What is Semantic Segmentation?

Semantic segmentation refers to the process of associating each pixel in an image with a specific class label. These labels can include objects such as a person, car, flower, or a piece of furniture, among others.

We can think of semantic segmentation as a form of image classification at the pixel level. For example, in an image containing multiple cars, semantic segmentation will label all objects as 'car.' However, there's a distinct class of models called instance segmentation, which is capable of labeling individual instances of an object within an image. This type of segmentation is particularly valuable in applications that require object counting, such as monitoring foot traffic in a mall.

Import Libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader
from torchvision import transforms as T
import torchvision
import torch.nn.functional as F
from torch.autograd import Variable

from PIL import Image
import cv2
import albumentations as A

import time
import os
from tqdm.notebook import tqdm

!pip install -q segmentation-models-pytorch
!pip install -q torchsummary
```

```
ERROR: pip's dependency resolver does not currently take into account all the package
s that are installed. This behaviour is the source of the following dependency confli
cts.
jupyterlab-git 0.11.0 requires nbdime<2.0.0,>=1.1.0, but you have nbdime 2.1.0 which
is incompatible.
bokeh 2.2.3 requires tornado>=5.1, but you have tornado 5.0.2 which is incompatible.
autogluon-core 0.1.0b20210210 requires numpy==1.19, but you have numpy 1.19.5 which i
s incompatible.
WARNING: You are using pip version 21.0.1; however, version 23.2.1 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgr
ade pip' command.
WARNING: You are using pip version 21.0.1; however, version 23.2.1 is available.
You should consider upgrading via the '/opt/conda/bin/python3.7 -m pip install --upgr
ade pip' command.
```

Preprocessing

In [2]:
```
IMAGE_PATH = '../input/semantic-drone-dataset/dataset/semantic_drone_dataset/original_
MASK_PATH = '../input/semantic-drone-dataset/dataset/semantic_drone_dataset/label_imag
```

In [3]:
```
n_classes = 23

def create_df():
    name = []
    for dirname, _, filenames in os.walk(IMAGE_PATH):
        for filename in filenames:
            name.append(filename.split('.')[0])

    return pd.DataFrame({'id': name}, index = np.arange(0, len(name)))

df = create_df()
print('Total Images: ', len(df))
```

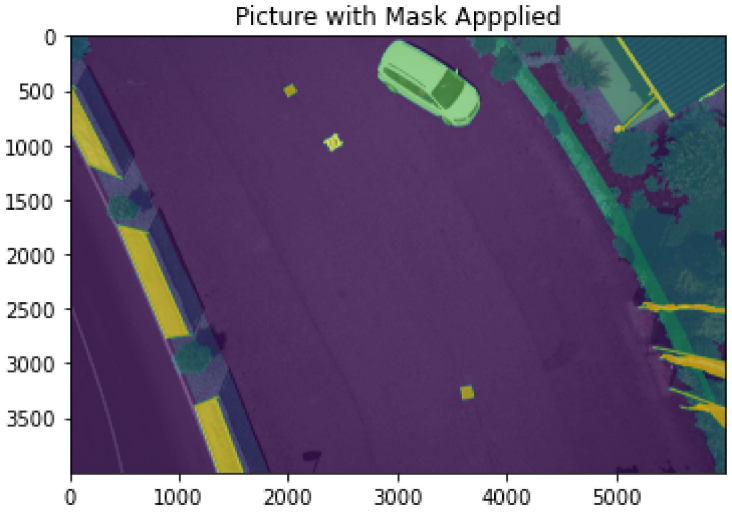
```
Total Images:  400
```

In [4]:
```
#split data
X_trainval, X_test = train_test_split(df['id'].values, test_size=0.1, random_state=19)
X_train, X_val = train_test_split(X_trainval, test_size=0.15, random_state=19)

print('Train Size   : ', len(X_train))
print('Val Size     : ', len(X_val))
print('Test Size    : ', len(X_test))
```

```
Train Size   :  306
Val Size     :  54
Test Size    :  40
```

In [5]:
```
img = Image.open(IMAGE_PATH + df['id'][100] + '.jpg')
mask = Image.open(MASK_PATH + df['id'][100] + '.png')
print('Image Size', np.asarray(img).shape)
print('Mask Size', np.asarray(mask).shape)


plt.imshow(img)
plt.imshow(mask, alpha=0.6)
plt.title('Picture with Mask Appplied')
plt.show()
```

```
Image Size (4000, 6000, 3)
Mask Size (4000, 6000)
```



Dataset

In [6]:

```python
class DroneDataset(Dataset):

    def __init__(self, img_path, mask_path, X, mean, std, transform=None, patch=False):
        self.img_path = img_path
        self.mask_path = mask_path
        self.X = X
        self.transform = transform
        self.patches = patch
        self.mean = mean
        self.std = std

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        img = cv2.imread(self.img_path + self.X[idx] + '.jpg')
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        mask = cv2.imread(self.mask_path + self.X[idx] + '.png', cv2.IMREAD_GRAYSCALE)

        if self.transform is not None:
            aug = self.transform(image=img, mask=mask)
            img = Image.fromarray(aug['image'])
            mask = aug['mask']

        if self.transform is None:
            img = Image.fromarray(img)

        t = T.Compose([T.ToTensor(), T.Normalize(self.mean, self.std)])
        img = t(img)
        mask = torch.from_numpy(mask).long()

        if self.patches:
            img, mask = self.tiles(img, mask)

        return img, mask
```

```
    def tiles(self, img, mask):

        img_patches = img.unfold(1, 512, 512).unfold(2, 768, 768)
        img_patches  = img_patches.contiguous().view(3,-1, 512, 768)
        img_patches = img_patches.permute(1,0,2,3)

        mask_patches = mask.unfold(0, 512, 512).unfold(1, 768, 768)
        mask_patches = mask_patches.contiguous().view(-1, 512, 768)

        return img_patches, mask_patches
```

In [7]:
```
mean=[0.485, 0.456, 0.406]
std=[0.229, 0.224, 0.225]

t_train = A.Compose([A.Resize(704, 1056, interpolation=cv2.INTER_NEAREST), A.Horizonta
                     A.GridDistortion(p=0.2), A.RandomBrightnessContrast((0,0.5),(0,0.
                     A.GaussNoise()])

t_val = A.Compose([A.Resize(704, 1056, interpolation=cv2.INTER_NEAREST), A.HorizontalF
                   A.GridDistortion(p=0.2)])

#datasets
train_set = DroneDataset(IMAGE_PATH, MASK_PATH, X_train, mean, std, t_train, patch=Fal
val_set = DroneDataset(IMAGE_PATH, MASK_PATH, X_val, mean, std, t_val, patch=False)

#dataloader
batch_size= 3

train_loader = DataLoader(train_set, batch_size=batch_size, shuffle=True)
val_loader = DataLoader(val_set, batch_size=batch_size, shuffle=True)
```

Model

In [8]:
```
model = smp.Unet('mobilenet_v2', encoder_weights='imagenet', classes=23, activation=No
```

```
Downloading: "https://download.pytorch.org/models/mobilenet_v2-b0353104.pth" to /roo
t/.cache/torch/hub/checkpoints/mobilenet_v2-b0353104.pth
  0%|          | 0.00/13.6M [00:00<?, ?B/s]
```

In [9]:
```
model
```

Out[9]:

```
Unet(
  (encoder): MobileNetV2Encoder(
    (features): Sequential(
      (0): ConvBNReLU(
        (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=Fa
lse)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
        (2): ReLU6(inplace=True)
      )
      (1): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), gr
oups=32, bias=False)
            (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): Conv2d(32, 16, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (2): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(16, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(96, 96, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), gr
oups=96, bias=False)
            (1): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_
stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(96, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (3): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=144, bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(144, 24, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
          (3): BatchNorm2d(24, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (4): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(24, 144, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(144, 144, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=144, bias=False)
            (1): BatchNorm2d(144, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(144, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (5): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=192, bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (6): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(192, 192, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=192, bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(192, 32, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
          (3): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (7): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(32, 192, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(192, 192, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=192, bias=False)
            (1): BatchNorm2d(192, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(192, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (8): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=384, bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (9): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=384, bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (10): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=384, bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(384, 64, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (11): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(64, 384, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(384, 384, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=384, bias=False)
            (1): BatchNorm2d(384, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(384, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (12): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=576, bias=False)
            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
          (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (13): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(576, 576, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=576, bias=False)
            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(576, 96, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(96, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        )
      )
      (14): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(96, 576, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(576, 576, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1),
groups=576, bias=False)
            (1): BatchNorm2d(576, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(576, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        )
      )
      (15): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=960, bias=False)
            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
```

```
          (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        )
      )
      (16): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=960, bias=False)
            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(960, 160, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(160, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        )
      )
      (17): InvertedResidual(
        (conv): Sequential(
          (0): ConvBNReLU(
            (0): Conv2d(160, 960, kernel_size=(1, 1), stride=(1, 1), bias=False)
            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (1): ConvBNReLU(
            (0): Conv2d(960, 960, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1),
groups=960, bias=False)
            (1): BatchNorm2d(960, eps=1e-05, momentum=0.1, affine=True, track_running
_stats=True)
            (2): ReLU6(inplace=True)
          )
          (2): Conv2d(960, 320, kernel_size=(1, 1), stride=(1, 1), bias=False)
          (3): BatchNorm2d(320, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
        )
      )
      (18): ConvBNReLU(
        (0): Conv2d(320, 1280, kernel_size=(1, 1), stride=(1, 1), bias=False)
        (1): BatchNorm2d(1280, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
        (2): ReLU6(inplace=True)
      )
    )
  )
  (decoder): UnetDecoder(
    (center): Identity()
    (blocks): ModuleList(
      (0): DecoderBlock(
        (conv1): Conv2dReLU(
          (0): Conv2d(1376, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), b
ias=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
```

```
tats=True)
          (2): ReLU(inplace=True)
        )
        (attention1): Attention(
          (attention): Identity()
        )
        (conv2): Conv2dReLU(
          (0): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
          (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
          (2): ReLU(inplace=True)
        )
        (attention2): Attention(
          (attention): Identity()
        )
      )
      (1): DecoderBlock(
        (conv1): Conv2dReLU(
          (0): Conv2d(288, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
          (2): ReLU(inplace=True)
        )
        (attention1): Attention(
          (attention): Identity()
        )
        (conv2): Conv2dReLU(
          (0): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
          (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_s
tats=True)
          (2): ReLU(inplace=True)
        )
        (attention2): Attention(
          (attention): Identity()
        )
      )
      (2): DecoderBlock(
        (conv1): Conv2dReLU(
          (0): Conv2d(152, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bia
s=False)
          (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
          (2): ReLU(inplace=True)
        )
        (attention1): Attention(
          (attention): Identity()
        )
        (conv2): Conv2dReLU(
          (0): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
          (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
          (2): ReLU(inplace=True)
        )
        (attention2): Attention(
          (attention): Identity()
        )
```

```
      )
      (3): DecoderBlock(
        (conv1): Conv2dReLU(
          (0): Conv2d(80, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
          (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
          (2): ReLU(inplace=True)
        )
        (attention1): Attention(
          (attention): Identity()
        )
        (conv2): Conv2dReLU(
          (0): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
          (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
          (2): ReLU(inplace=True)
        )
        (attention2): Attention(
          (attention): Identity()
        )
      )
      (4): DecoderBlock(
        (conv1): Conv2dReLU(
          (0): Conv2d(32, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
          (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
          (2): ReLU(inplace=True)
        )
        (attention1): Attention(
          (attention): Identity()
        )
        (conv2): Conv2dReLU(
          (0): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
          (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_st
ats=True)
          (2): ReLU(inplace=True)
        )
        (attention2): Attention(
          (attention): Identity()
        )
      )
    )
  )
  (segmentation_head): SegmentationHead(
    (0): Conv2d(16, 23, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): Identity()
    (2): Activation(
      (activation): Identity()
    )
  )
)
```

Training

In [10]:
```
def pixel_accuracy(output, mask):
    with torch.no_grad():
        output = torch.argmax(F.softmax(output, dim=1), dim=1)
        correct = torch.eq(output, mask).int()
        accuracy = float(correct.sum()) / float(correct.numel())
    return accuracy
```

In [11]:
```
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().it
                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)
```

In [12]:
```
def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

def fit(epochs, model, train_loader, val_loader, criterion, optimizer, scheduler, patc
    torch.cuda.empty_cache()
    train_losses = []
    test_losses = []
    val_iou = []; val_acc = []
    train_iou = []; train_acc = []
    lrs = []
    min_loss = np.inf
    decrease = 1 ; not_improve=0

    model.to(device)
    fit_time = time.time()
    for e in range(epochs):
        since = time.time()
        running_loss = 0
        iou_score = 0
        accuracy = 0
        #training loop
        model.train()
        for i, data in enumerate(tqdm(train_loader)):
            #training phase
            image_tiles, mask_tiles = data
            if patch:
                bs, n_tiles, c, h, w = image_tiles.size()
```

```
                image_tiles = image_tiles.view(-1,c, h, w)
                mask_tiles = mask_tiles.view(-1, h, w)

            image = image_tiles.to(device); mask = mask_tiles.to(device);
            #forward
            output = model(image)
            loss = criterion(output, mask)
            #evaluation metrics
            iou_score += mIoU(output, mask)
            accuracy += pixel_accuracy(output, mask)
            #backward
            loss.backward()
            optimizer.step() #update weight
            optimizer.zero_grad() #reset gradient

            #step the learning rate
            lrs.append(get_lr(optimizer))
            scheduler.step()

            running_loss += loss.item()

        else:
            model.eval()
            test_loss = 0
            test_accuracy = 0
            val_iou_score = 0
            #validation loop
            with torch.no_grad():
                for i, data in enumerate(tqdm(val_loader)):
                    #reshape to 9 patches from single image, delete batch size
                    image_tiles, mask_tiles = data

                    if patch:
                        bs, n_tiles, c, h, w = image_tiles.size()

                        image_tiles = image_tiles.view(-1,c, h, w)
                        mask_tiles = mask_tiles.view(-1, h, w)

                    image = image_tiles.to(device); mask = mask_tiles.to(device);
                    output = model(image)
                    #evaluation metrics
                    val_iou_score +=  mIoU(output, mask)
                    test_accuracy += pixel_accuracy(output, mask)
                    #loss
                    loss = criterion(output, mask)
                    test_loss += loss.item()

            #calculatio mean for each batch
            train_losses.append(running_loss/len(train_loader))
            test_losses.append(test_loss/len(val_loader))


            if min_loss > (test_loss/len(val_loader)):
                print('Loss Decreasing.. {:.3f} >> {:.3f} '.format(min_loss, (test_los
                min_loss = (test_loss/len(val_loader))
                decrease += 1
                if decrease % 5 == 0:
                    print('saving model...')
                    torch.save(model, 'Unet-Mobilenet_v2_mIoU-{:.3f}.pt'.format(val_ic
```

```
            if (test_loss/len(val_loader)) > min_loss:
                not_improve += 1
                min_loss = (test_loss/len(val_loader))
                print(f'Loss Not Decrease for {not_improve} time')
                if not_improve == 7:
                    print('Loss not decrease for 7 times, Stop Training')
                    break

            #iou
            val_iou.append(val_iou_score/len(val_loader))
            train_iou.append(iou_score/len(train_loader))
            train_acc.append(accuracy/len(train_loader))
            val_acc.append(test_accuracy/ len(val_loader))
            print("Epoch:{}/{}..".format(e+1, epochs),
                  "Train Loss: {:.3f}..".format(running_loss/len(train_loader)),
                  "Val Loss: {:.3f}..".format(test_loss/len(val_loader)),
                  "Train mIoU:{:.3f}..".format(iou_score/len(train_loader)),
                  "Val mIoU: {:.3f}..".format(val_iou_score/len(val_loader)),
                  "Train Acc:{:.3f}..".format(accuracy/len(train_loader)),
                  "Val Acc:{:.3f}..".format(test_accuracy/len(val_loader)),
                  "Time: {:.2f}m".format((time.time()-since)/60))

    history = {'train_loss' : train_losses, 'val_loss': test_losses,
               'train_miou' :train_iou, 'val_miou':val_iou,
               'train_acc' :train_acc, 'val_acc':val_acc,
               'lrs': lrs}
    print('Total time: {:.2f} m' .format((time.time()- fit_time)/60))
    return history
```

In [13]:

```
max_lr = 1e-3
epoch = 15
weight_decay = 1e-4

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=max_lr, weight_decay=weight_decay
sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epoch,
                                            steps_per_epoch=len(train_loader))

history = fit(epoch, model, train_loader, val_loader, criterion, optimizer, sched)
```

```
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. inf >> 2.201
Epoch:1/15.. Train Loss: 2.891.. Val Loss: 2.201.. Train mIoU:0.051.. Val mIoU: 0.11
9.. Train Acc:0.189.. Val Acc:0.500.. Time: 4.62m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 2.201 >> 1.497
Epoch:2/15.. Train Loss: 1.974.. Val Loss: 1.497.. Train mIoU:0.118.. Val mIoU: 0.15
2.. Train Acc:0.556.. Val Acc:0.645.. Time: 4.35m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 1.497 >> 1.198
Epoch:3/15.. Train Loss: 1.473.. Val Loss: 1.198.. Train mIoU:0.118.. Val mIoU: 0.14
2.. Train Acc:0.578.. Val Acc:0.642.. Time: 4.36m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
```

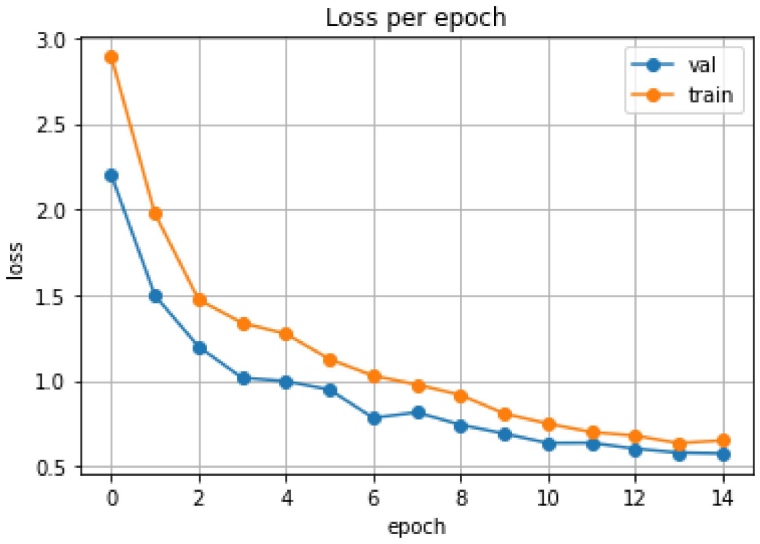
```
Loss Decreasing.. 1.198 >> 1.016
saving model...
Epoch:4/15.. Train Loss: 1.337.. Val Loss: 1.016.. Train mIoU:0.144.. Val mIoU: 0.17
8.. Train Acc:0.607.. Val Acc:0.700.. Time: 4.38m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 1.016 >> 0.997
Epoch:5/15.. Train Loss: 1.273.. Val Loss: 0.997.. Train mIoU:0.155.. Val mIoU: 0.19
0.. Train Acc:0.617.. Val Acc:0.716.. Time: 4.38m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 0.997 >> 0.947
Epoch:6/15.. Train Loss: 1.125.. Val Loss: 0.947.. Train mIoU:0.179.. Val mIoU: 0.20
8.. Train Acc:0.666.. Val Acc:0.710.. Time: 4.38m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 0.947 >> 0.782
Epoch:7/15.. Train Loss: 1.029.. Val Loss: 0.782.. Train mIoU:0.211.. Val mIoU: 0.23
8.. Train Acc:0.701.. Val Acc:0.768.. Time: 4.42m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Not Decrease for 1 time
Epoch:8/15.. Train Loss: 0.977.. Val Loss: 0.814.. Train mIoU:0.219.. Val mIoU: 0.21
7.. Train Acc:0.711.. Val Acc:0.764.. Time: 4.36m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 0.814 >> 0.742
Epoch:9/15.. Train Loss: 0.916.. Val Loss: 0.742.. Train mIoU:0.233.. Val mIoU: 0.25
7.. Train Acc:0.732.. Val Acc:0.774.. Time: 4.42m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 0.742 >> 0.691
saving model...
Epoch:10/15.. Train Loss: 0.808.. Val Loss: 0.691.. Train mIoU:0.259.. Val mIoU: 0.27
1.. Train Acc:0.763.. Val Acc:0.788.. Time: 4.36m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 0.691 >> 0.635
Epoch:11/15.. Train Loss: 0.749.. Val Loss: 0.635.. Train mIoU:0.274.. Val mIoU: 0.29
2.. Train Acc:0.780.. Val Acc:0.810.. Time: 4.37m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Not Decrease for 2 time
Epoch:12/15.. Train Loss: 0.699.. Val Loss: 0.636.. Train mIoU:0.295.. Val mIoU: 0.30
0.. Train Acc:0.796.. Val Acc:0.806.. Time: 4.39m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 0.636 >> 0.602
Epoch:13/15.. Train Loss: 0.678.. Val Loss: 0.602.. Train mIoU:0.298.. Val mIoU: 0.31
7.. Train Acc:0.801.. Val Acc:0.821.. Time: 4.41m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
Loss Decreasing.. 0.602 >> 0.579
Epoch:14/15.. Train Loss: 0.634.. Val Loss: 0.579.. Train mIoU:0.318.. Val mIoU: 0.32
1.. Train Acc:0.814.. Val Acc:0.825.. Time: 4.43m
  0%|          | 0/102 [00:00<?, ?it/s]
  0%|          | 0/18 [00:00<?, ?it/s]
```

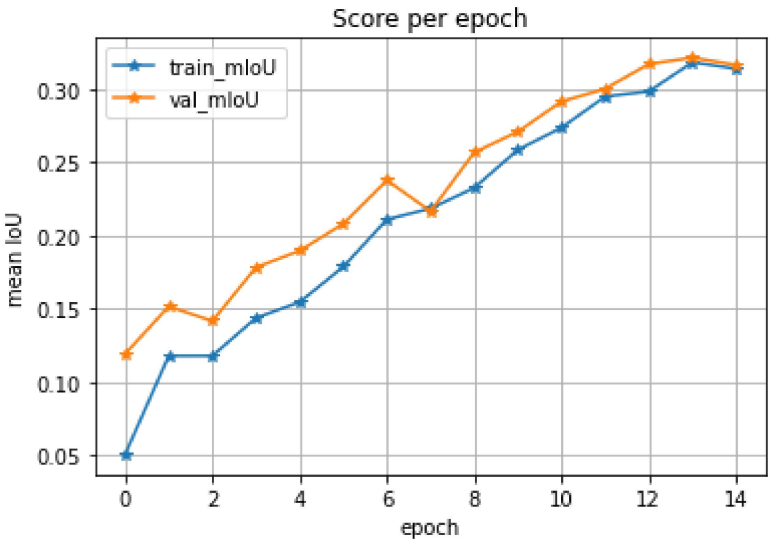
```
Loss Decreasing.. 0.579 >> 0.574
Epoch:15/15.. Train Loss: 0.650.. Val Loss: 0.574.. Train mIoU:0.314.. Val mIoU: 0.31
7.. Train Acc:0.812.. Val Acc:0.825.. Time: 4.39m
Total time: 66.03 m
```

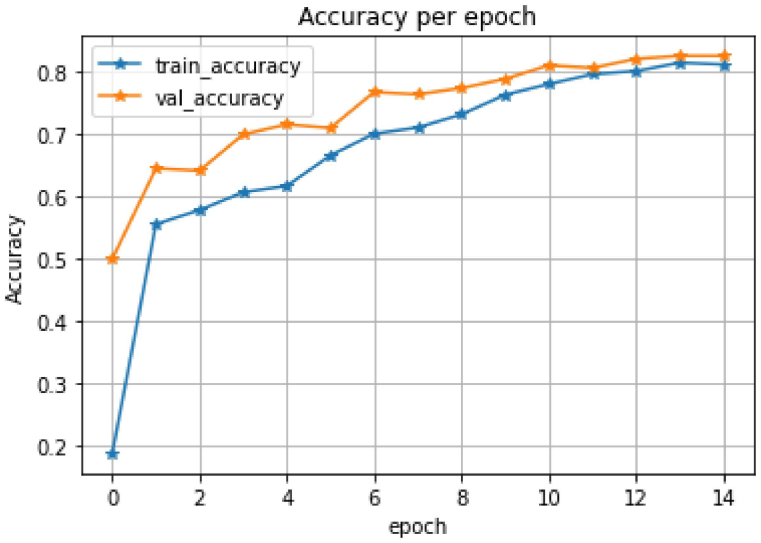
In [14]:
```
torch.save(model, 'Unet-Mobilenet.pt')
```

In [15]:
```
def plot_loss(history):
    plt.plot(history['val_loss'], label='val', marker='o')
    plt.plot( history['train_loss'], label='train', marker='o')
    plt.title('Loss per epoch'); plt.ylabel('loss');
    plt.xlabel('epoch')
    plt.legend(), plt.grid()
    plt.show()

def plot_score(history):
    plt.plot(history['train_miou'], label='train_mIoU', marker='*')
    plt.plot(history['val_miou'], label='val_mIoU',  marker='*')
    plt.title('Score per epoch'); plt.ylabel('mean IoU')
    plt.xlabel('epoch')
    plt.legend(), plt.grid()
    plt.show()

def plot_acc(history):
    plt.plot(history['train_acc'], label='train_accuracy', marker='*')
    plt.plot(history['val_acc'], label='val_accuracy',  marker='*')
    plt.title('Accuracy per epoch'); plt.ylabel('Accuracy')
    plt.xlabel('epoch')
    plt.legend(), plt.grid()
    plt.show()
```

In [16]:
```
plot_loss(history)
plot_score(history)
plot_acc(history)
```







Evaluation

In [17]:

```python
class DroneTestDataset(Dataset):

    def __init__(self, img_path, mask_path, X, transform=None):
        self.img_path = img_path
        self.mask_path = mask_path
        self.X = X
        self.transform = transform

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        img = cv2.imread(self.img_path + self.X[idx] + '.jpg')
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        mask = cv2.imread(self.mask_path + self.X[idx] + '.png', cv2.IMREAD_GRAYSCALE)

        if self.transform is not None:
            aug = self.transform(image=img, mask=mask)
            img = Image.fromarray(aug['image'])
            mask = aug['mask']
```

```
        if self.transform is None:
            img = Image.fromarray(img)

        mask = torch.from_numpy(mask).long()

        return img, mask


t_test = A.Resize(768, 1152, interpolation=cv2.INTER_NEAREST)
test_set = DroneTestDataset(IMAGE_PATH, MASK_PATH, X_test, transform=t_test)
```

Result

In [18]:
```
def predict_image_mask_miou(model, image, mask, mean=[0.485, 0.456, 0.406], std=[0.229
    model.eval()
    t = T.Compose([T.ToTensor(), T.Normalize(mean, std)])
    image = t(image)
    model.to(device); image=image.to(device)
    mask = mask.to(device)
    with torch.no_grad():

        image = image.unsqueeze(0)
        mask = mask.unsqueeze(0)

        output = model(image)
        score = mIoU(output, mask)
        masked = torch.argmax(output, dim=1)
        masked = masked.cpu().squeeze(0)
    return masked, score
```

In [19]:
```
def predict_image_mask_pixel(model, image, mask, mean=[0.485, 0.456, 0.406], std=[0.22
    model.eval()
    t = T.Compose([T.ToTensor(), T.Normalize(mean, std)])
    image = t(image)
    model.to(device); image=image.to(device)
    mask = mask.to(device)
    with torch.no_grad():

        image = image.unsqueeze(0)
        mask = mask.unsqueeze(0)

        output = model(image)
        acc = pixel_accuracy(output, mask)
        masked = torch.argmax(output, dim=1)
        masked = masked.cpu().squeeze(0)
    return masked, acc
```

In [20]:
```
image, mask = test_set[3]
pred_mask, score = predict_image_mask_miou(model, image, mask)
```

In [21]:
```
def miou_score(model, test_set):
    score_iou = []
    for i in tqdm(range(len(test_set))):
        img, mask = test_set[i]
        pred_mask, score = predict_image_mask_miou(model, img, mask)
```

```
        score_iou.append(score)
    return score_iou
```

In [22]:
```
mob_miou = miou_score(model, test_set)
```

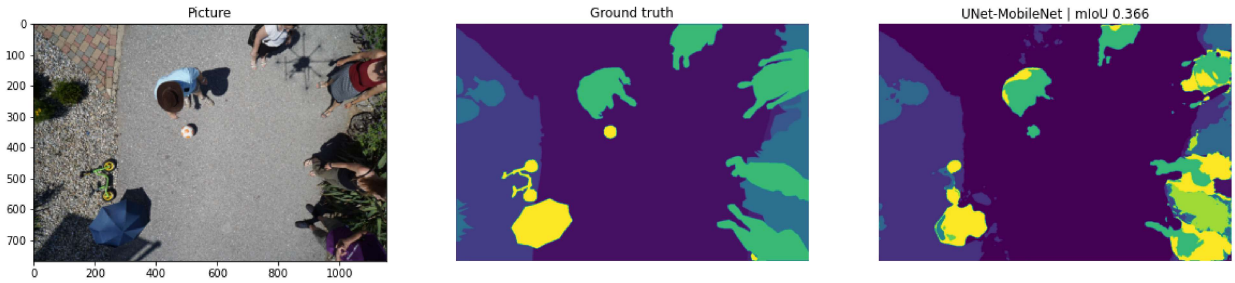
```
  0%|          | 0/40 [00:00<?, ?it/s]
```

In [23]:
```
def pixel_acc(model, test_set):
    accuracy = []
    for i in tqdm(range(len(test_set))):
        img, mask = test_set[i]
        pred_mask, acc = predict_image_mask_pixel(model, img, mask)
        accuracy.append(acc)
    return accuracy
```

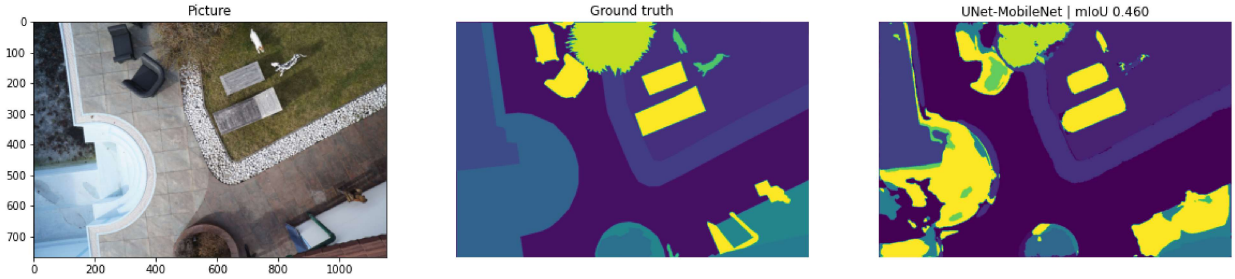
In [24]:
```
mob_acc = pixel_acc(model, test_set)
```

```
  0%|          | 0/40 [00:00<?, ?it/s]
```

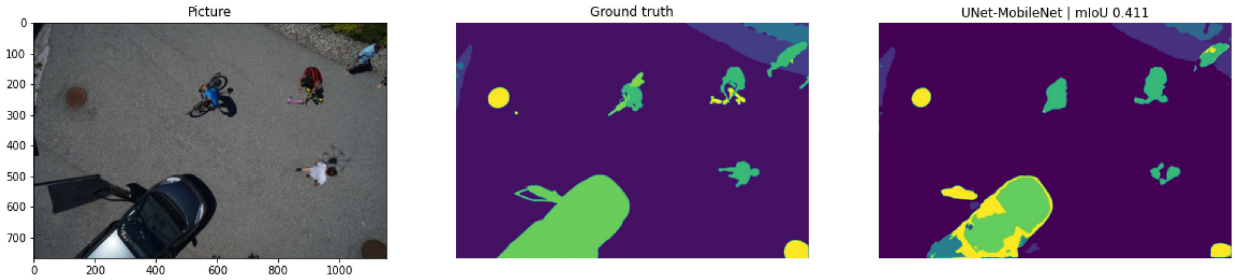
In [25]:
```
fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(20,10))
ax1.imshow(image)
ax1.set_title('Picture');

ax2.imshow(mask)
ax2.set_title('Ground truth')
ax2.set_axis_off()

ax3.imshow(pred_mask)
ax3.set_title('UNet-MobileNet | mIoU {:.3f}'.format(score))
ax3.set_axis_off()
```



In [26]:
```
image2, mask2 = test_set[4]
pred_mask2, score2 = predict_image_mask_miou(model, image2, mask2)

fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(20,10))
ax1.imshow(image2)
ax1.set_title('Picture');

ax2.imshow(mask2)
ax2.set_title('Ground truth')
ax2.set_axis_off()

ax3.imshow(pred_mask2)
ax3.set_title('UNet-MobileNet | mIoU {:.3f}'.format(score2))
ax3.set_axis_off()
```



In [27]:
```
image3, mask3 = test_set[6]
pred_mask3, score3 = predict_image_mask_miou(model, image3, mask3)

fig, (ax1, ax2, ax3) = plt.subplots(1,3, figsize=(20,10))
ax1.imshow(image3)
ax1.set_title('Picture');

ax2.imshow(mask3)
ax2.set_title('Ground truth')
ax2.set_axis_off()

ax3.imshow(pred_mask3)
ax3.set_title('UNet-MobileNet | mIoU {:.3f}'.format(score3))
ax3.set_axis_off()
```



In [28]:
```
print('Test Set mIoU', np.mean(mob_miou))
```

```
Test Set mIoU 0.3526479529490619
```

In [29]:
```
print('Test Set Pixel Accuracy', np.mean(mob_acc))
```

```
Test Set Pixel Accuracy 0.8131760208695024
```