Semantic Segmentation on Aerial Drone Images using U-Net

Star Chen, Muyang Xu, Chengyu Zhang

May.2021

Agenda

1. Problem introduction
2. Descriptive Analysis:
   a. Dataset
   b. Preprocessing of the dataset
3. Model Construction:
   a. U-Net model:
      i. Simple structure
      ii. Mobile-Unet
   b. Training Process
4. Evaluation :
   a. Pixel Accuracy
   b. MIoU (mean of intersection over union)
   c. Prediction
5. Reflection and future work

Problem Introduction

- The rise of usage of aerial drone in civil use.
- Drone photography & filming.

- Issues:
- Automation in Drone piloting & Image capturing

- Solution:
- Semantic segmentation on aerial drone images

DJI: Biggest commercial unmanned aerial vehicles manufacturer

Image of an aerial drone

Dataset

- **Semantic Drone Dataset**
- https://www.tugraz.at/index.php?id=22387
- 400 (
  - original images
  - rgb masks
  - .csv file that maps rgb to label)
- 6000 x 4000 px

Institute of Computer Graphics and Vision

SCIENCE
PASSION
TECHNOLOGY

TU Graz

Preprocessing dataset

- Rgb masks → label images
  - Rgb to [0,23]

- 6000 x 4000 px → 768 x 512 px

- Train : val : test = 0.75 : 0.15: 0.1

```python
def processImages(image_files):
    for i, item in enumerate(image_files):
        print("Processing:", i, item)
        img = Image.open(item)
        img = img.resize((768, 512),Image.NEAREST)
        # print(test_img)

        output_filename = item[-7:]
        # img.save(os.path.join(image_path,'compressed_img', output_filename))
        img.save(os.path.join(image_path,'new_size_img', output_filename))
```

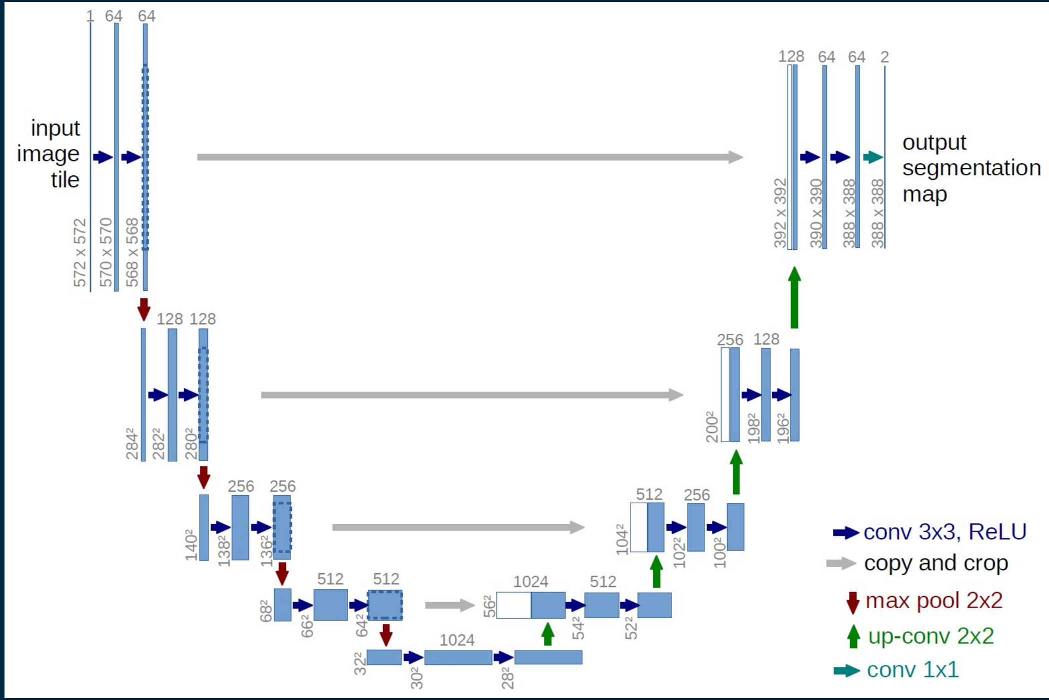
Model Construction

03

- U-Net model
- Training Process

U-Net Model

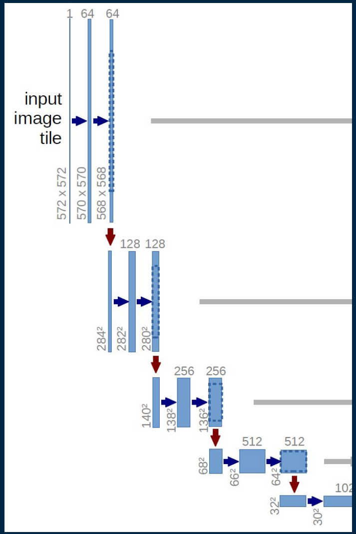
------Simple Structure



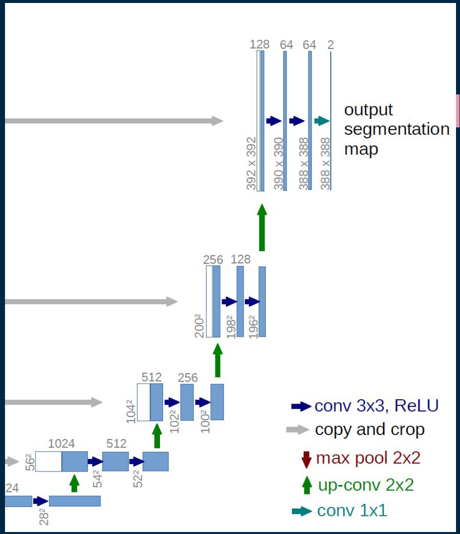
U-Net Structure

Semantic Segmentation:
Doing classification on every pixel

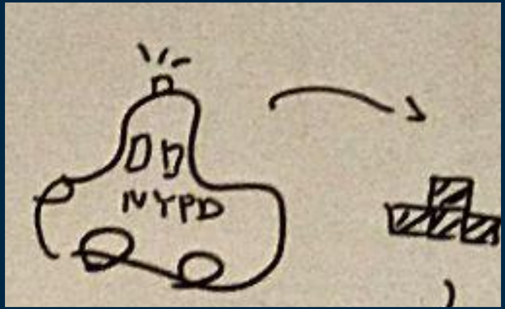
Down Sample
&
Crop+Copy
&
UpSample/Up Conv



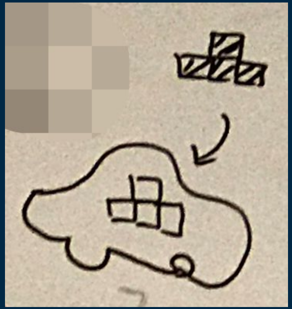
1 64 64
input
image
tile
572 x 572
570 x 570
568 x 568
128 128
284²
282²
280²
256 256
140²
138²
136²
512 512
68²
66²
64²
32²
30²



128 64 64 2
output
segmentation
map
392 x 392
390 x 390
388 x 388
388 x 388
256 128
200²
198²
196²
512 256
104²
102²
100²
1024 512
56²
54²
52²
28²
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1



NYPD

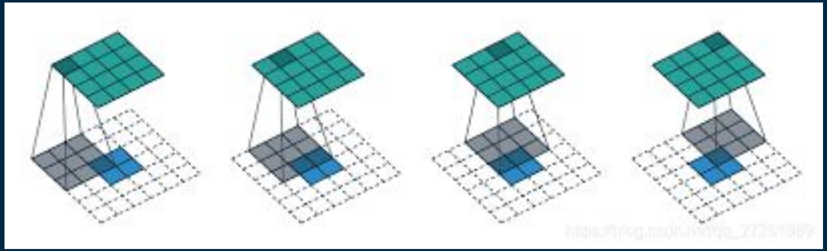


U-Net Model

------Simple Structure

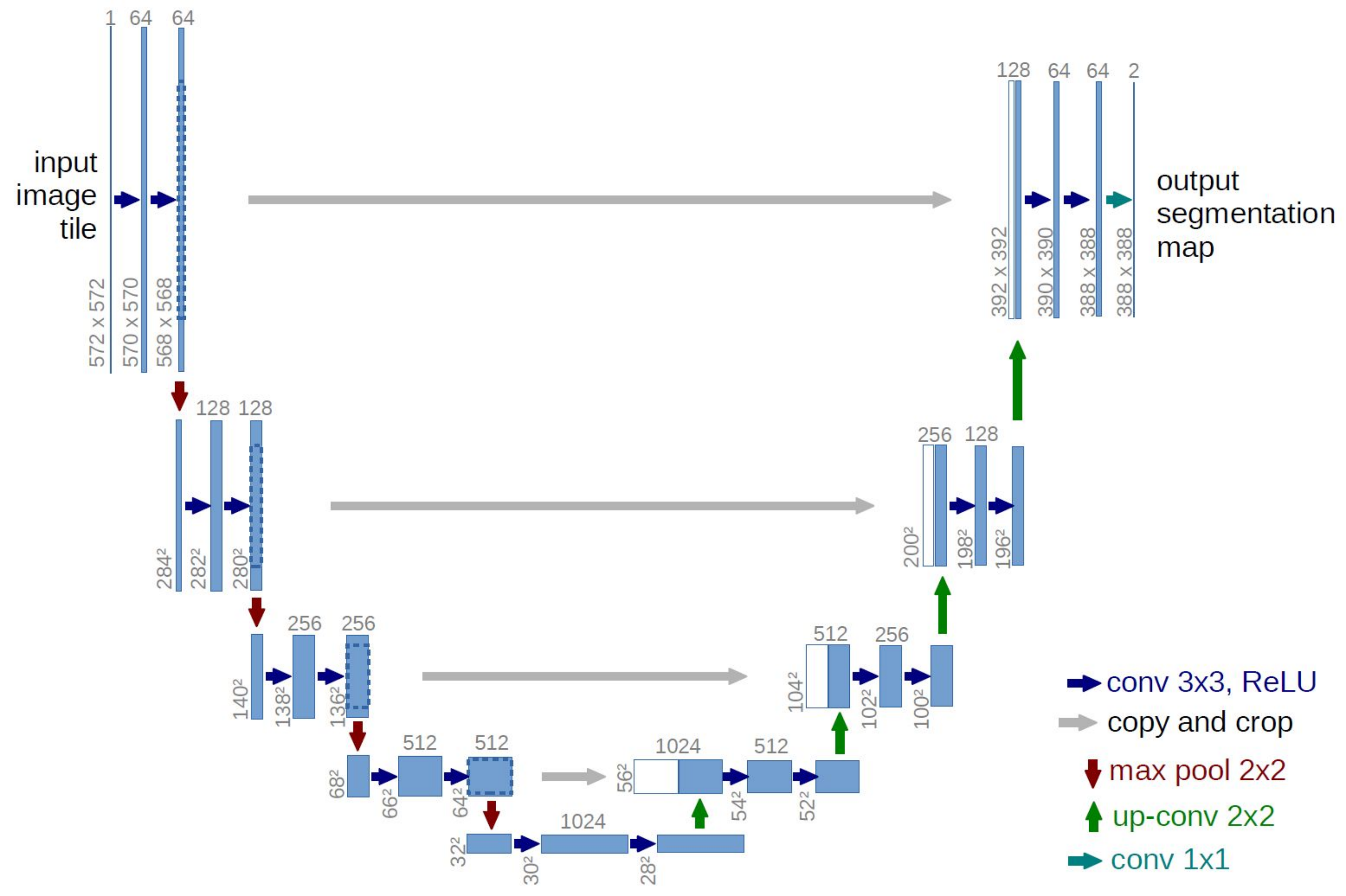
```python
def up_conv(self, in_channels):
    return nn.ConvTranspose2d(
                in_channels,
                out_channels=in_channels // 2,
                kernel_size=2,
                stride=2
                )
```

Applies a 2D transposed convolution operator over an input image composed of several input planes.



Comparing to UpSample:

- Learnable Process
- Reduces Dimension



input image tile
1
64
64
572 x 572
570 x 570
568 x 568
128
128
284²
282²
280²
256
256
140²
138²
136²
512
512
68²
66²
64²
1024
32²
30²
28²
1024
512
56²
54²
52²
512
256
104²
102²
100²
256
128
200²
198²
196²
128
64
64
2
392 x 392
390 x 390
388 x 388
388 x 388
output segmentation map
conv 3x3, ReLU
copy and crop
max pool 2x2
up-conv 2x2
conv 1x1

```python
def double_conv(self, in_channels, out_channels):
    return nn.Sequential(
        nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
        nn.ReLU(inplace=True),
        nn.Conv2d(out_channels, out_channels, kernel_size=3, padding=1),
        nn.ReLU(inplace=True)
    )
```

```python
def Up(self, copy, t):
    th, tw = copy.size()[2:]
    h, w = t.size()[2:]
    diffH = th - h
    diffW = tw - w
    # pad: (padding_left,padding_right, \text{padding\_top}, \text{pa
    t = F.pad(t, [diffW // 2, diffW - diffW // 2,
                diffH // 2, diffH - diffH // 2],  "constant", 0)
    # print(t.size(), copy.size())
    t = torch.cat([copy, t], dim=1)
    return t

# def crop(self, t,target_width,target_height):
#     w, h = t.size()[2:]
#     x1 = int(round((w - target_width) / 2.))
#     y1 = int(round((h - target_height) / 2.))
#     return t[:,:, x1:x1+target_width,y1:y1+target_height]
```

Crop+Copy
to
Pad+Copy

Maintains Shape

No further reshaping on validation set needed

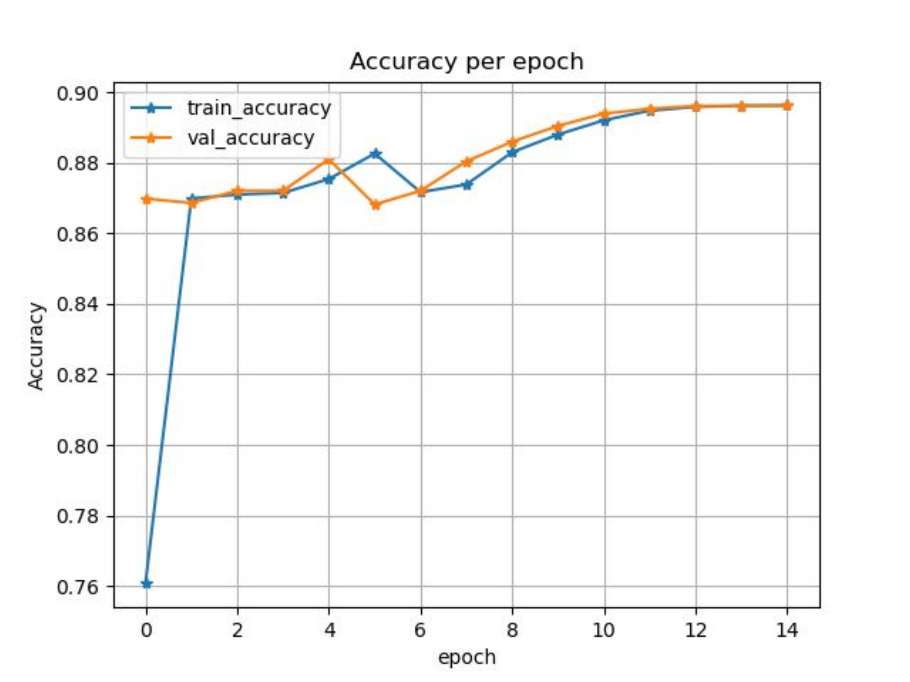
Training Process

First attempts:

Using U-Net: (But on a *wrong* val set)

(Loss Function: criterion = CrossEntropy())

1. Batch_size=1, Epoch=15, Weight_decay=0,Optimizer=Adam
2. Batch_size=2, Epoch=15, Weight_decay=1e-4,Optimizer=Adam
3. Batch_size=2, Epoch=20, Weight_decay=1e-4,Optimizer=AdamW
4. Batch_size=2, Epoch=15, Weight_decay=1e-4,Optimizer=AdamW
5. Batch_size=2, Epoch=15, Weight_decay=1e-4,Optimizer=AdamW, Scheduler=ONECYCLELR

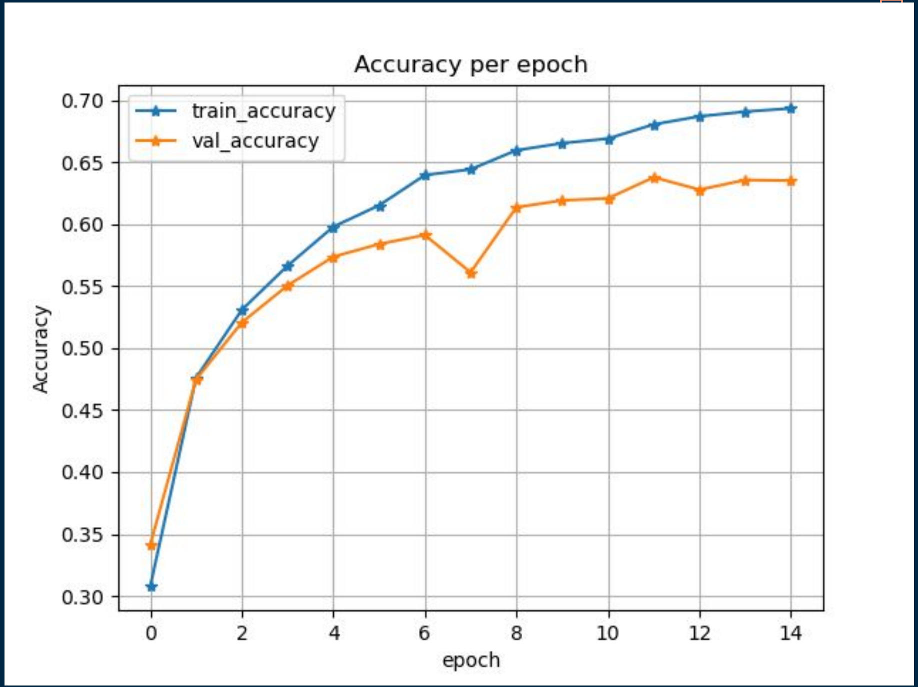


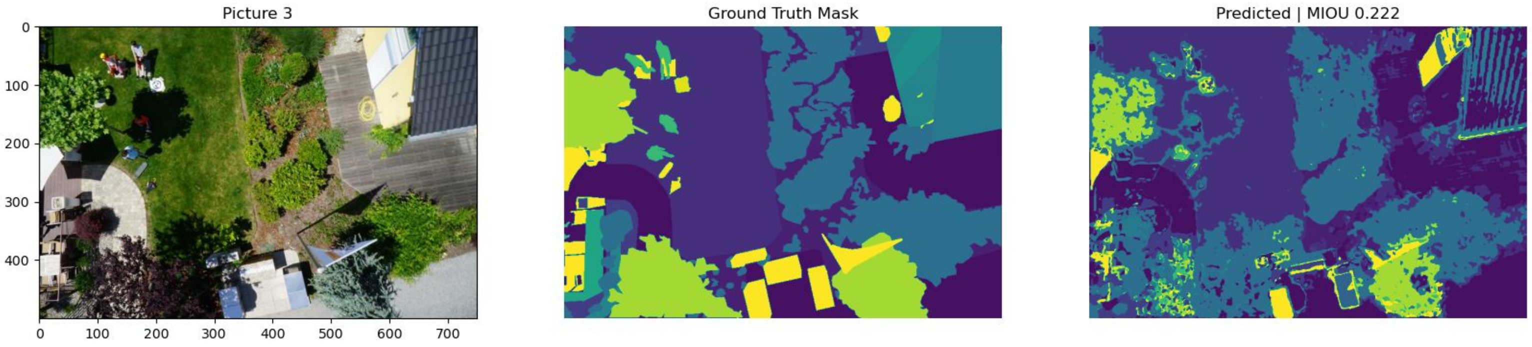
Training Process

Second attempts:

Using U-Net:

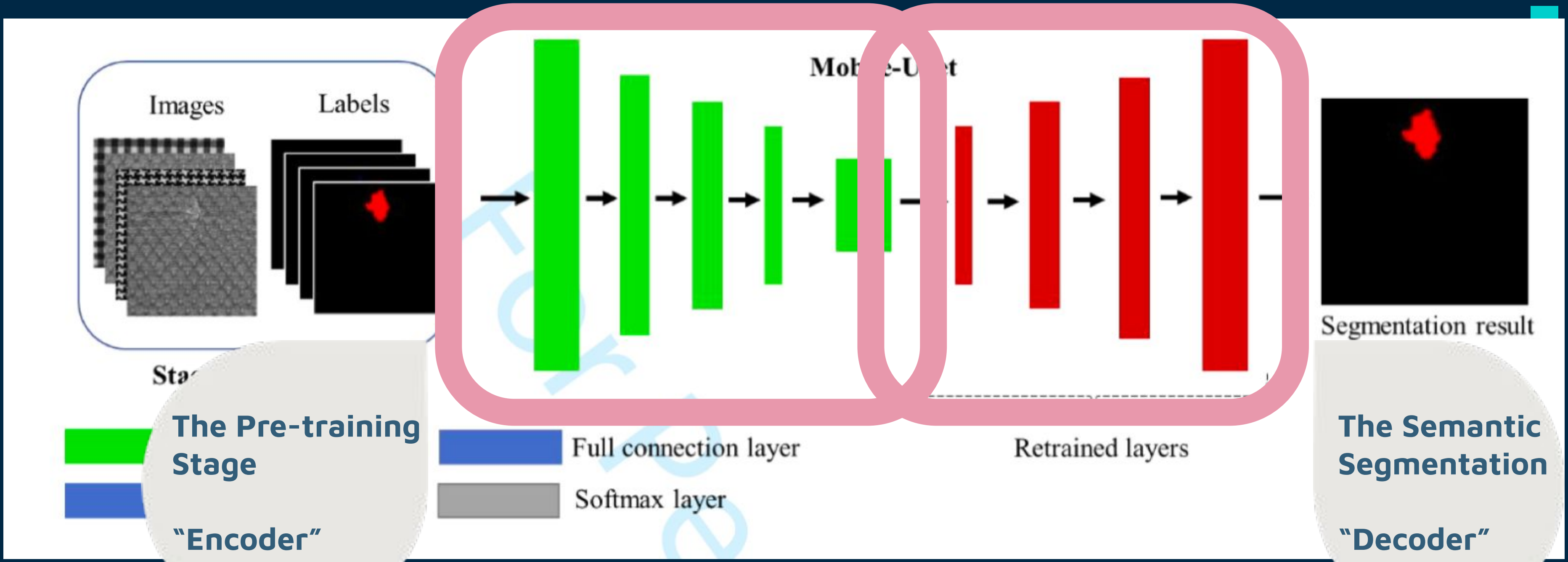
1. Batch_size=2, Epoch=15, Weight_decay=1e-4,Optimizer=AdamW, Scheduler=ONECYCLELR
2. Epoch to 30




Picture 3
Ground Truth Mask
Predicted | MIOU 0.222
0
100
200
300
400
0
100
200
300
400
500
600
700

U-Net Model

------Mobile-Unet



The Pre-training Stage

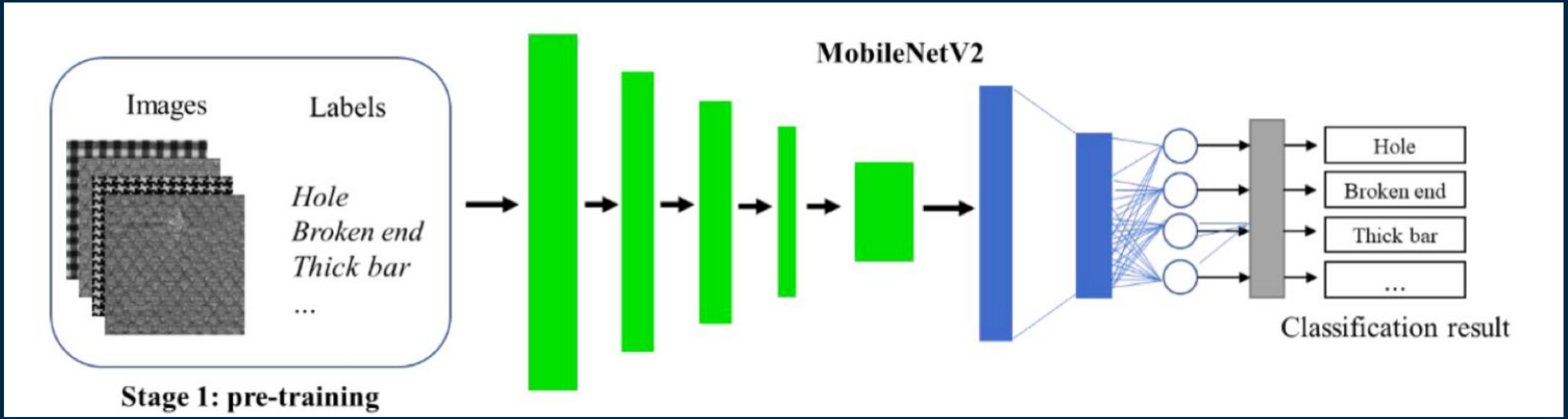
"Encoder"

The Semantic Segmentation

"Decoder"

Two stage

Encoder-MobileNetV2



BASIC CONCEPT

Use depth-wise separable convolutions

- Less parameters need to be adjusted;
- Reduce possible over-fitting

Computationally cheaper

Two Advantages

Depthwise Separable Convolution

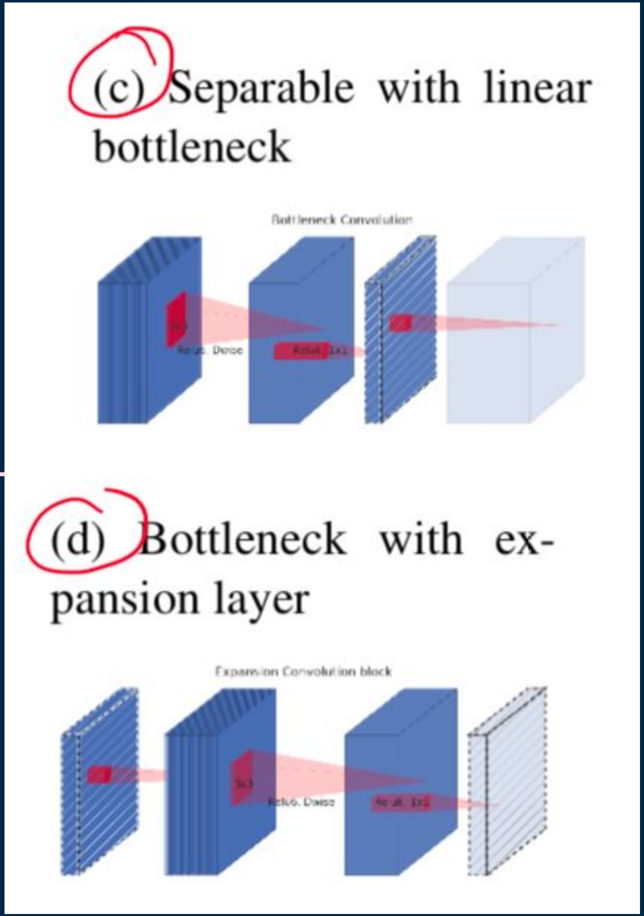
Depthwise
Convolution

First Layer

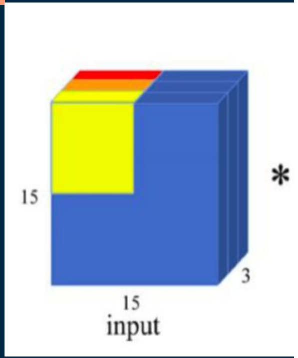
Second Layer

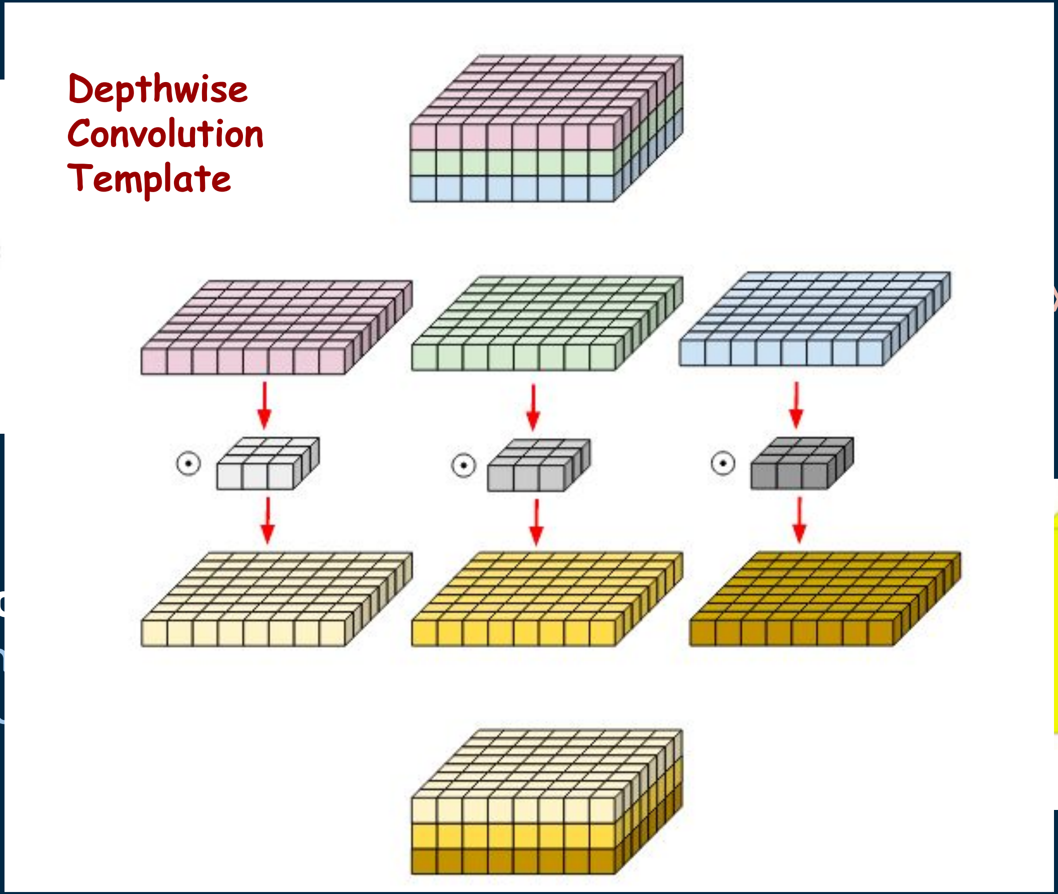
JUPITER

- 1 × 1 Convolution
- Pointwise Convolution

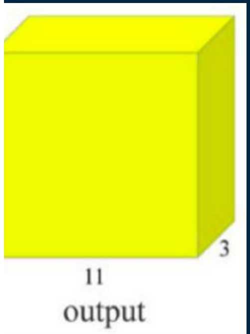


Depthwise Convolution





y at



Norm

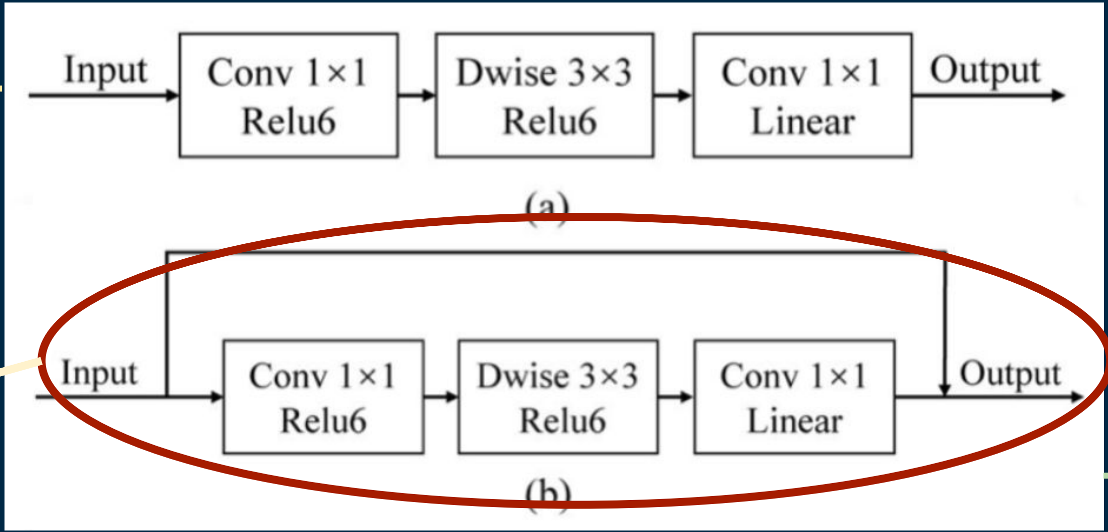
Each

su

Depth-wise Convolution

-------- Convolution Blocks

Stride=2 block



Skip Connection

Stride=1 block

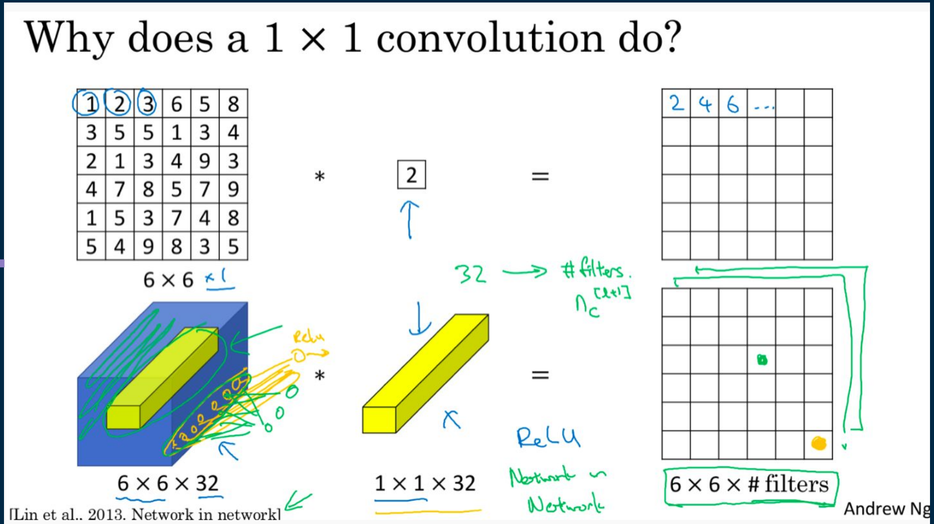
Inverted Residual Blocks

1 × 1 Convolution

Project a "three-dimensional" sample onto **one-dimensional space**

Effect

A Linear Projection

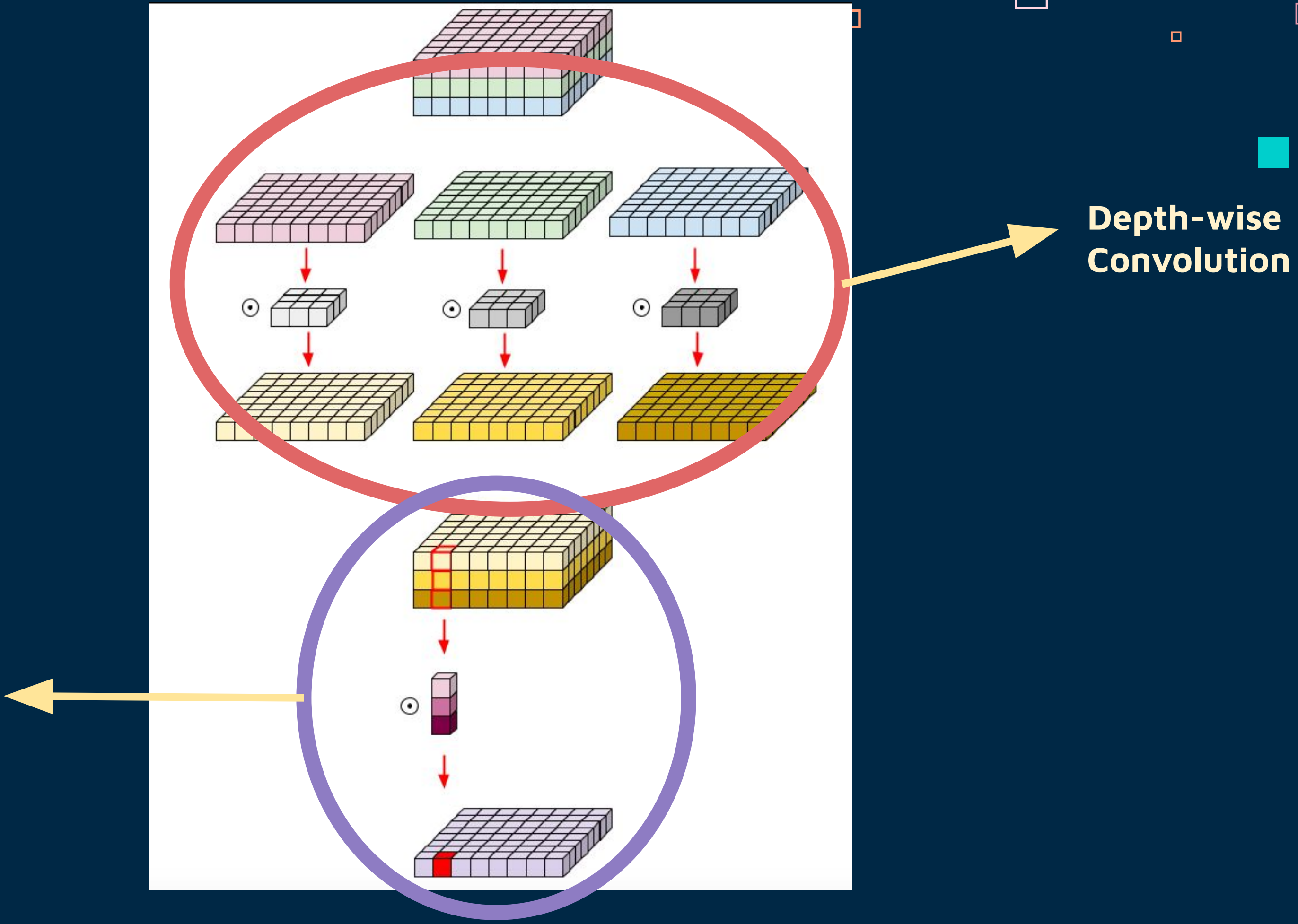


Channel-wise pooling

Objective

Dimensionality Reduction

Depthwise Separable Convolution



Training Process

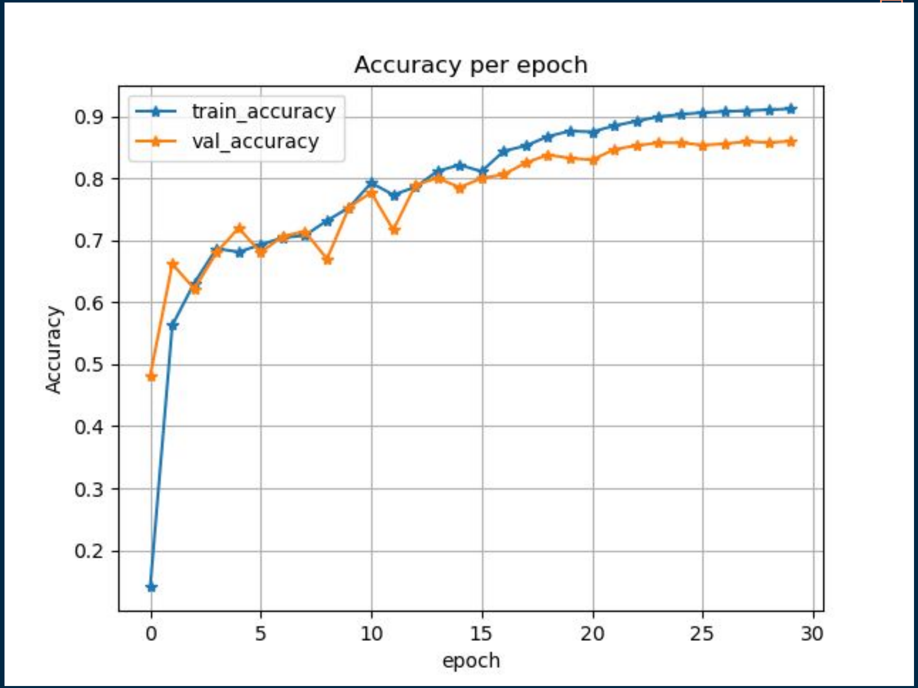
Third attempts:

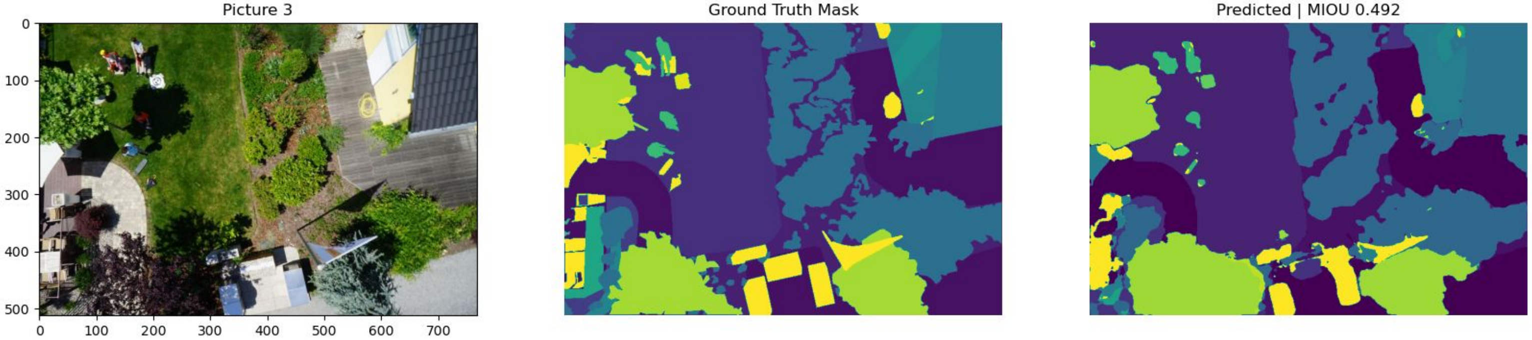
Using Mobile-Unet:

1. Epoch = 15
2. Epoch = 30, with early stop
3. Added Noise
4. Epoch = 30, full

```python
import albumentations as A

# adding transformation after 3rd training
# not really working
t =  A.Compose([
                A.HorizontalFlip(),
                A.VerticalFlip(),
                A.GridDistortion(p=0.2),
                A.RandomBrightnessContrast((0,0.5),(0,0.5)),
                A.GaussNoise()
                ])
```




Picture 3
0
100
200
300
400
500
0
100
200
300
400
500
600
700
Ground Truth Mask
Predicted | MIOU 0.492

U-Net Model

------Mobile-Unet

```
# model = UNet()
model = smp.Unet(
    'mobilenet_v2',
    encoder_weights='imagenet',
    classes=23,
    encoder_depth=5,
    decoder_channels=[256, 128, 64, 32, 16]
    )
```

Better Computational Efficiency:

- Import segmentation_models_pytorch packages
- Utilize the Mobile-Unet model

Comparison

Using U-Net:

```
Epoch:15/15.. Train Loss: 0.992.. Val Loss: 2.316.. Train mIoU:0.181.. Val mIoU: 0.183.. Train Acc:0.694.. Val Acc:0.635.. Time: 2.68m
Total time: 39.83 m
```

Using Mobile-Unet:

```
Epoch:30/30.. Train Loss: 0.306.. Val Loss: 0.961.. Train mIoU:0.519.. Val mIoU: 0.444.. Train Acc:0.912.. Val Acc:0.860.. Time: 0.70m
Total time: 20.75 m
```

Evaluation

- Pixel Accuracy
- MiOU (mean of intersection over union)
- Prediction

04

Pixel Accuracy

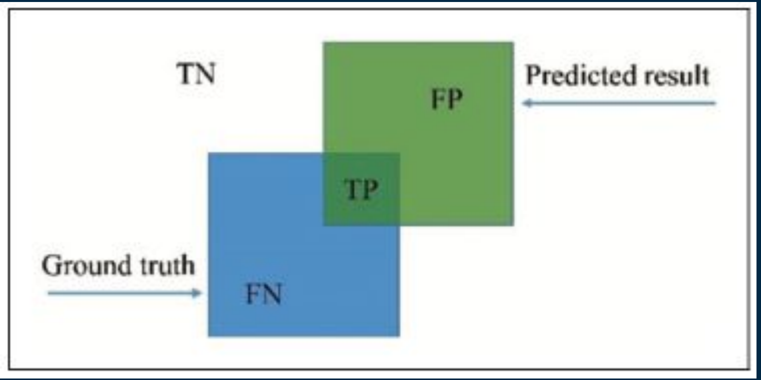
```
def pixel_accuracy(output,label):
    output = torch.argmax(F.softmax(output, dim=1), dim=1)
    accur = torch.eq(output, label).int()
    return (torch.sum(accur).float() / output.nelement())
```

The number of output pixels

The number of True Positive and True Negative pixels

Pixel Accuracy= $$\frac{TP+TN}{TP+FP+TN+FN}$$

MIoU (Mean of Intersection over Union)



MIoU (Mean of Intersection over Union)

```python
def MiOU(pred_mask, mask, smooth=1e-10, n_classes=23):
    with torch.no_grad():
        pred_mask = F.softmax(pred_mask, dim=1)
        pred_mask = torch.argmax(pred_mask, dim=1)
        pred_mask = pred_mask.contiguous().view(-1)
        mask = mask.contiguous().view(-1)

        iou_per_class = []
        for clas in range(0, n_classes): #loop per pixel class
            true_class = pred_mask == clas
            true_label = mask == clas

            if true_label.long().sum().item() == 0: #no exist label in this loop
                iou_per_class.append(np.nan)
            else:
                intersect = torch.logical_and(true_class, true_label).sum().float().item()
                union = torch.logical_or(true_class, true_label).sum().float().item()

                iou = (intersect + smooth) / (union +smooth)
                iou_per_class.append(iou)
        return np.nanmean(iou_per_class)
```

"TRANSFORM" THE OUTPUT INTO ONE CHANNEL ("ONE-DIMENSIONAL PIXEL MAP")

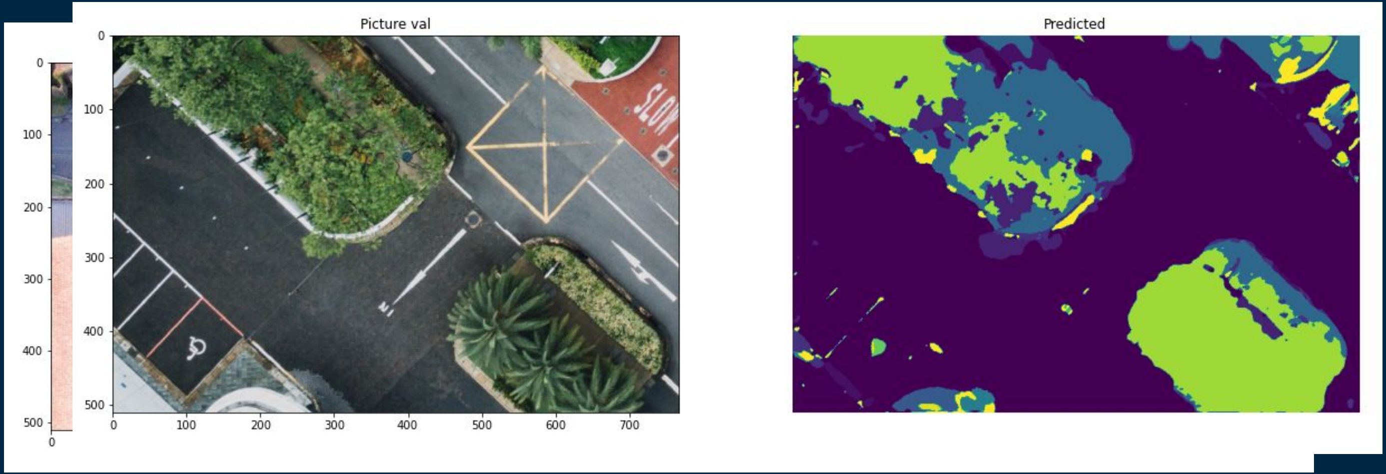
#TRUE POSITIVE

#TRUE POSITIVE+FALSE POSITIVE+FALSE NEGATIVE

Prediction

- We create our own prediction dataset with aerial drone images
- Problem: Hard to create rgb mask or grayscale mask for images
- Solution: Semantic Segmentation Editor
  - Load rgb classes
  - Auto-segmentation works badly


Picture val
Ground Truth Mask
Predicted | MIOU 0.191
0 100 200 300 400 500
0 100 200 300 400 500 600 700


Picture val
Predicted
0 100 200 300 400 500
0 100 200 300 400 500 600 700

Reflection & Future Work

- Extend to video (model)
- Improve prediction speed (model)
- Include more context of the images (model)
- Mobilenet_v3? (model)

- The improvement of computation:
  - Eg: MiOU----OneHot matrix (output*label=sum of intersection)

- More angles (dataset)
- Messier neighborhood (dataset)
- More scenarios (dataset)

- => Since our purpose of improving automation in Drone piloting & Image capturing requires real time accurate response.

THANK YOU!