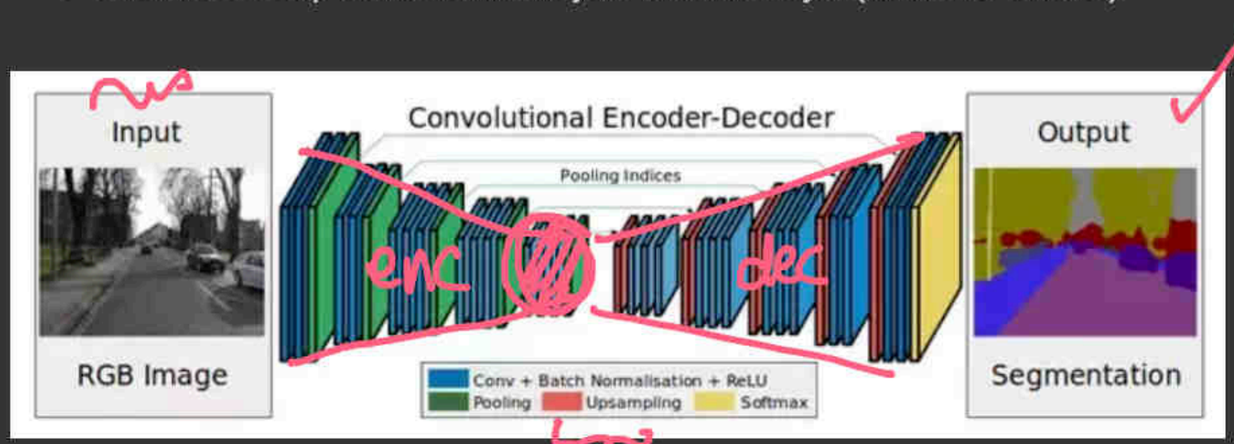
- But How would we increase HxW 7x7 to 224x224?
- [Q] What if we dont use decrease it? Ans : We will lack the "What"
- We use something called Encoder Decoder network
- Encoder part is same as designed for object classification without Flatten/FC layers
- Each encoder layer has a corresponding decoder layers
- Decoder network up-samples its input feature map using transposed Convolutional Layer

- Each class will require one channel, therefore output layer will have dim (bs, H, W, #oflabels)



- Til now we have seen models where HxW dim decreases as we go down the model and number of features increases
- For ex here, HxW ->224 -> 112 -> 56 -> 28 -> 14 ->7 and #offeatures 3 -> 64 -> 128 ->256 ->512
- But we would require 224x224 instead of 7x7 / 1 instead of 512 for our final layer

- Final CNN layers conveys Deep, coarse, semantic information: which Answers the "What"
- First few layers conveys Shallow, fine, appearance information: which Answers the "Where"
- In image segmentation we need both "What" and "Where", "What" helps in detecting the label(face/background)
- "Where" helps in dectecting the boundary of the "What", in image classification models we lack the "Where" information
- Therfore we skip connect initial layers with final layer (to infuse "Where")



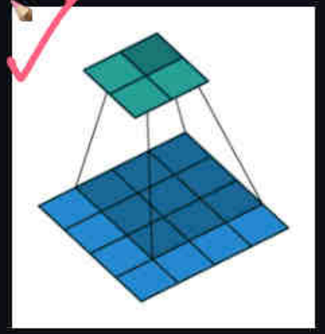
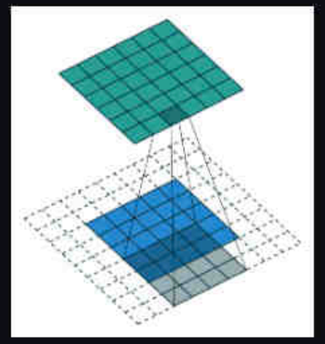
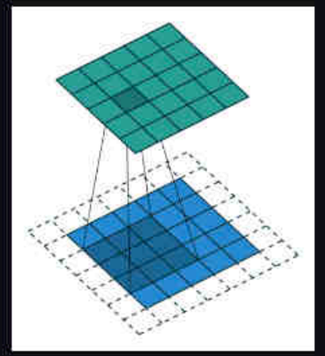
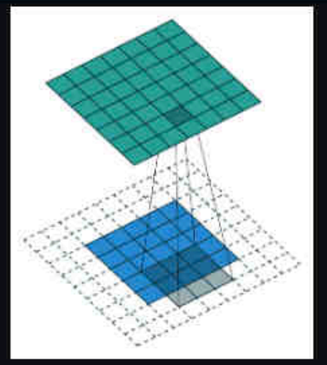
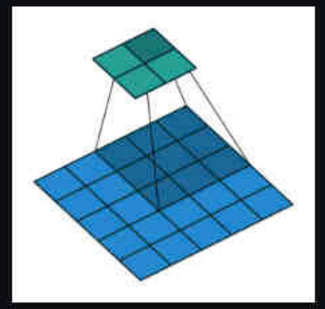
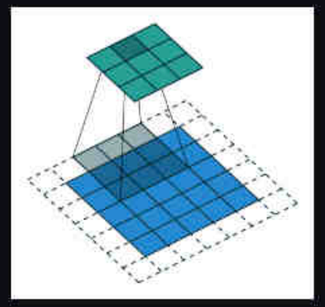
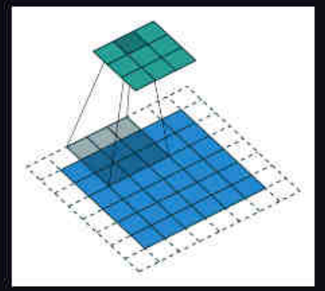
Upsampling

- But How would we increase HxW 7x7 to 224x224?
- [Q] What if we dont use decrease it? Ans : We will lack the "What"
- We use something called Encoder Decoder network
- Encoder part is same as designed for object classification without Flatten/FC layers
- Each encoder layer has a corresponding decoder layers
- Decoder network up-samples its input feature map using transposed Convolutional Layer

README.md

Convolution animations

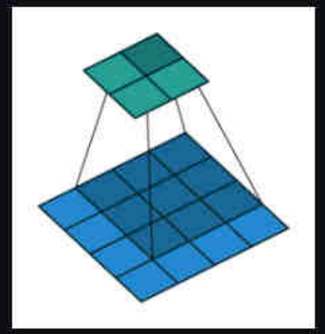
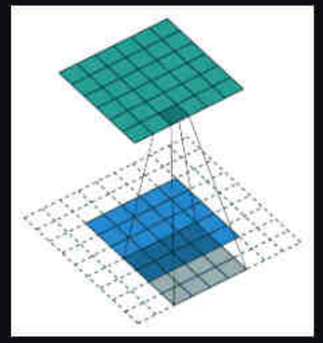
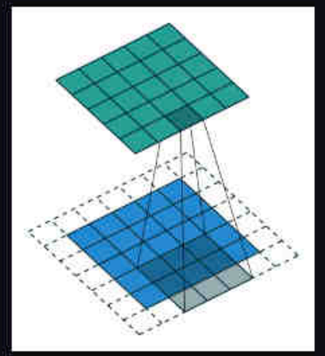
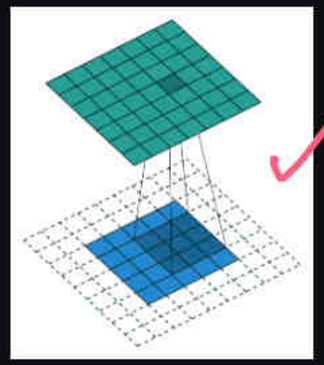
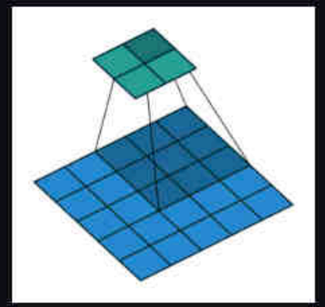
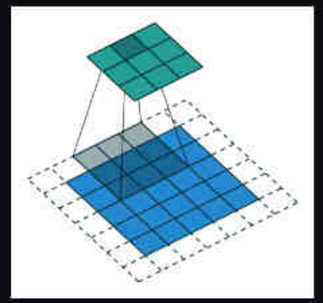
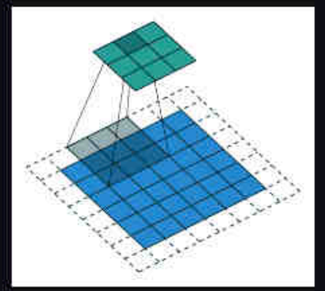
N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

TeX 79.7% Python 20.3%

57 / 57

N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

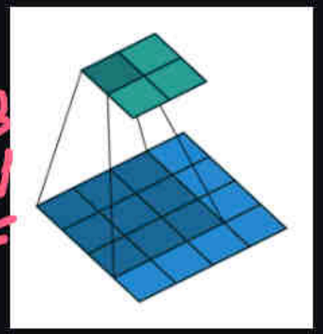
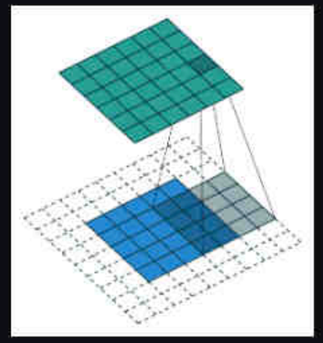
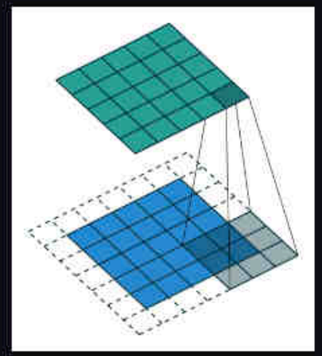
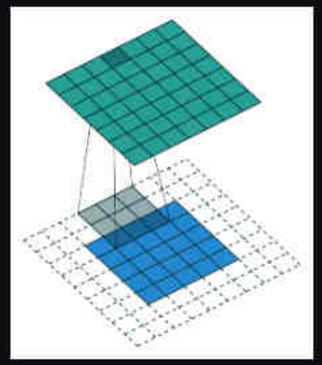
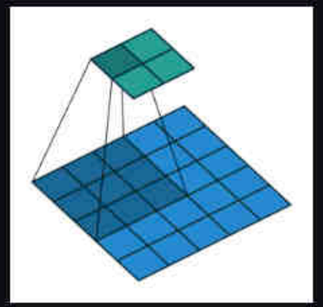
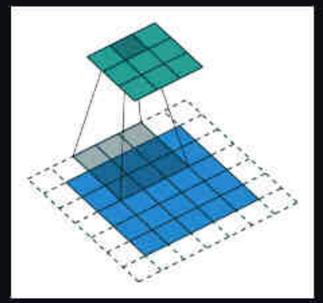
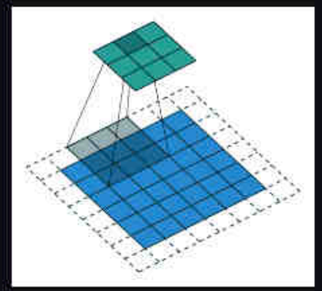
Transposed convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

README.md

N.B.: Blue maps are inputs, and cyan maps are outputs.

3x3 Conv

			
No padding, no strides	Arbitrary padding, no strides	Half padding, no strides	Full padding, no strides
			
No padding, strides	Padding, strides	Padding, strides (odd)	

Transposed convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

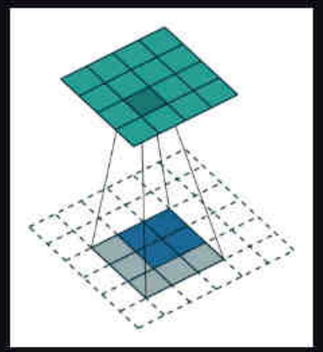
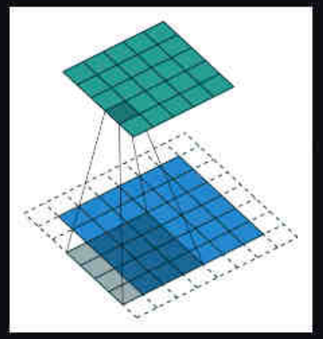
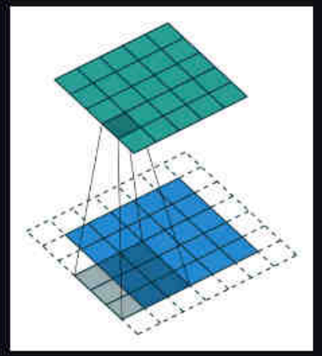
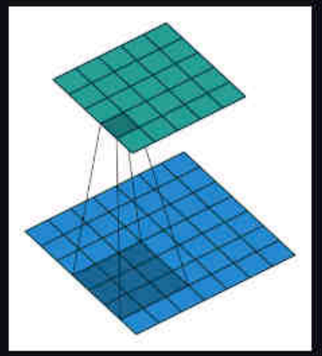
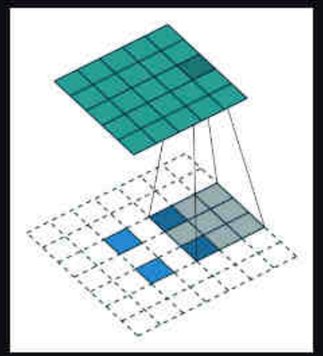
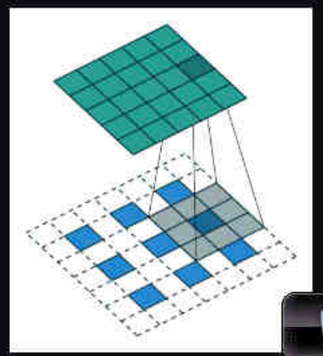
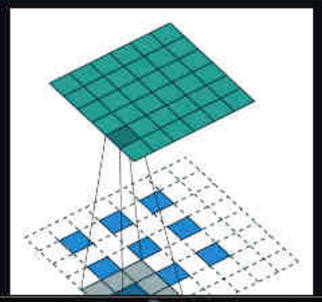
59 / 59

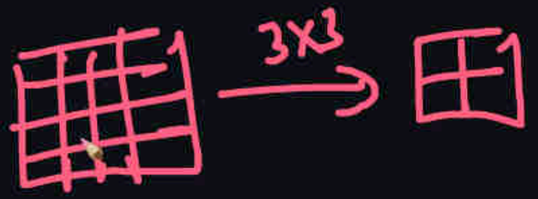
README.md

No padding, strides	Padding, strides	Padding, strides (odd)	

Transposed convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

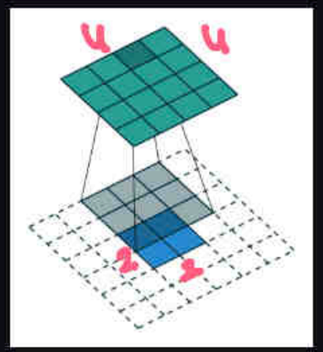
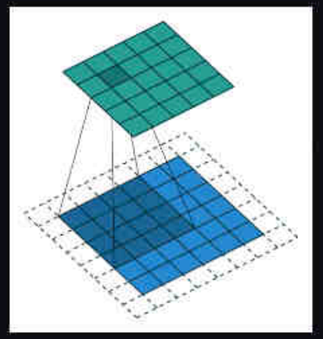
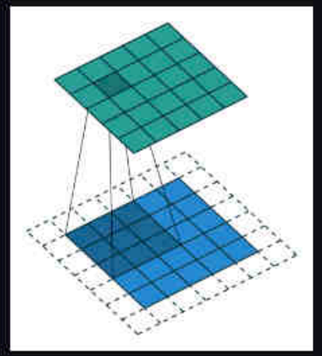
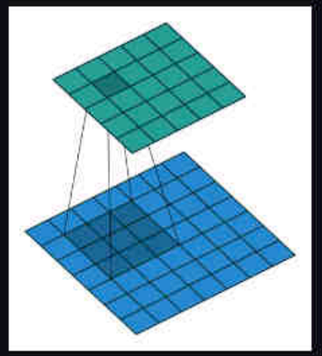
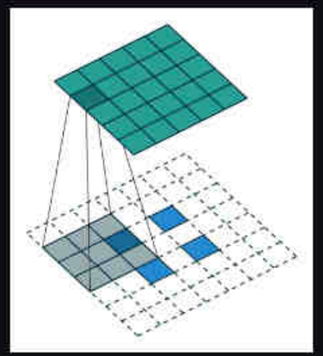
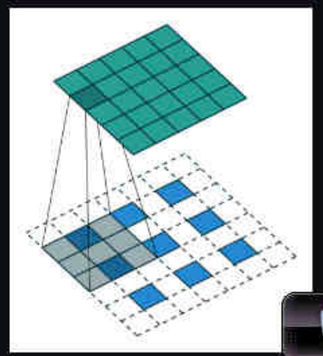
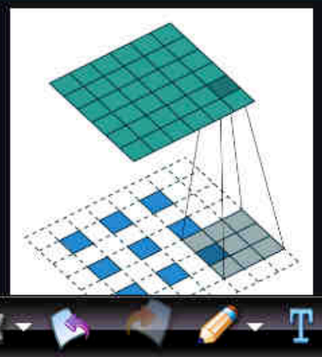
			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			

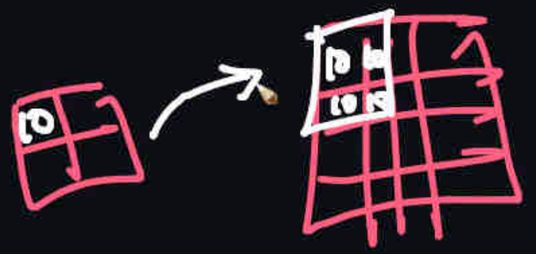


No padding, strides	Padding, strides	Padding, strides (odd)	

Transposed convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			

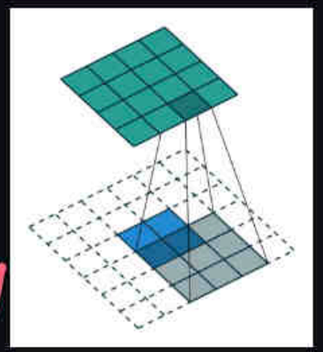
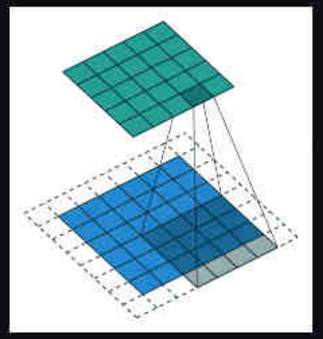
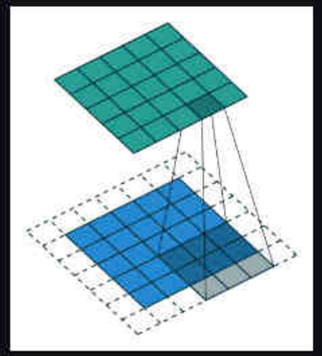
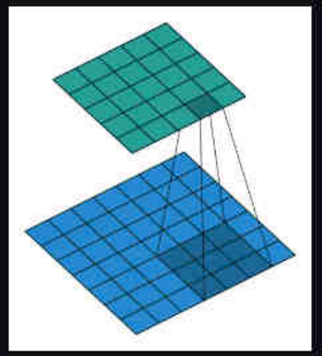
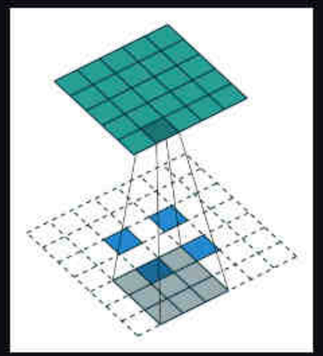
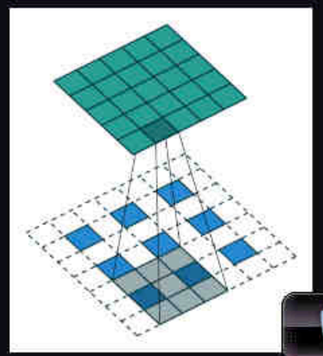
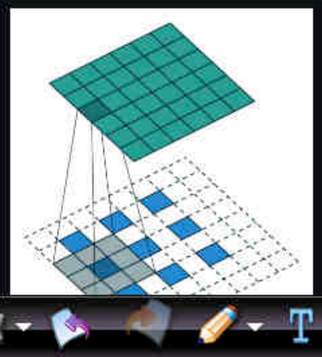


README.md

No padding, strides	Padding, strides	Padding, strides (odd)	

Transposed convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			

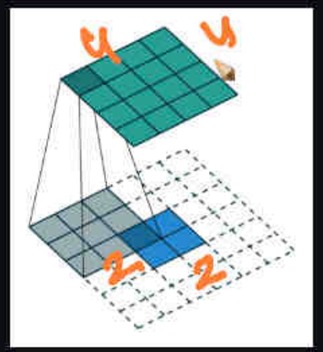
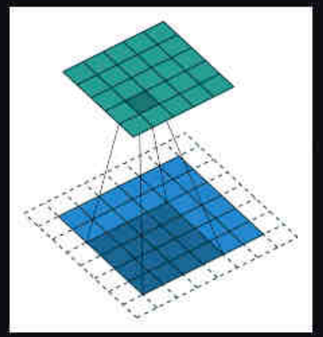
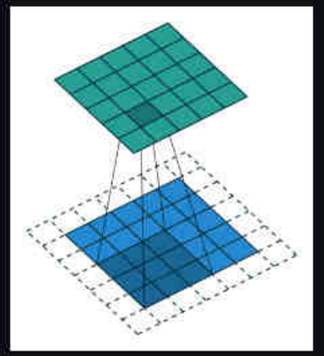
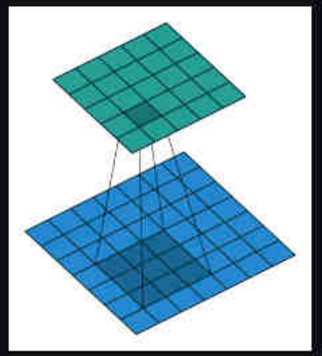
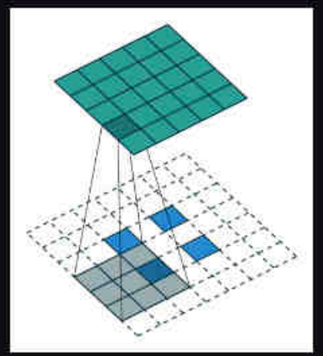
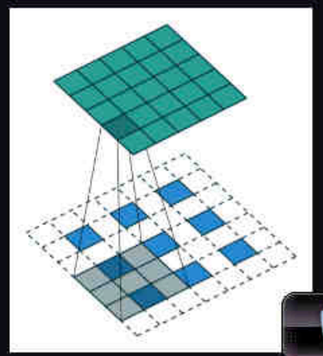
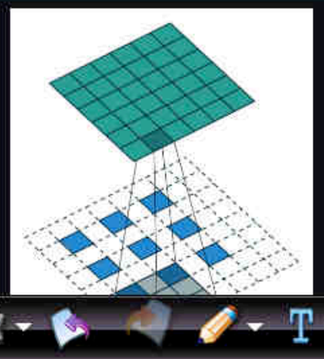
62 / 62

README.md

No padding, strides	Padding, strides	Padding, strides (odd)	

Transposed convolution animations

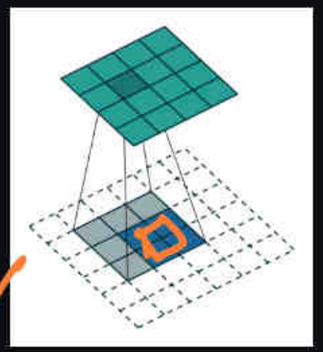
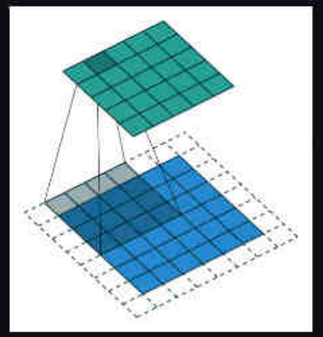
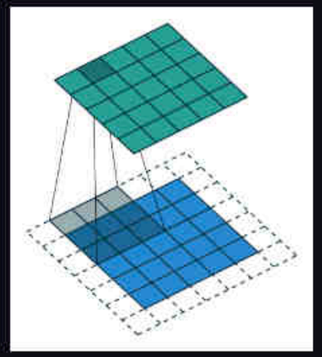
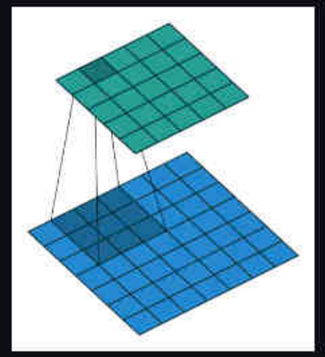
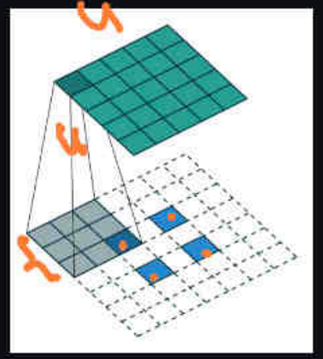
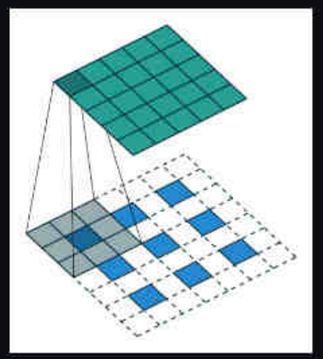
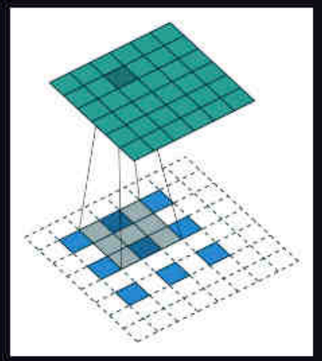
N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			

63 / 63

README.md

N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	Padding, strides,	

3x3

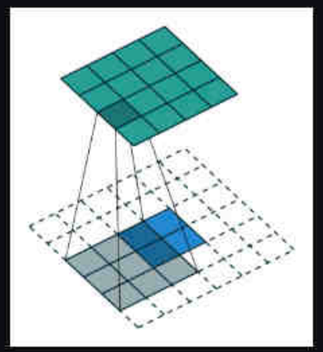
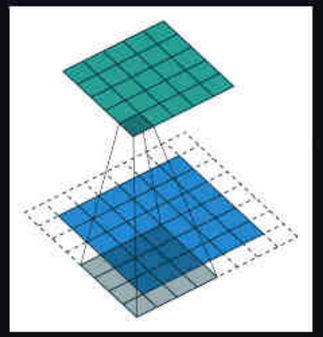
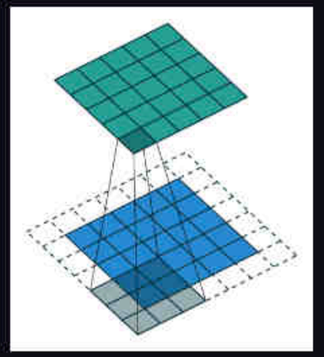
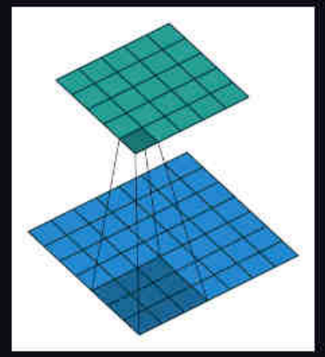
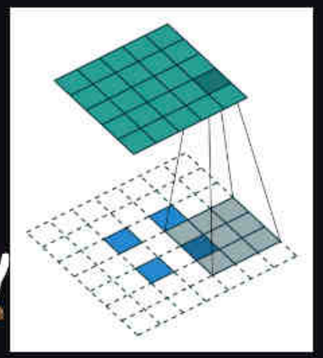
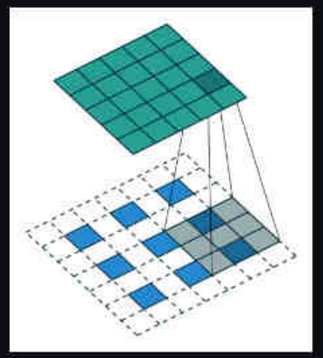
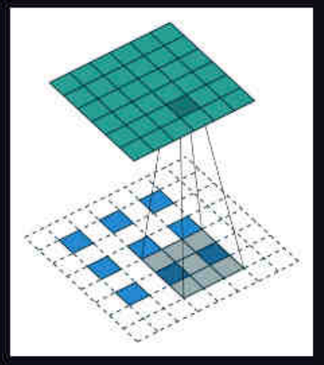
Conv t→s

Tr.conv 3x3

s→l

README.md

N.B.: Blue maps are inputs, and cyan maps are outputs.

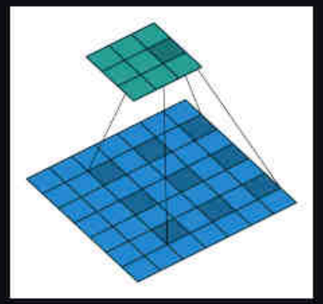
			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	Padding, strides,	

65 / 65

README.md

Dilated convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.



No padding, no stride, dilation

Generating the Makefile

From the repository's root directory:

```
$ ./bin/generate_makefile
```

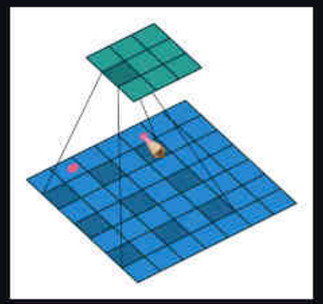
Generating the animations

README.md

No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

Dilated convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.


No padding, no stride, dilation

Generating the Makefile

From the repository's root directory:

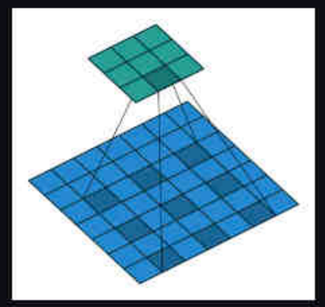
```
$ ./bin/generate_makefile
```

README.md

No padding, strides, transposed	Padding, strides, transposed	Padding, strides, transposed (odd)	

Dilated convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.



3x3 conv dilated

No padding, no stride, dilation

Generating the Makefile

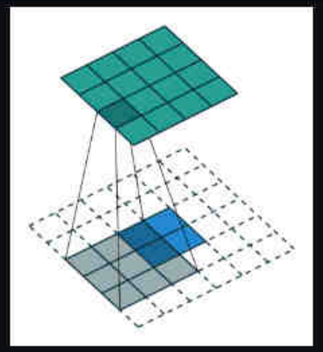
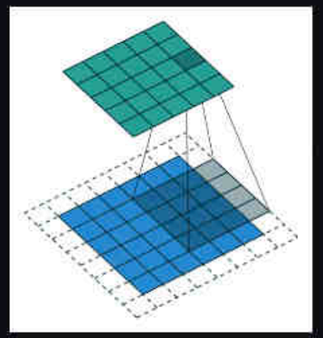
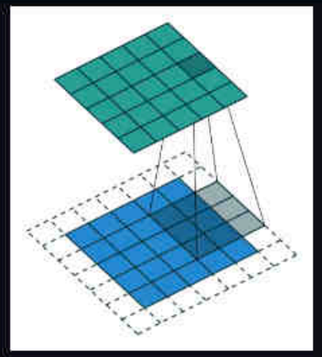
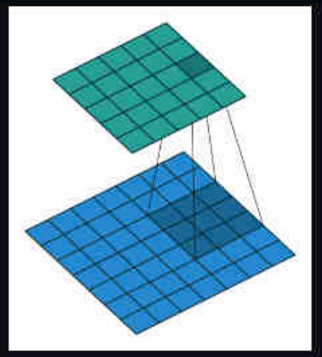
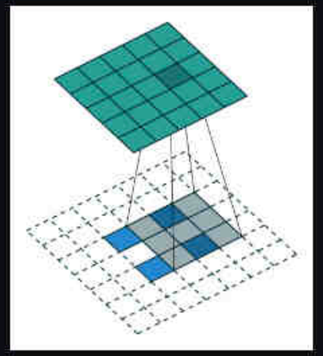
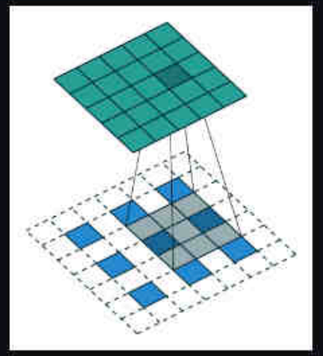
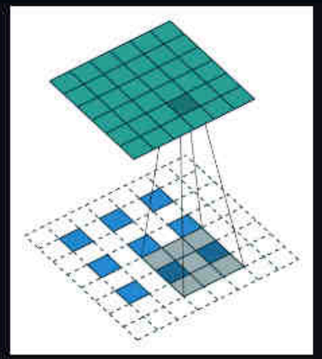
From the repository's root directory:

```
$ ./bin/generate_makefile
```

README.md

Transposed convolution animations

N.B.: Blue maps are inputs, and cyan maps are outputs.

			
No padding, no strides, transposed	Arbitrary padding, no strides, transposed	Half padding, no strides, transposed	Full padding, no strides, transposed
			
No padding, strides, transposed	Padding, strides, transposed	transposed (odd)	

How we will evaluate Image Segmentation Models?

- We classify pixels of an image as 1 or 0. If there is a mask in a pixel we state 1, if there is not a mask (background) we state 0.
- Making pixelwise binary classification of image

- we can formulate it a two-class semantic segmentation problem, with binary cross entropy loss as the loss function.
- . However, it works badly because of two reasons:
  - highly unbalanced label distribution and
  - per-pixel intrinsic issue of cross entropy loss.
- When using cross entropy loss, the statistical distributions of labels play a big role in training accuracy.
- The more unbalanced the label distributions are, the more difficult the training will be.
- the loss is calculated as the average of per-pixel loss, without knowing whether its adjacent pixels are boundaries or not.
- As a result, cross entropy loss only considers loss in a micro sense rather than considering it globally, which is not enough for image level prediction.

- So we need a new loss that can handle class imblance and consider a macro sense of prediction distribution

- **IOU**
- We have discussed IOU in object detecion

How we will evaluate Image Segmentation Models?

→ per pixel

metric

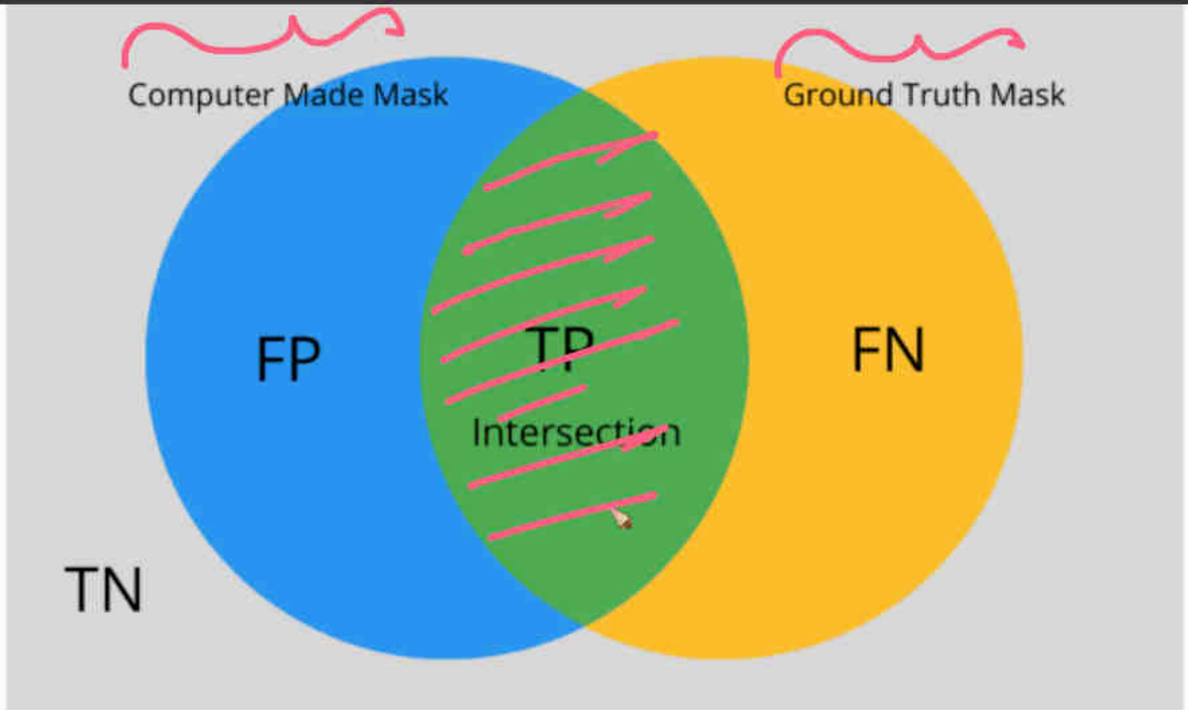
~ IoU

- We classify pixels of an image as 1 or 0. If there is a mask in a pixel we state 1, if there is not a mask (background) we state 0.
- Making pixelwise binary classification of image

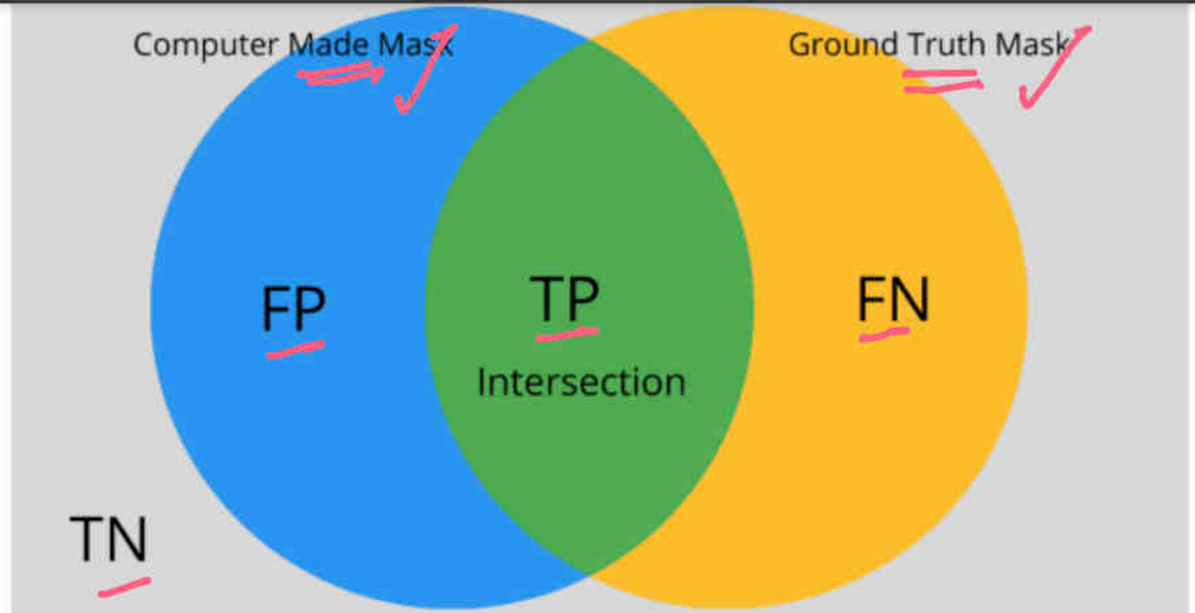
- we can formulate it a two-class semantic segmentation problem, with binary cross entropy loss as the loss function.
- . However, it works badly because of two reasons:
  - highly unbalanced label distribution and
  - per-pixel intrinsic issue of cross entropy loss.
- When using cross entropy loss, the statistical distributions of labels play a big role in training accuracy.
- The more unbalanced the label distributions are, the more difficult the training will be.
- the loss is calculated as the average of per-pixel loss, without knowing whether its adjacent pixels are boundaries or not.
- As a result, cross entropy loss only considers loss in a micro sense rather than considering it globally, which is not enough for image level prediction.

- So we need a new loss that can handle class imblance and consider a macro sense of prediction distribution

- **IOU**
- We have discussed IOU in object detecion



$$\frac{\text{Intersection}}{\text{Union}} = \frac{\text{TP}}{\text{TP + FN + FP}}$$



$$\frac{\text{Intersection}}{\text{Union}} = \frac{\text{TP}}{\text{TP + FN + FP}}$$

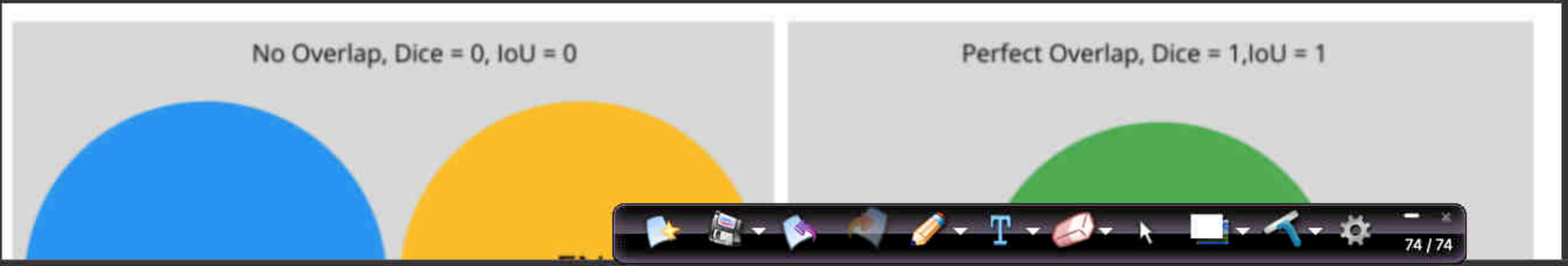
= IoU

- IOU can be used in image segmentation, but we use a slightly different metric **Dice**
- Dice coefficient is very similar to IOU. Dice c

$$\frac{\text{Intersection}}{\text{Union}} = \frac{2\,TP}{2TP + FN + FP}$$

- IOU can be used in image segmentation, but we use a slightly different metric **Dice**
- Dice coefficient is very similar to IOU. Dice coefficient double counts the intersection(TP).

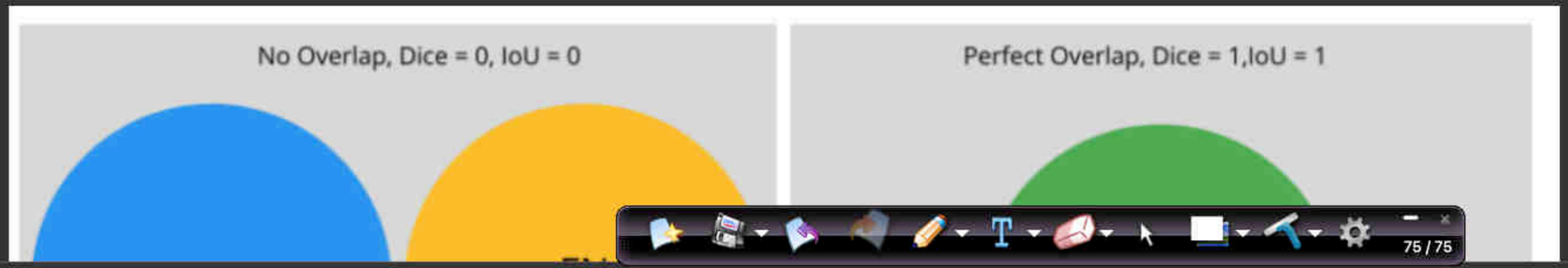
$$\text{Dice Coefficient} = \frac{2\text{xIntersection}}{\text{Union} + \text{Intersection}} = \frac{2TP}{2TP + FN + FP}$$



$$\frac{\text{Intersection}}{\text{Union}} = \frac{\text{TP}}{\text{TP + FN + FP}}$$

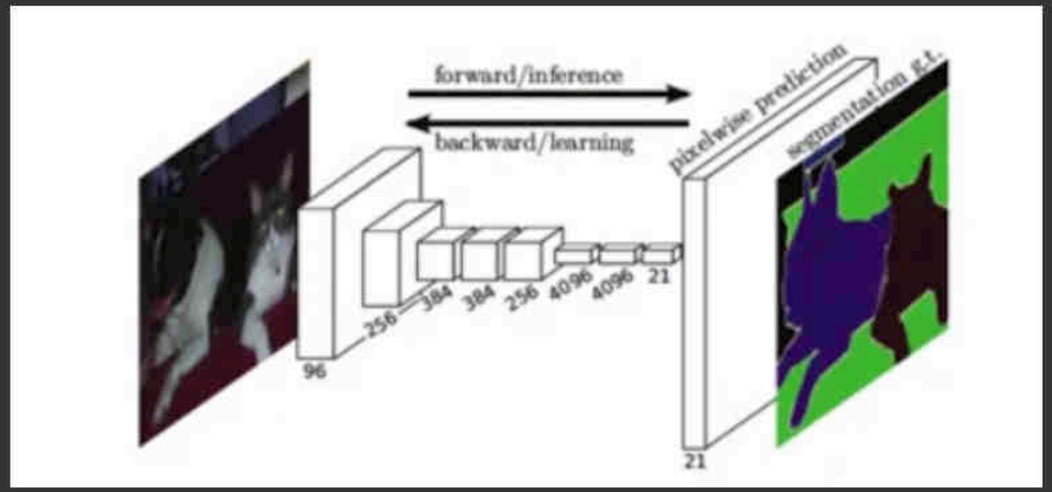
- IOU can be used in image segmentation, but we use a slightly different metric **Dice**
- Dice coefficient is very similar to IOU. Dice coefficient double counts the intersection(TP).

$$\text{Dice Coefficient} = \frac{2\text{xIntersection}}{\text{Union + Intersection}} = \frac{2\text{TP}}{2\text{TP + FN + FP}}$$



FCN Coding Tutorial

- With This, we will dive deeper into the Convolutional Encoder-Decoder models by implementing a Fully Convolutional Neural Networks from scratch



How does the model understand the key features of the image ?

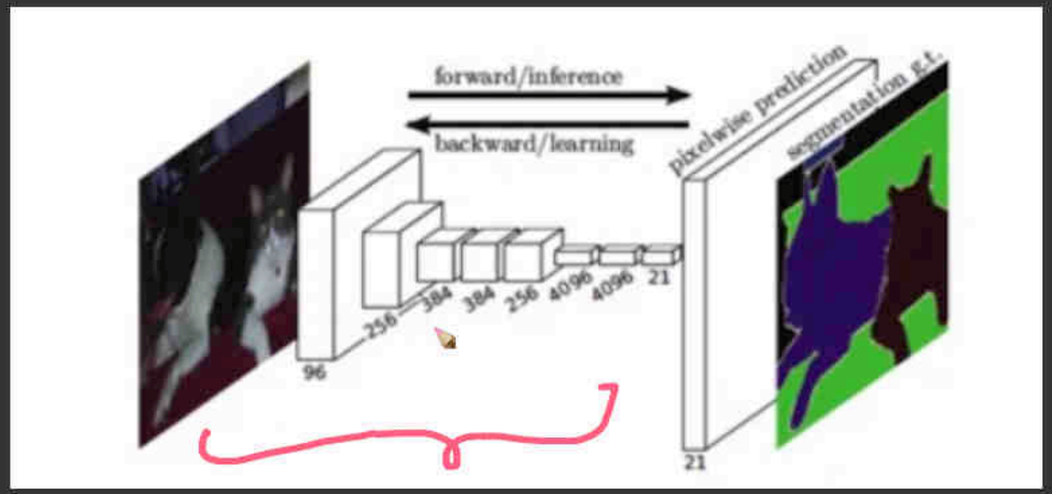
- The FCN model uses convolutional layers as feature extractor
- When used in the Encoder part of the FCN model by downsampling the image

Now, the last layers contains all the key features we pass it to a decoder part of the model instead of Fully Connected Layers

- The decoder consists of Deconvolutional layers that upsamples these key features to the original size of the image

FCN Coding Tutorial

- With This, we will dive deeper into the Convolutional Encoder-Decoder models by implementing a Fully Convolutional Neural Networks from scratch



How does the model understand the key features of the image ?

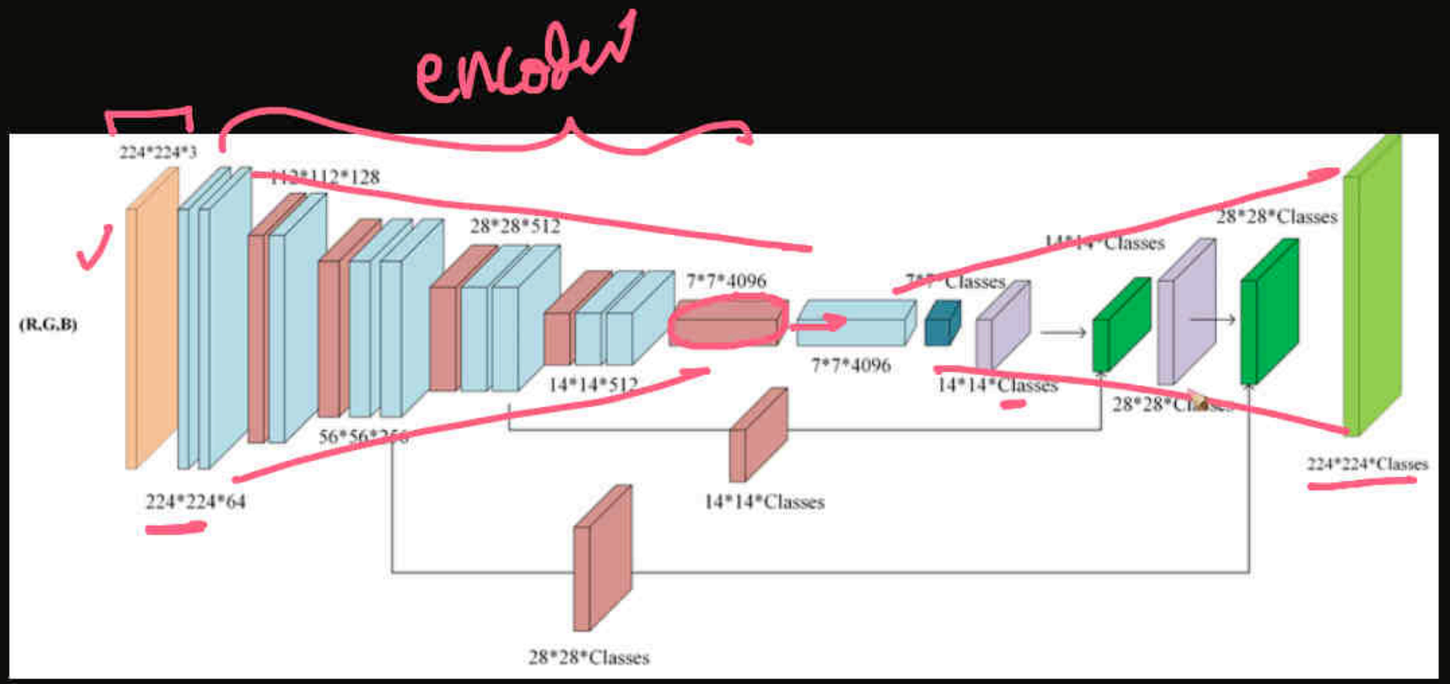
- The FCN model uses convolutional layers as feature extractor
- When used in the Encoder part of the FCN model by downsampling the image

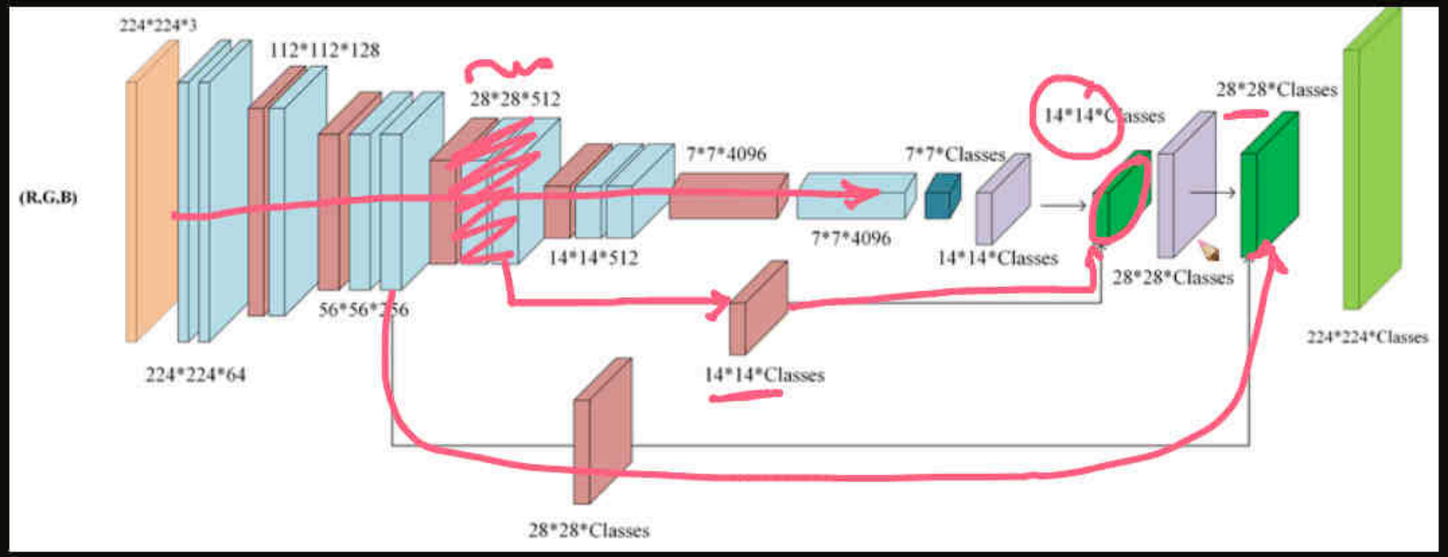
Now, the last layers contains all the key features we pass it to a decoder part of the model instead of Fully Connected Layers

- The decoder consists of Deconvolutional la ge

```
!rm -rf ./logs/
```

```
def FCN8():
    VGG16 = tf.keras.applications.VGG16(weights="imagenet", include_top=False,input_tensor=tf.keras.Input(shape=(128, 128, 3)))
    VGG16.trainable  = False
    x = VGG16(VGG16.input)

    #Decoder
    o = tf.keras.layers.Conv2D( 4096 , ( 7 , 7 ) , activation='relu' , padding='same', name="conv6")(x)
    conv7 = tf.keras.layers.Conv2D( 4096 , ( 1 , 1 ) , activation='relu' , padding='same', name="conv7")(o)
    print(o.shape,conv7.shape)
    ## 4 times upsamping for pool4 layer
    conv7_4 = tf.keras.layers.Conv2DTranspose( 2 , kernel_size=(4,4) ,  strides=(4,4) , use_bias=False)(conv7)

    ## 2 times upsampling for pool411

    pool4 = VGG16.get_layer('block4_pool').output
    pool411 = tf.keras.layers.Conv2D( 2 , ( 1 , 1 ) , activation='relu' , padding='same', name="pool4_11")(pool4)
    pool411_2 = tf.keras.layers.Conv2DTranspose(2 , kernel_size=(2,2) ,  strides=(2,2) , use_bias=False)(pool411)

    pool3 = VGG16.get_layer('block3_pool').output
    pool311 = tf.keras.layers.Conv2D(2 , ( 1 , 1 ) , activation='relu' , padding='same', name="pool3_11")(pool3)
    o = tf.keras.layers.Add(name="add")([pool411_2, pool311])

    print(o.shape,conv7_4.shape)
    o = tf.keras.layers.Add(name="add1")([o, conv7_4 ])
    o = tf.keras.layers.Conv2DTranspose(2 , kernel_size=(8,8) ,  strides=(8,8) , use_bias=False)(o)
    o = tf.keras.layers.Activation('softmax')(o)

    model = tf.keras.Model(VGG16.inpu
```

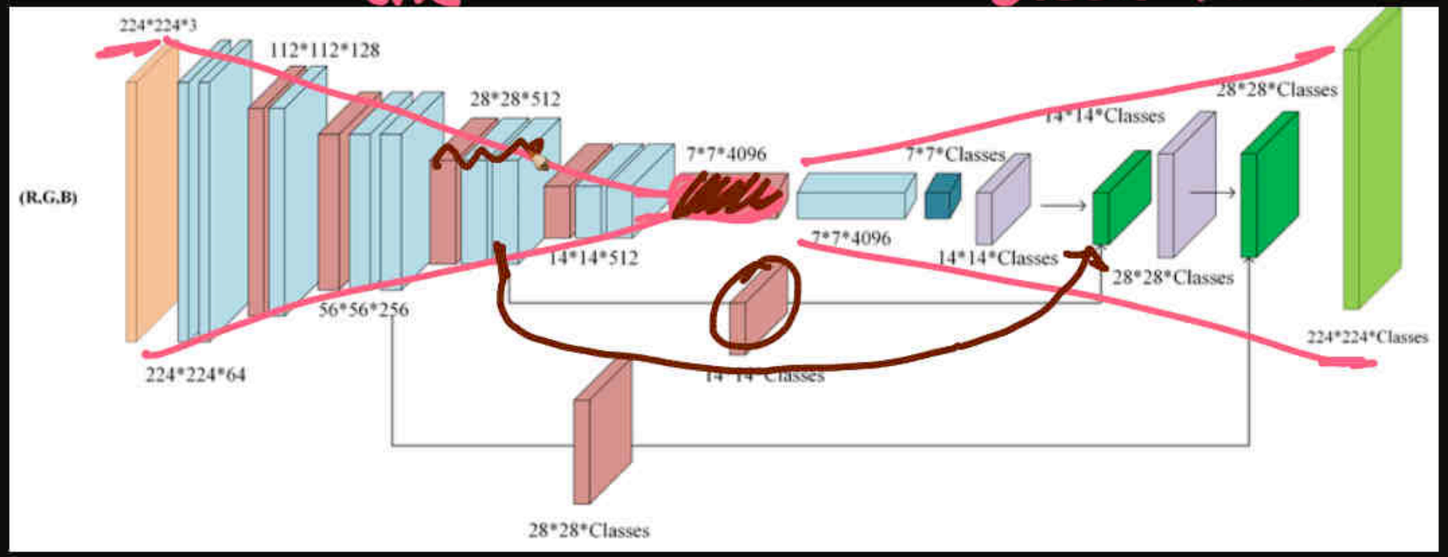


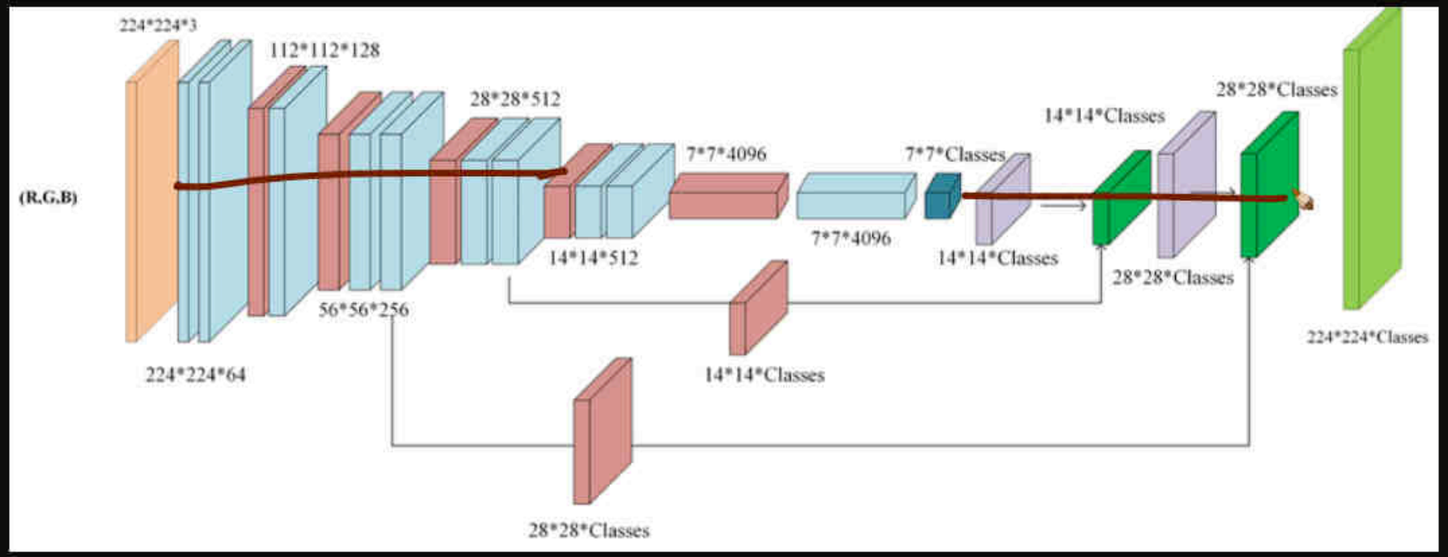


```
VGG16.trainable  = False
x = VGG16(VGG16.input)

#Decoder
o = tf.keras.layers.Conv2D( 4096 , ( 7 , 7 ) , activation='relu' , padding='same', name="conv6")(x)
conv7 = tf.keras.layers.Conv2D( 4096 , ( 1 , 1 ) , activation='relu' , padding='same', name="conv7")(o)
print(o.shape,conv7.shape)
## 4 times upsamping for pool4 layer
conv7_4 = tf.keras.layers.Conv2DTranspose( 2 , kernel_size=(4,4) ,  strides=(4,4) , use_bias=False)(conv7)

## 2 times upsampling for pool411

pool4 = VGG16.get_layer('block4_pool').output
pool411 = tf.keras.layers.Conv2D( 2 , ( 1 , 1 ) , activation='relu' , padding='same', name="pool4_11")(pool4)
pool411_2 = tf.keras.layers.Conv2DTranspose(2 , kernel_size=(2,2) ,  strides=(2,2) , use_bias=False)(pool411)

pool3 = VGG16.get_layer('block3_pool').output
pool311 = tf.keras.layers.Conv2D(2 , ( 1 , 1 ) , activation='relu' , padding='same', name="pool3_11")(pool3)
o = tf.keras.layers.Add(name="add")([pool411_2, pool311])

print(o.shape,conv7_4.shape)
o = tf.keras.layers.Add(name="add1")([o, conv7_4 ])
o = tf.keras.layers.Conv2DTranspose(2 , kernel_size=(8,8) ,  strides=(8,8) , use_bias=False)(o)
o = tf.keras.layers.Activation('softmax')(o)

model = tf.keras.Model(VGG16.input, o)

return model
```

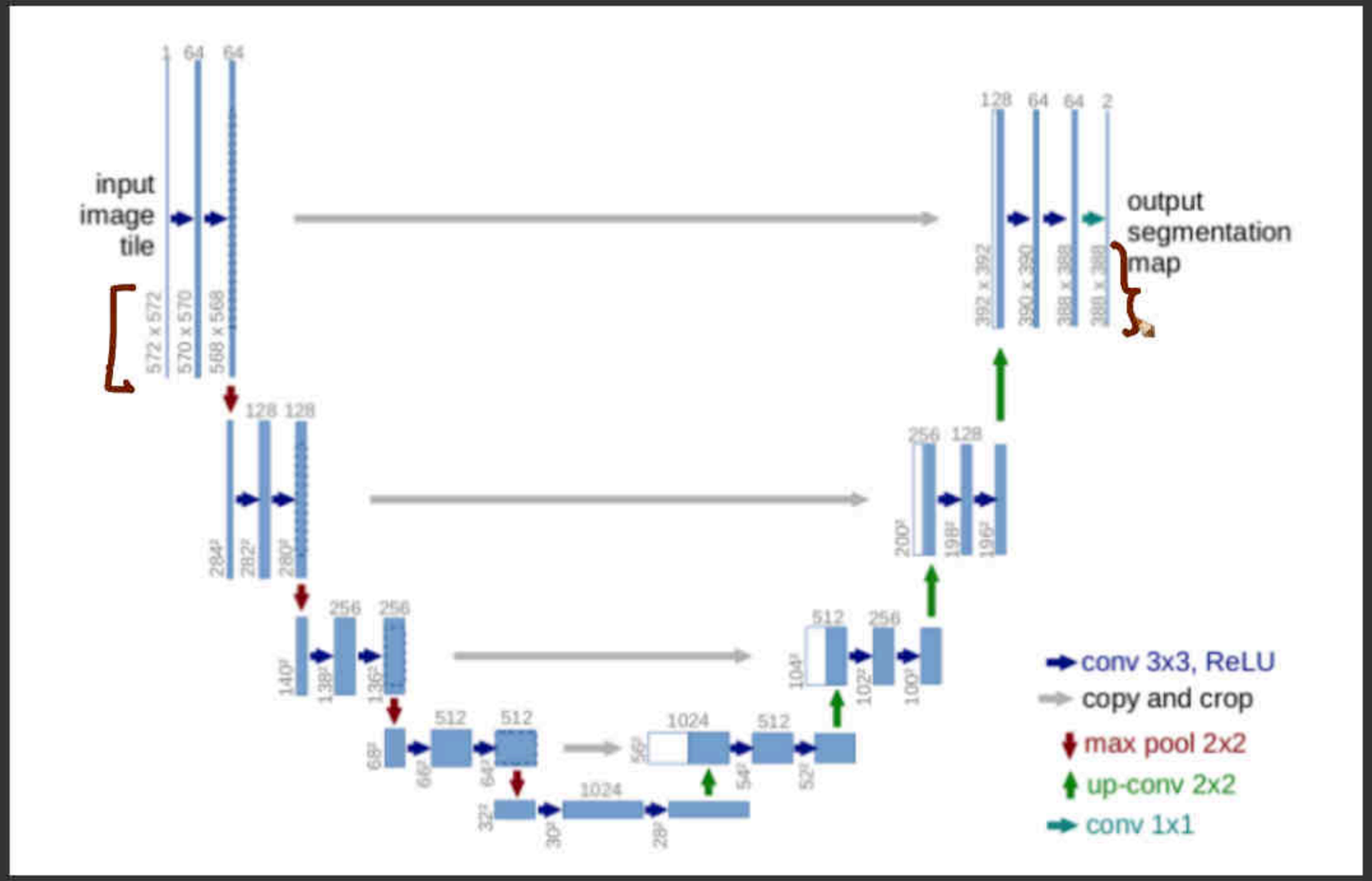
enc

dec: Tr. conv




224*224*3
112*112*128
28*28*512
7*7*4096
7*7*Classes
14*14*Classes
28*28*Classes
(R.G.B)
7*7*4096
14*14*512
14*14*Classes
28*28*Classes
56*56*256
224*224*Classes
224*224*64
14*14*Classes
28*28*Classes

- Original paper: https://arxiv.org/pdf/1505.04597.pdf

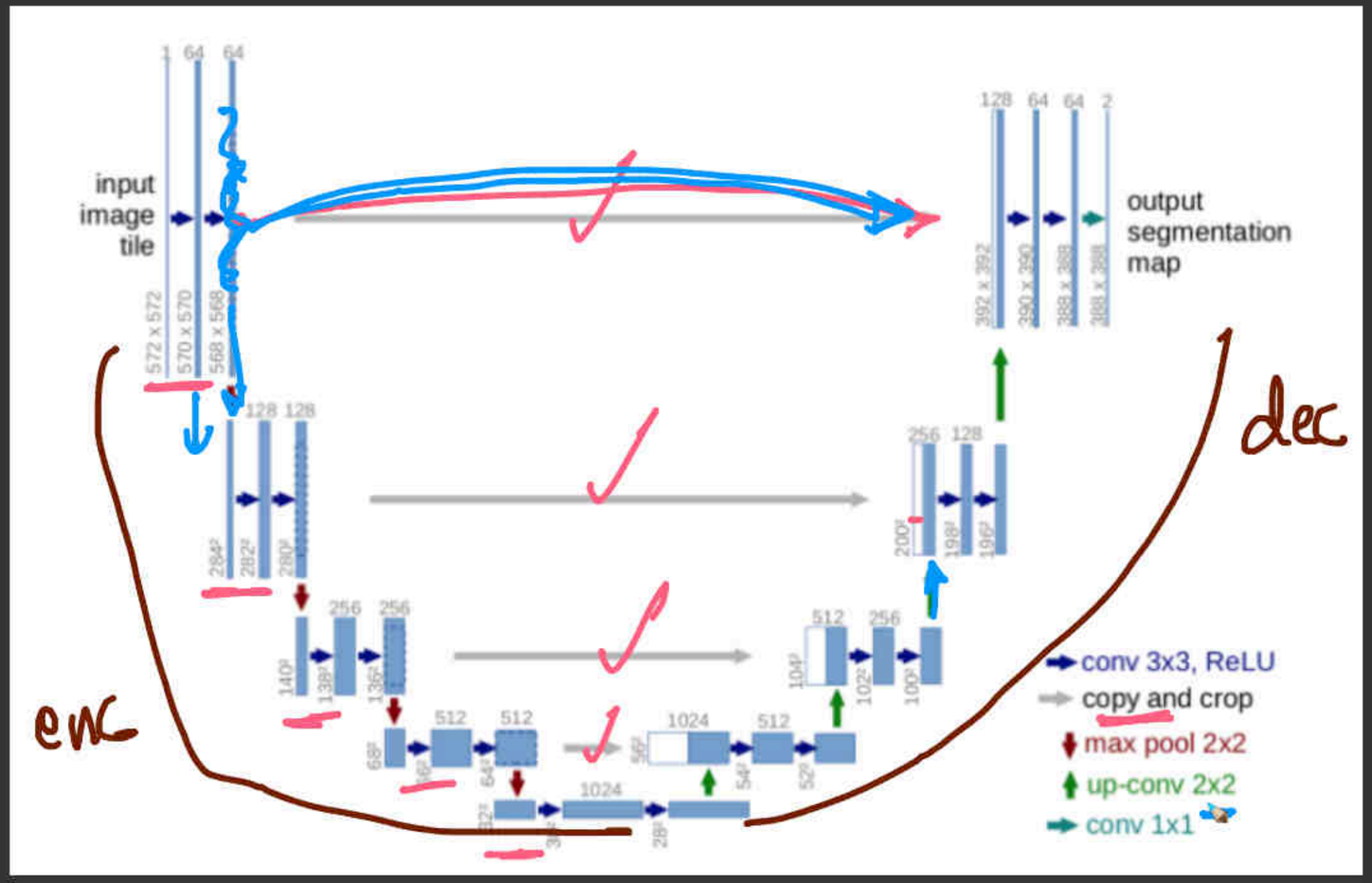


What can we notice ?

FCN model.

- Original paper: https://arxiv.org/pdf/1505.04597.pdf

U-net



enc

dec

① upsamplings : cheaper

② many more skip-connections

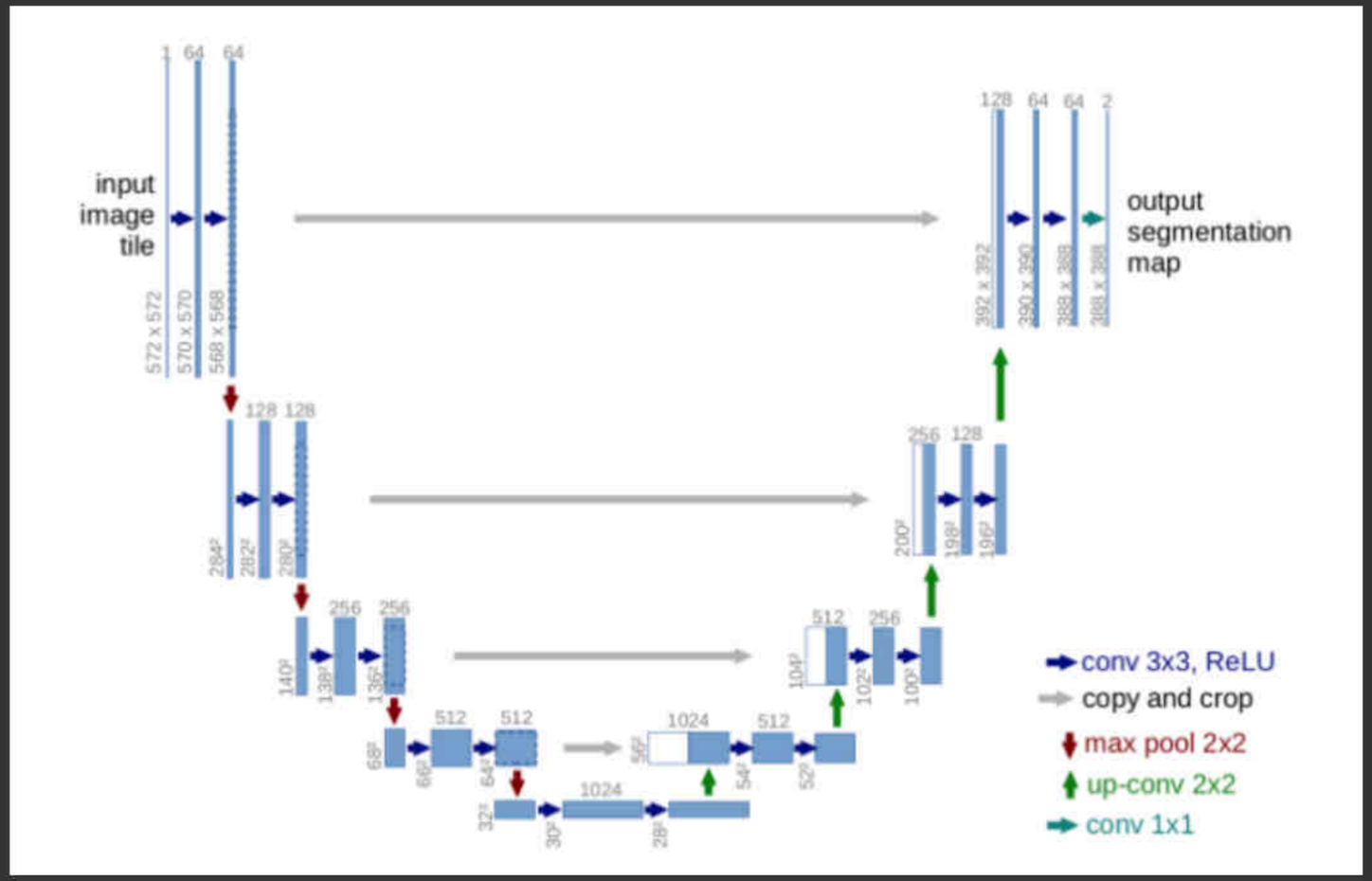
Copy + crop

What can we notice ?

85 / 86

FCN model.

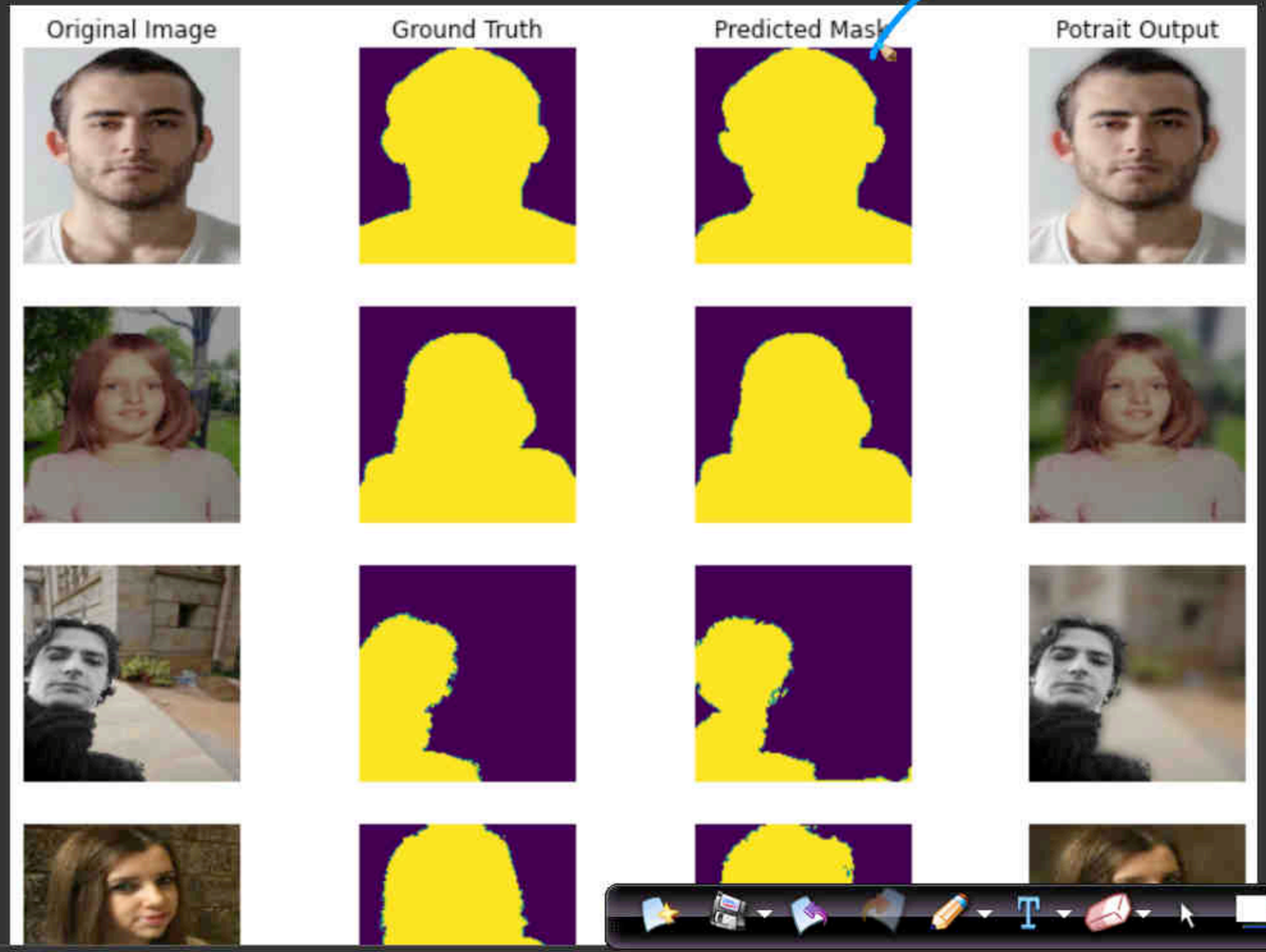
- Original paper: https://arxiv.org/pdf/1505.04597.pdf



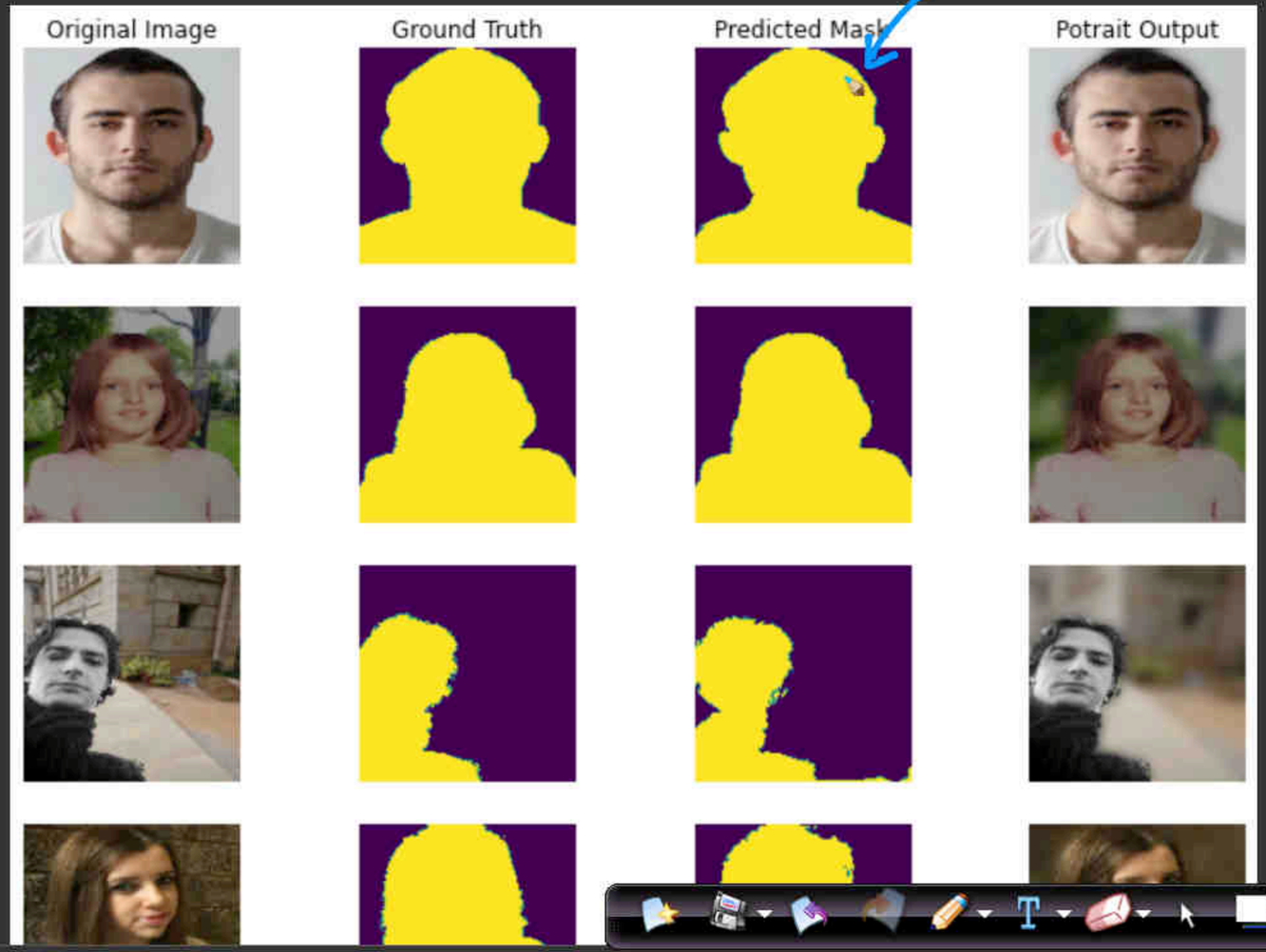
Intuitively

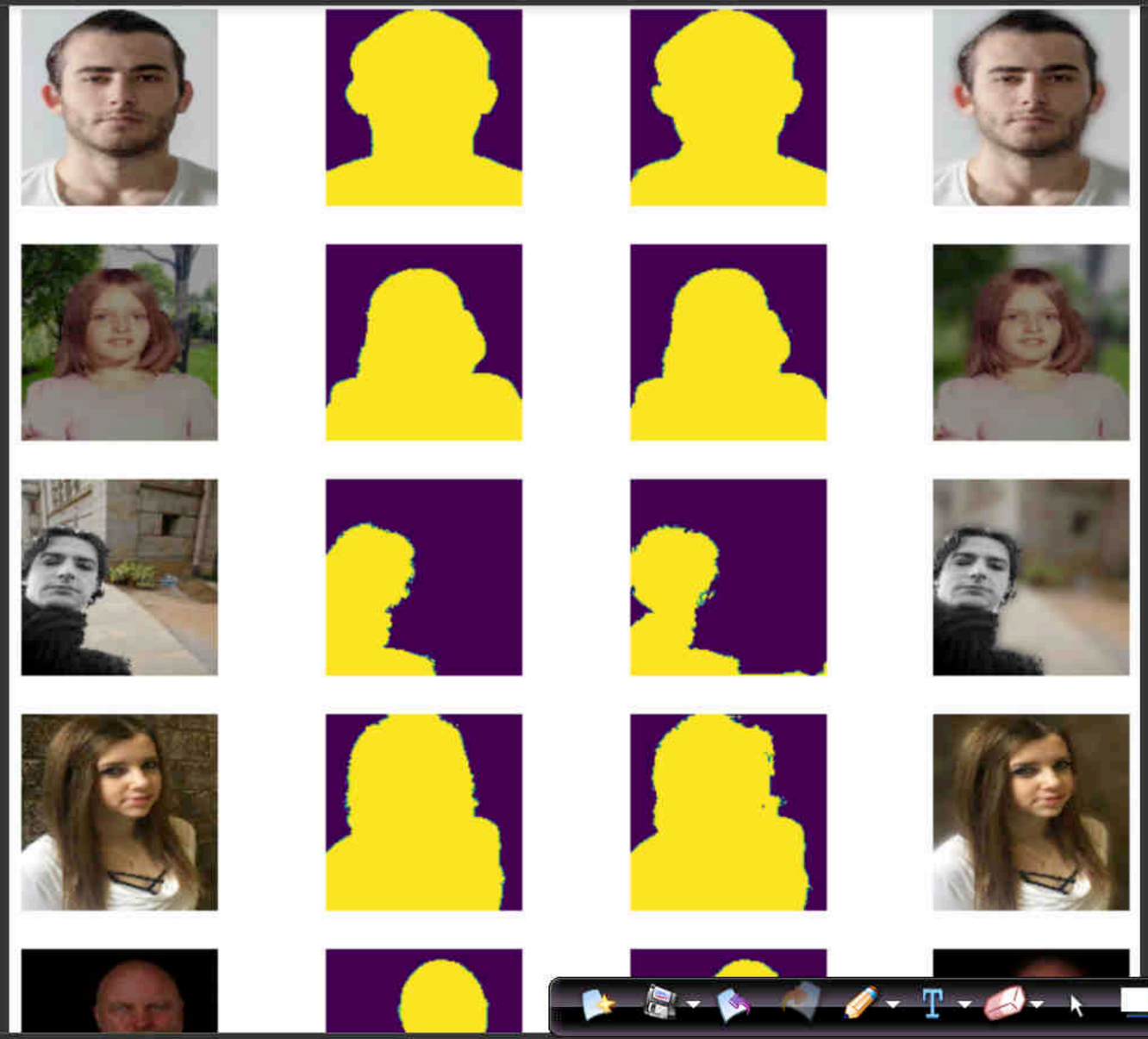
{ conv — Tr.conv
{ maxpooling — Upsampling

```
        ax[idx1][2].axis('off')
        ax[idx1][3].axis('off')
    plt.show()
```



```
        ax[idx1][2].axis('off')
        ax[idx1][3].axis('off')
    plt.show()
```





YOLO (recap)
YOLOV3
RetinaNet (FPN)
Seg: ....
FCN8
U-net

Mask RCNN
DeeplabV3
dilated conv

89 / 89

- Dataset : https://drive.google.com/drive/folders/1NKEogbBaPsKUw_6qEkcB_nCB4P84Alwl?usp=sharing

Outline

- Motivation
- Business Case
- EDA
- image segmentation over classification / detection
- Semantic vs Instance Segmentation
- The idea of Encoder-Decoder network
- Dice loss, Dice coeff
- FCN coding tutorial
- Unet coding tutorial
- Intuition for Mask R-CNN
- Intuition for DeeplabV3

Motivation

- We started with image classification, whi nage or not
- Then in Object Detection we learnt about object localization : where exactly the object is present in the image and we drew a rectangular