Assignment 4: Pytorch Segmentation

For this assignment, we're goining to use Deep Learning for a new task: semantic segmentation , instead of classification we've been doing. We will also use some common techniques in Deep Learning like pretraining.

Short recap of semantic segmentation

The goal of semantic segmentation is to classify each pixel of the image to a corresponding class of what the pixel represent. One major deference between semantic segmentation and classification is that for semantic segmentation, model output label for each pixel instead of a single label for the whole image

Metrics

In semantic segmentations, we will average pixel-wise accuracy and IoU to benchmark semantic segmentation methods. Here we provide, the code for Evaluation

In [1]:
```
import numpy as np

def _hist(pred, gt, n_class):
#     mask = (label_true >= 0) & (label_true < n_class)
    hist = np.bincount(
        n_class * gt.astype(int) +
        pred, minlength=n_class ** 2
    ).reshape(n_class, n_class)
    return hist


def metrics(preds, gts, n_class):
    hist = np.zeros((n_class, n_class))
    for pred, gt in zip(preds, gts):
        hist += _hist(pred.flatten(), gt.flatten(), n_class)
    acc = np.diag(hist).sum() / hist.sum()
    iou = np.diag(hist) / (
        hist.sum(axis=1) + hist.sum(axis=0) - np.diag(hist)
    )
    mean_iou = np.nanmean(iou)
    return acc, mean_iou
```

CMP Facade DB

In this assignment, we use a new dataset named: CMP Facade Database for semantic segmentation. This dataset is made up with 606 rectified images of the facade of various buildings. The facades are from different cities arount the world with different architectural styles.

CMP Facade DB include 12 semantic classes:

- facade
- molding
- cornice
- pillar
- window
- door
- sill
- blind
- balcony
- shop
- deco
- background

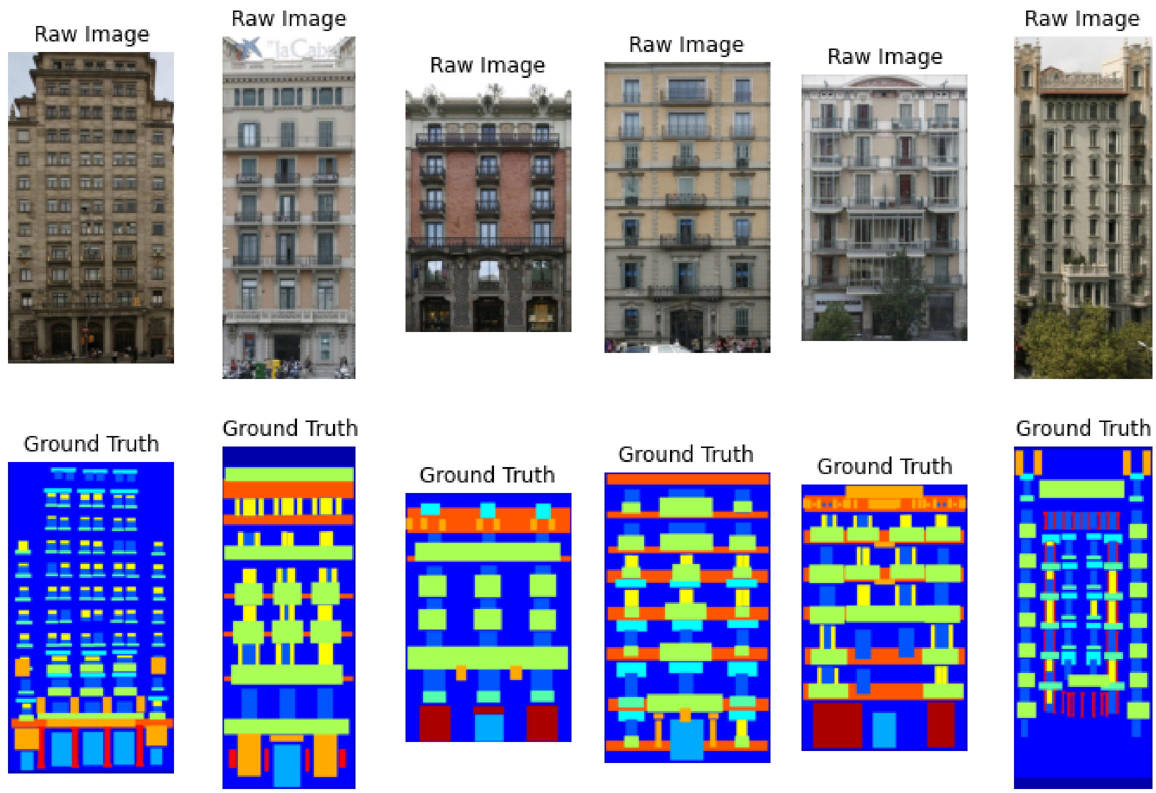
In this assignment, we should use a model to classify each pixel in images to one of these 12 classes.

For more detail about CMP Facade Dataset, if you are intereseted, please check: https://cmp.felk.cvut.cz/~tylecr1/facade/ (https://cmp.felk.cvut.cz/~tylecr1/facade/)

Visualize of the Dataset

In [2]:

```
import matplotlib.pyplot as plt
import numpy as np
# import PI

idxs = [1, 2, 5, 6, 7, 8]
fig, axes = plt.subplots(nrows=2, ncols=6, figsize=(12, 8))
for i, idx in enumerate(idxs):
    pic = plt.imread("dataset/base/cmp_b000{}.jpg".format(idx))
    label = plt.imread("dataset/base/cmp_b000{}.png".format(idx), format="PNG

    axes[0][i].axis('off')
    axes[0][i].imshow(pic)
    axes[0][i].set_title("Raw Image")

    axes[1][i].imshow(label)
    axes[1][i].axis('off')
    axes[1][i].set_title("Ground Truth")
```



Prepare

In [3]:
```
import torch
import copy

USE_GPU = True

dtype = torch.float32 # we will be using float throughout this tutorial

if USE_GPU and torch.cuda.is_available():
    device = torch.device('cuda')
else:
    device = torch.device('cpu')

# Constant to control how frequently we print train loss
print_every = 100

print('using device:', device)
```

```
using device: cuda
```

Build Dataset Class in Pytorch

In [4]:

```
import torch
import PIL
from torch.utils.data import Dataset
import os
import os.path as osp
import torchvision.transforms as transforms
from PIL import Image


def get_full_list(
    root_dir,
    base_dir="base",
    extended_dir="extended",
):
    data_list = []
    for name in [base_dir, extended_dir]:
        data_dir = osp.join(
            root_dir, name
        )
        data_list += sorted(
            osp.join(data_dir, img_name) for img_name in
            filter(
                lambda x: x[-4:] == '.jpg',
                os.listdir(data_dir)
            )
        )
    return data_list

class CMP_Facade_DB(Dataset):
    def __init__(
        self,
        data_list
    ):
        self.data_list = data_list

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

    def __getitem__(self, i):
        # input and target images
        in_name = self.data_list[i]
        gt_name = self.data_list[i].replace('.jpg','.png')

        # process the images
        normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                         std=[0.229, 0.224, 0.225])
        transf = transforms.Compose([
            transforms.ToTensor(),
            normalize
        ])
        in_image = transf(
            Image.open(in_name).convert('RGB')
        )
        gt_im = Image.open(gt_name)

        gt_label = torch.LongTensor(
```

```
            np.frombuffer(gt_im.tobytes(), dtype=np.ubyte).reshape(
                in_image.shape[1:]
            )
        ) - 1

        return in_image, gt_label

    def revert_input(self, img, label):
        img = np.transpose(img.cpu().numpy(), (1, 2, 0))
        std_img = np.array([0.229, 0.224, 0.225]).reshape((1, 1, -1))
        mean_img = np.array([0.485, 0.456, 0.406]).reshape((1, 1, -1))
        img *= std_img
        img += mean_img
        label = label.cpu().numpy()
        return img, label + 1

TRAIN_SIZE = 500
VAL_SIZE = 30
TEST_SIZE = 70
full_data_list = get_full_list("dataset")

train_data_set = CMP_Facade_DB(full_data_list[: TRAIN_SIZE])
val_data_set = CMP_Facade_DB(full_data_list[TRAIN_SIZE: TRAIN_SIZE + VAL_SIZE
test_data_set = CMP_Facade_DB(full_data_list[TRAIN_SIZE + VAL_SIZE:])

print("Training Set Size:", len(train_data_set))
print("Validation Set Size:", len(val_data_set))
print("Test Set Size:", len(test_data_set))
```

```
Training Set Size: 500
Validation Set Size: 30
Test Set Size: 76
```

Fully Convolutional Networks for Semantic Segmentation

We've seen that CNNs are powerful models to get hiereachical visual features in Deep Learning. There we are going to explore the classical work: "Fully Convolutional Networks for Semantic Segmentation"(FCN).

Though we've already used CNN models for image classifications in the previous assignemtn, those models have one major drawback: Those model take input with fixed shape and output a single vector. However, in semantic segmentation, we want the model to be able to process image with arbitary shape and predict the label map with the same shape as the input image.

In FCN, the model utilize the Transpose Convolution layers, which we've already learned during the lecture, to make it happen. For the overal introduction of Transpose Convolution and Fully Convolutional Networks, please review the lecture recording and lecture slides on Canvas(Lecture 10).

Here we do not cover all the details in FCN. If you need more reference, you can check the original paper: https://arxiv.org/pdf/1411.4038.pdf (https://arxiv.org/pdf/1411.4038.pdf) and some other materials online.

Besides of transpose Convolution, there are also some difference compared with the models we've been working on:

- Use 1x1 Convolution to replace fully connected layers to output score for each class.
- Use skip connection to combine high-level feature and local feature.

Naive FCN: FCN-32s (30%)

In this section, we first try to implement naive variant of FCN without skip connection: FCN-32s. Here we use FCN-32s with VGG-16 architecture for feature encoding.

Compared with VGG-16, FCN-32s only replace the fully connecteed layers with 1x1 convolution and add a Transpose Convolution at the end to output dense prediction.

FC-32s architecture:

The following Conv use kernel size = 3, padding = 1, stride =1(except conv1_1. conv1_1 should use padding = 100)

- [conv1_1(3,64)-relu] -> [conv1_2(64,64)-relu] -> [maxpool1(2,2)]
- [conv2_1(64,128)-relu] -> [conv2_2(128,128)-relu] -> [maxpool2(2,2)]
- [conv3_1(128,256)-relu] -> [conv3_2(256,256)-relu] ->[conv3_3(256,256)-relu] -> [maxpool3(2,2)]
- [conv4_1(256,512)-relu] -> [conv4_2(512,512)-relu] ->[conv4_3(512,512)-relu] -> [maxpool3(2,2)]
- [conv5_1(512,512)-relu] -> [conv5_2(512,512)-relu] ->[conv5_3(512,512)-relu] -> [maxpool3(2,2)]

The following Conv use kernel size = 7, stride = 1, padding = 0

- [fc6=conv(512, 4096, 7)-relu-dropout2d]

The following Conv use kernel size = 1, stride = 1, padding = 0

- [fc7=conv1x1(4096, 4096)-relu-dropout2d]
- [score=conv1x1(4096, num_classes)]

The transpose convolution: kernal size = 64, stride = 32, bias = False

- [transpose_conv(n_class, n_class)]

Note: The output of the transpose convlution might not have the same shape as the input, take [19: 19 + input_image_width], [19: 19 + input_image_height] for width and height dimension of the output to get the output with the same shape as the input

It's expected that you model perform very poor in this section

Try to name the layers use the name provide above to ensure the next section works correctly, and use a new nn.RELU() for each activation

In [5]:

```
ort torch.nn as nn

 get_upsampling_weight(in_channels, out_channels, kernel_size):
  """Make a 2D bilinear kernel suitable for upsampling"""
  factor = (kernel_size + 1) // 2
  if kernel_size % 2 == 1:
      center = factor - 1
  else:
      center = factor - 0.5
  og = np.ogrid[:kernel_size, :kernel_size]
  filt = (1 - abs(og[0] - center) / factor) * \
         (1 - abs(og[1] - center) / factor)
  weight = np.zeros((in_channels, out_channels, kernel_size, kernel_size),
                    dtype=np.float64)
  weight[range(in_channels), range(out_channels), :, :] = filt
  return torch.from_numpy(weight).float()

ss FCN32s(nn.Module):
  def __init__(self, n_class=12):
      super(FCN32s, self).__init__()

      self.conv1_1=nn.Conv2d(3,64,kernel_size = 3, stride=1,padding=100)
      self.relu1_1=nn.ReLU()
      self.conv1_2=nn.Conv2d(64,64,kernel_size = 3, stride=1,padding=1)
      self.relu1_2=nn.ReLU()
      self.pool1=nn.MaxPool2d(2,stride=2,ceil_mode = True)

      self.conv2_1=nn.Conv2d(64,128,kernel_size = 3, stride=1,padding=1)
      self.relu2_1=nn.ReLU()
      self.conv2_2=nn.Conv2d(128,128,kernel_size = 3, stride=1,padding=1)
      self.relu2_2=nn.ReLU()
      self.pool2=nn.MaxPool2d(2,stride=2,ceil_mode = True)

      self.conv3_1=nn.Conv2d(128,256,kernel_size = 3, stride=1,padding=1)
      self.relu3_1=nn.ReLU()
      self.conv3_2=nn.Conv2d(256,256,kernel_size = 3, stride=1,padding=1)
      self.relu3_2=nn.ReLU()
      self.conv3_3=nn.Conv2d(256,256,kernel_size = 3, stride=1,padding=1)
      self.relu3_3=nn.ReLU()
      self.pool3=nn.MaxPool2d(2,stride=2,ceil_mode = True)

      self.conv4_1=nn.Conv2d(256,512,kernel_size = 3, stride=1,padding=1)
      self.relu4_1=nn.ReLU()
      self.conv4_2=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
      self.relu4_2=nn.ReLU()
      self.conv4_3=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
      self.relu4_3=nn.ReLU()
      self.pool4=nn.MaxPool2d(2,stride=2,ceil_mode = True)

      self.conv5_1=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
      self.relu5_1=nn.ReLU()
      self.conv5_2=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
      self.relu5_2=nn.ReLU()
      self.conv5_3=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
      self.relu5_3=nn.ReLU()
      self.pool5=nn.MaxPool2d(2,stride=2,ceil_mode = True)
```

```
        self.fc6=nn.Conv2d(512,4096,kernel_size=7,stride=1,padding=0)
        self.relu6=nn.ReLU()
        self.dropout=nn.Dropout2d()
        self.fc7=nn.Conv2d(4096,4096,kernel_size=1,stride=1,padding=0)
        self.relu7=nn.ReLU()
        self.dropout2=nn.Dropout2d()
        self.score=nn.Conv2d(4096,n_class,kernel_size=1,stride=1,padding=0)
        self.transpose=nn.ConvTranspose2d(n_class,n_class,kernel_size=64,stride=

        self._initialize_weights()

    def _initialize_weights(self):
        for m in self.modules():
              if isinstance(m, nn.Conv2d):
                  m.weight.data.zero_()
                  if m.bias is not None:
                      m.bias.data.zero_()
            if isinstance(m, nn.ConvTranspose2d):
                assert m.kernel_size[0] == m.kernel_size[1]
                initial_weight = get_upsampling_weight(
                    m.in_channels, m.out_channels, m.kernel_size[0])
                m.weight.data.copy_(initial_weight)

    def forward(self, x):

        h=self.pool1(self.relu1_2(self.conv1_2(self.relu1_1(self.conv1_1(x)))))
        h=self.pool2(self.relu2_2(self.conv2_2(self.relu2_1(self.conv2_1(h)))))
        h=self.pool3(self.relu3_3(self.conv3_3(self.relu3_2(self.conv3_2(self.re
        h=self.pool4(self.relu4_3(self.conv4_3(self.relu4_2(self.conv4_2(self.re
        h=self.relu5_1(self.conv5_1(h))
        h= self.transpose(self.score(self.dropout2(self.relu7(self.fc7(self.drop
        h = h[:,:,19: 19 + x.size()[2],19: 19 + x.size()[3]]

        return h


    def copy_params_from_vgg16(self, vgg16):
        features = [
            self.conv1_1, self.relu1_1,
            self.conv1_2, self.relu1_2,
            self.pool1,
            self.conv2_1, self.relu2_1,
            self.conv2_2, self.relu2_2,
            self.pool2,
            self.conv3_1, self.relu3_1,
            self.conv3_2, self.relu3_2,
            self.conv3_3, self.relu3_3,
            self.pool3,
            self.conv4_1, self.relu4_1,
            self.conv4_2, self.relu4_2,
            self.conv4_3, self.relu4_3,
            self.pool4,
            self.conv5_1, self.relu5_1,
            self.conv5_2, self.relu5_2,
            self.conv5_3, self.relu5_3,
            self.pool5,
```

```
    ]
    for l1, l2 in zip(vgg16.features, features):
        if isinstance(l1, nn.Conv2d) and isinstance(l2, nn.Conv2d):
            assert l1.weight.size() == l2.weight.size()
            assert l1.bias.size() == l2.bias.size()
            l2.weight.data = l1.weight.data
            l2.bias.data = l1.bias.data
    for i, name in zip([0, 3], ['fc6', 'fc7']):
        l1 = vgg16.classifier[i]
        l2 = getattr(self, name)
        l2.weight.data = l1.weight.data.view(l2.weight.size())
        l2.bias.data = l1.bias.data.view(l2.bias.size())
```

In [6]:

```
# You can change it if you want
lr = 1e-4
weight_decay = 2e-5
```

In [7]:

```
train_loader = torch.utils.data.DataLoader(
    train_data_set, batch_size=1, shuffle=True
)
val_loader = torch.utils.data.DataLoader(
    val_data_set, batch_size=1, shuffle=True
)
test_loader = torch.utils.data.DataLoader(
    test_data_set, batch_size=1, shuffle=True
)
```

In [8]:
```
def Evaluate(
    val_loader,
    model,
    current_best,
    n_class=12
):
    val_loss = 0
    visualizations = []
    preds, gts = [], []

    model.eval()
    for batch_idx, (data, target) in enumerate(val_loader):
        data, target = data.to(device), target.to(device)
        with torch.no_grad():
            score = model(data)

        pred = score.max(1)[1].cpu().numpy()
        gt = target.cpu().numpy()
        preds.append(pred)
        gts.append(gt)

    avg_acc, mean_iou = metrics(
        preds, gts, n_class)

    if mean_iou > current_best["IoU"]:
        current_best["IoU"] = mean_iou
        current_best["model"] = copy.deepcopy(model)

    return avg_acc, mean_iou, current_best

def Train(
    model,
    loss_func,
    optim,
    scheduler,
    epochs,
    train_loader,
    val_lodaer,
    test_loader,
    display_interval = 100
):

    current_best = {
        "IoU": 0,
        "model": model
    }
    avg_acc, mean_iou, current_best = Evaluate(
        val_loader,
        model,
        current_best
    )

    print("Init Model")
    print("Avg Acc: {:.4}, Mean IoU: {:.4}".format(
        avg_acc, mean_iou
    ))
```

```
    for i in range(epochs):
        print("Epochs: {}".format(i))
        total_loss = 0
        model.train()
        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to("cuda:0"), target.to("cuda:0")
            optim.zero_grad()

            score = model(data)
            loss = loss_func(score, target.squeeze(1))
            loss_data = loss.item()
            if np.isnan(loss_data):
                raise ValueError('loss is nan while training')
            loss.backward()
            optim.step()
            total_loss += loss.item()
            if batch_idx % display_interval == 0 and batch_idx != 0:
                print("{} / {}, Current Avg Loss:{:.4}".format(
                    batch_idx, len(train_loader), total_loss / (batch_idx + 1
                ))


        total_loss /= len(train_loader)
        model.eval()
        avg_acc, mean_iou, current_best = Evaluate(
            val_loader,
            model,
            current_best
        )
        scheduler.step(total_loss)
        print("Avg Loss: {:.4}, Avg Acc: {:.4}, Mean IoU: {:.4}".format(
            total_loss, avg_acc, mean_iou
        ))

    test_acc, test_iou, current_best = Evaluate(
        val_loader,
        current_best["model"],
        current_best
    )
    print("Test Acc: {:.4}, Test Mean IoU: {:.4}".format(
        test_acc, test_iou
    ))
    return current_best["model"]
```

In [9]:

```
model = FCN32s(n_class=12)
model.to(device)

optim = torch.optim.Adam(
    model.parameters(),
    lr=lr,
    weight_decay=weight_decay,
)
from torch.optim.lr_scheduler import ReduceLROnPlateau
scheduler = ReduceLROnPlateau(
    optim, 'min', patience=3,
    min_lr=1e-10, verbose=True
)

# Choose the right loss function in torch.nn
loss_func = nn.CrossEntropyLoss()
best_model = Train(
    model,
    loss_func,
    optim,
    scheduler,
    5,
    train_loader,
    val_loader,
    test_loader
)
```

```
Init Model
Avg Acc: 0.02884, Mean IoU: 0.002403
Epochs: 0
100 / 500, Current Avg Loss:2.025
200 / 500, Current Avg Loss:1.964
300 / 500, Current Avg Loss:1.965
400 / 500, Current Avg Loss:1.952
Avg Loss: 1.934, Avg Acc: 0.3922, Mean IoU: 0.05361
Epochs: 1
100 / 500, Current Avg Loss:1.838
200 / 500, Current Avg Loss:1.821
300 / 500, Current Avg Loss:1.817
400 / 500, Current Avg Loss:1.811
Avg Loss: 1.797, Avg Acc: 0.3997, Mean IoU: 0.05516
Epochs: 2
100 / 500, Current Avg Loss:1.745
200 / 500, Current Avg Loss:1.76
300 / 500, Current Avg Loss:1.755
400 / 500, Current Avg Loss:1.762
Avg Loss: 1.762, Avg Acc: 0.3462, Mean IoU: 0.02983
Epochs: 3
100 / 500, Current Avg Loss:1.932
200 / 500, Current Avg Loss:1.868
300 / 500, Current Avg Loss:1.827
400 / 500, Current Avg Loss:1.809
Avg Loss: 1.801, Avg Acc: 0.3999, Mean IoU: 0.05388
Epochs: 4
100 / 500, Current Avg Loss:1.734
200 / 500, Current Avg Loss:1.718
```

```
300 / 500, Current Avg Loss:1.738
400 / 500, Current Avg Loss:1.743
Avg Loss: 1.741, Avg Acc: 0.4189, Mean IoU: 0.06268
Test Acc: 0.4189, Test Mean IoU: 0.06268
```

Visualize Output

In this section, we visualize several model outputs to see how our model actually perform.

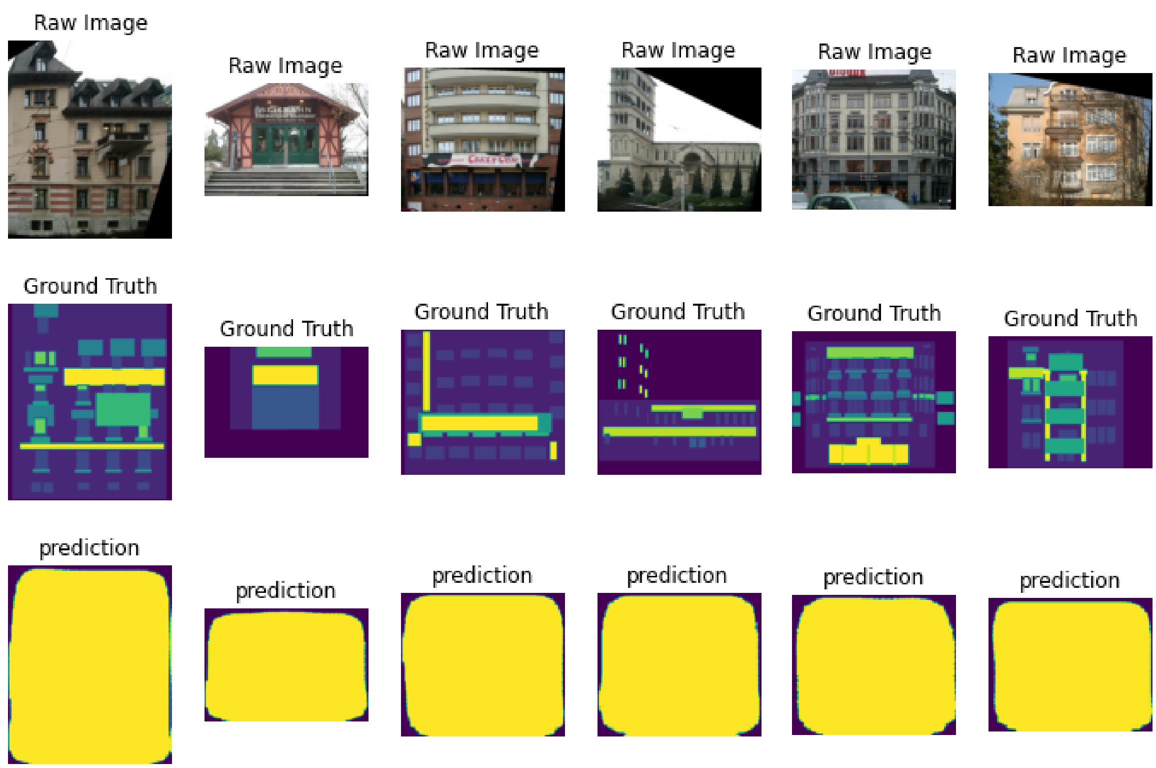
In [12]:

```
def visualize(model, test_loader):
    idxs = [1, 2, 5, 6, 7, 8]
    fig, axes = plt.subplots(nrows=3, ncols=6, figsize=(12, 8))
    model.eval()
    for i, idx in enumerate(idxs):
        img, label = test_loader.dataset[idx]

        pred = model(img.unsqueeze(0).to(device))
        pred = (pred.max(1)[1] + 1).squeeze(0).cpu().numpy()

        img, label = test_loader.dataset.revert_input(img, label)

        axes[0][i].axis('off')
        axes[0][i].imshow(img)
        axes[0][i].set_title("Raw Image")

        axes[1][i].imshow(label)
        axes[1][i].axis('off')
        axes[1][i].set_title("Ground Truth")

        axes[2][i].imshow(pred)
        axes[2][i].axis('off')
        axes[2][i].set_title("prediction")
```

In [11]:

```
visualize(best_model, test_loader)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



Utilize the pretrain features

In the previous section, we use the random initalized weights to train FCN-32S from scrath. We can see that it perform poorly. In this section, we utilize the feature from pretrained model(In our case, we use VGG-16) to help us get a better result.

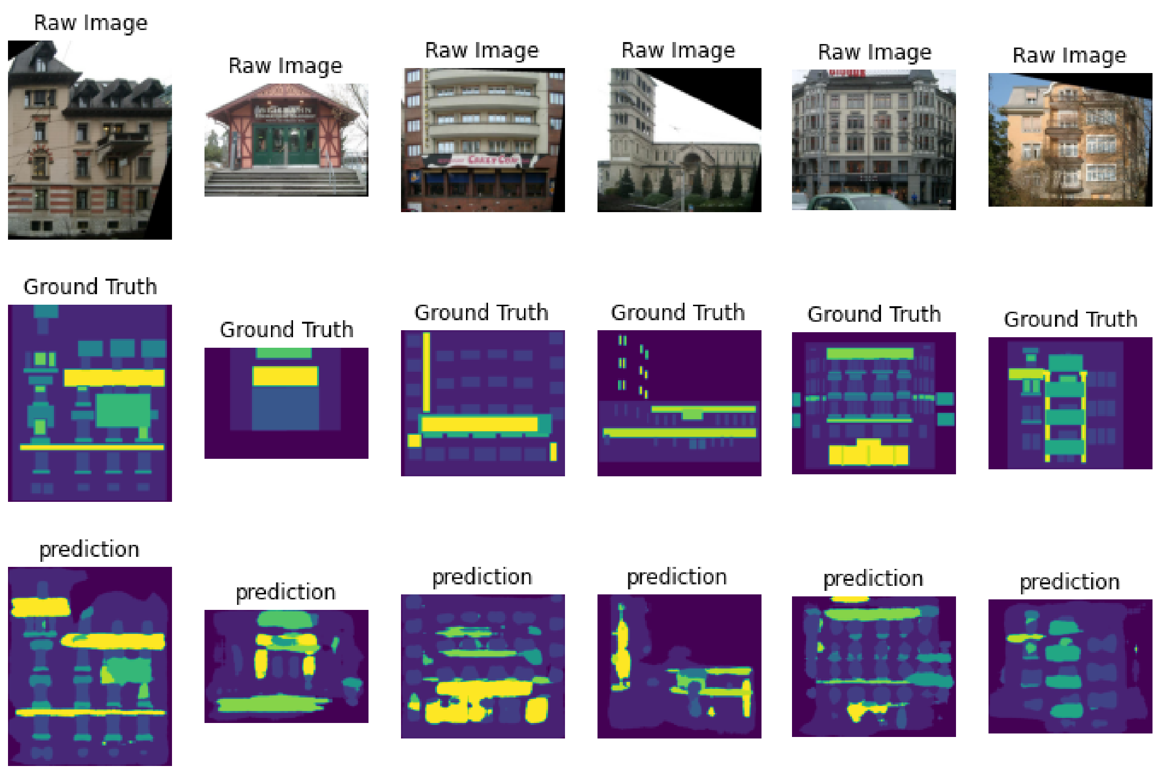
In [9]:

```
import torchvision
vgg16 = torchvision.models.vgg16(pretrained=True)

model = FCN32s(n_class=12)
model.copy_params_from_vgg16(vgg16)
model.to(device)

optim = torch.optim.Adam(
    model.parameters(),
    lr=lr,
    weight_decay=weight_decay,
)
from torch.optim.lr_scheduler import ReduceLROnPlateau
scheduler = ReduceLROnPlateau(
    optim, 'min', patience=3,
    min_lr=1e-10, verbose=True
)

loss_func = nn.CrossEntropyLoss()
best_model_pretrain = Train(
    model,
    loss_func,
    optim,
    scheduler,
    25,
    train_loader,
    val_loader,
    test_loader
)
```

```
400 / 500, Current Avg Loss:0.3717
Avg Loss: 0.3678, Avg Acc: 0.7169, Mean IoU: 0.4693
Epochs: 14
100 / 500, Current Avg Loss:0.3552
200 / 500, Current Avg Loss:0.3554
300 / 500, Current Avg Loss:0.3553
400 / 500, Current Avg Loss:0.3544
Avg Loss: 0.3547, Avg Acc: 0.7218, Mean IoU: 0.4755
Epochs: 15
100 / 500, Current Avg Loss:0.325
200 / 500, Current Avg Loss:0.3378
300 / 500, Current Avg Loss:0.337
400 / 500, Current Avg Loss:0.3332
Avg Loss: 0.334, Avg Acc: 0.7223, Mean IoU: 0.4649
Epochs: 16
100 / 500, Current Avg Loss:0.321
200 / 500, Current Avg Loss:0.3279
300 / 500, Current Avg Loss:0.3242
400 / 500, Current Avg Loss:0.3262
Avg Loss: 0.3245, Avg Acc: 0.7228, Mean IoU: 0.4675
```

In [12]:
```
visualize(best_model_pretrain, test_loader)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



Skip Connection: FCN-8s(40%)

Though we've get a prety good result using FCN-32s with VGG-16 pretrain. We can actully do better with another technique introduced in FCN paper: Skip Connection.

With skip connection, we are supposed to get a better performance especially for some details.

Here we provide the structure of FCN-8s, the variant of FCN with skip connections.

FCN-8s architecture:

The following Conv use kernel size = 3, padding = 1, stride =1(except conv1_1. conv1_1 should use padding = 100) **As you can see, the structure of this part is the same as FCN-32s**

- [conv1_1(3,64)-relu] -> [conv1_2(64,64)-relu] -> [maxpool1(2,2)]
- [conv2_1(64,128)-relu] -> [conv2_2(128,128)-relu] -> [maxpool2(2,2)]
- [conv3_1(128,256)-relu] -> [conv3_2(256,256)-relu] ->[conv3_3(256,256)-relu] -> [maxpool3(2,2)]
- [conv4_1(256,512)-relu] -> [conv4_2(512,512)-relu] ->[conv4_3(512,512)-relu] -> [maxpool3(2,2)]
- [conv5_1(512,512)-relu] -> [conv5_2(512,512)-relu] ->[conv5_3(512,512)-relu] -> [maxpool3(2,2)]

The following Conv use kernel size = 7, stride = 1, padding = 0

- [fc6=conv(512, 4096, 7)-relu-dropout2d]

The following Conv use kernel size = 1, stride = 1, padding = 0

- [fc7=conv1x1(4096, 4096)-relu-dropout2d]
- [score=conv1x1(4096, num_classes)]

The Additional Score Pool use kernel size = 1, stride = 1, padding = 0

- [score_pool_3 =conv1x1(256, num_classes)]
- [score_pool_4 =conv1x1(512, num_classes)]

The transpose convolution: kernal size = 4, stride = 2, bias = False

- [upscore1 = transpose_conv(n_class, n_class)]

The transpose convolution: kernal size = 4, stride = 2, bias = False

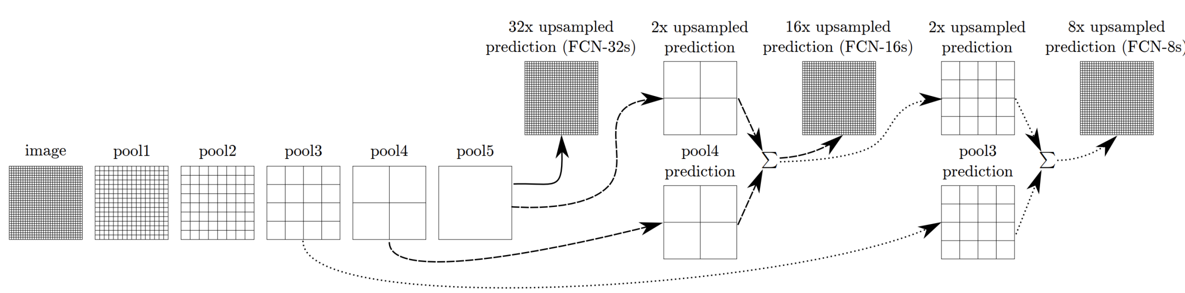
- [upscore2 = transpose_conv(n_class, n_class)]

The transpose convolution: kernal size = 16, stride = 8, bias = False

- [upscore3 = transpose_conv(n_class, n_class)]

Different from FCN-32s which has only single path from input to output, there are multiple data path from input to output in FCN-8s.

The following graph is from original FCN paper, you can also find the graph there.



"Layers are shown as grids that reveal relative spatial coarseness. Only pooling and prediction layers are shown; intermediate convolution layers (including converted fully connected layers) are omitted. " ---- FCN

Detailed path specification:

- score_pool_3
  - input: output from layer "pool3"
  - take [9: 9 + upscore2_width], [9: 9 + upscore2_height]
- score_pool_4,
  - input: output from layer "pool4"
  - take [5: 5 + upscore1_width], [5: 5 + upscore1_height]

- upscore1
  - input: output from layer "score"
- upscore2:
  - input: output from layer "score_pool_4" + output from layer "upscore1"
- upscore3:
  - input: output from layer "score_pool_3" + output from layer "upscore2"
  - take [31: 31 + input_image_width], [31: 31 + input_image_height]

In [9]:

```
import torch.nn as nn

class FCN8s(nn.Module):

    def __init__(self, n_class=12):
        super(FCN8s, self).__init__()

        self.conv1_1=nn.Conv2d(3,64,kernel_size = 3, stride=1,padding=100)
        self.relu1_1=nn.ReLU()
        self.conv1_2=nn.Conv2d(64,64,kernel_size = 3, stride=1,padding=1)
        self.relu1_2=nn.ReLU()
        self.pool1=nn.MaxPool2d(2,stride=2,ceil_mode = True)

        self.conv2_1=nn.Conv2d(64,128,kernel_size = 3, stride=1,padding=1)
        self.relu2_1=nn.ReLU()
        self.conv2_2=nn.Conv2d(128,128,kernel_size = 3, stride=1,padding=1)
        self.relu2_2=nn.ReLU()
        self.pool2=nn.MaxPool2d(2,stride=2,ceil_mode = True)

        self.conv3_1=nn.Conv2d(128,256,kernel_size = 3, stride=1,padding=1)
        self.relu3_1=nn.ReLU()
        self.conv3_2=nn.Conv2d(256,256,kernel_size = 3, stride=1,padding=1)
        self.relu3_2=nn.ReLU()
        self.conv3_3=nn.Conv2d(256,256,kernel_size = 3, stride=1,padding=1)
        self.relu3_3=nn.ReLU()
        self.pool3=nn.MaxPool2d(2,stride=2,ceil_mode = True)

        self.conv4_1=nn.Conv2d(256,512,kernel_size = 3, stride=1,padding=1)
        self.relu4_1=nn.ReLU()
        self.conv4_2=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
        self.relu4_2=nn.ReLU()
        self.conv4_3=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
        self.relu4_3=nn.ReLU()
        self.pool4=nn.MaxPool2d(2,stride=2,ceil_mode = True)

        self.conv5_1=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
        self.relu5_1=nn.ReLU()
        self.conv5_2=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
        self.relu5_2=nn.ReLU()
        self.conv5_3=nn.Conv2d(512,512,kernel_size = 3, stride=1,padding=1)
        self.relu5_3=nn.ReLU()
        self.pool5=nn.MaxPool2d(2,stride=2,ceil_mode = True)

        self.fc6=nn.Conv2d(512,4096,kernel_size=7,stride=1,padding=0)
        self.relu6=nn.ReLU()
        self.dropout=nn.Dropout2d()
        self.fc7=nn.Conv2d(4096,4096,kernel_size=1,stride=1,padding=0)
        self.relu7=nn.ReLU()
        self.dropout2=nn.Dropout2d()
        self.score=nn.Conv2d(4096,n_class,kernel_size=1,stride=1,padding=0)
        self.score3=nn.Conv2d(256,n_class,kernel_size=1,stride=1,padding=0)
        self.score4=nn.Conv2d(512,n_class,kernel_size=1,stride=1,padding=0)
        self.transpose1=nn.ConvTranspose2d(n_class,n_class,kernel_size=4,stri
        self.transpose2=nn.ConvTranspose2d(n_class,n_class,kernel_size=4,stri
        self.transpose3=nn.ConvTranspose2d(n_class,n_class,kernel_size=16,str
```

```
        self._initialize_weights()

    def _initialize_weights(self):
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                m.weight.data.zero_()
                if m.bias is not None:
                    m.bias.data.zero_()
            if isinstance(m, nn.ConvTranspose2d):
                assert m.kernel_size[0] == m.kernel_size[1]
                initial_weight = get_upsampling_weight(
                    m.in_channels, m.out_channels, m.kernel_size[0])
                m.weight.data.copy_(initial_weight)


    def forward(self, x):

        h=self.pool1(self.relu1_2(self.conv1_2(self.relu1_1(self.conv1_1(x)))
        h=self.pool2(self.relu2_2(self.conv2_2(self.relu2_1(self.conv2_1(h)))
        h=self.pool3(self.relu3_3(self.conv3_3(self.relu3_2(self.conv3_2(self
        step3=h
        h=self.pool4(self.relu4_3(self.conv4_3(self.relu4_2(self.conv4_2(self
        step4=h
        h=self.relu5_1(self.conv5_1(h))
        h= self.transpose1(self.score(self.dropout2(self.relu7(self.fc7(self.
        upscore=h
        h=self.score4(step4)
        h=h[:,:,5:5+upscore.size()[2],5:5+upscore.size()[3]]
        score4=h
        h=upscore+score4
        h=self.transpose2(h)
        upscore4=h
        h=self.score3(step3)
        h=h[:,:,9:9+upscore4.size()[2],9:9+upscore4.size()[3]]
        scorepool3=h
        h=upscore4+scorepool3
        h=self.transpose3(h)
        h=h[:,:,31:31+x.size()[2],31:31+x.size()[3]]

        return h

    def copy_params_from_vgg16(self, vgg16):
        features = [
            self.conv1_1, self.relu1_1,
            self.conv1_2, self.relu1_2,
            self.pool1,
            self.conv2_1, self.relu2_1,
            self.conv2_2, self.relu2_2,
            self.pool2,
            self.conv3_1, self.relu3_1,
            self.conv3_2, self.relu3_2,
            self.conv3_3, self.relu3_3,
            self.pool3,
            self.conv4_1, self.relu4_1,
            self.conv4_2, self.relu4_2,
            self.conv4_3, self.relu4_3,
            self.pool4,
```

```
            self.conv5_1, self.relu5_1,
            self.conv5_2, self.relu5_2,
            self.conv5_3, self.relu5_3,
            self.pool5,
        ]
        for l1, l2 in zip(vgg16.features, features):
            if isinstance(l1, nn.Conv2d) and isinstance(l2, nn.Conv2d):
                assert l1.weight.size() == l2.weight.size()
                assert l1.bias.size() == l2.bias.size()
                l2.weight.data.copy_(l1.weight.data)
                l2.bias.data.copy_(l1.bias.data)
        for i, name in zip([0, 3], ['fc6', 'fc7']):
            l1 = vgg16.classifier[i]
            l2 = getattr(self, name)
            l2.weight.data.copy_(l1.weight.data.view(l2.weight.size()))
            l2.bias.data.copy_(l1.bias.data.view(l2.bias.size()))
```

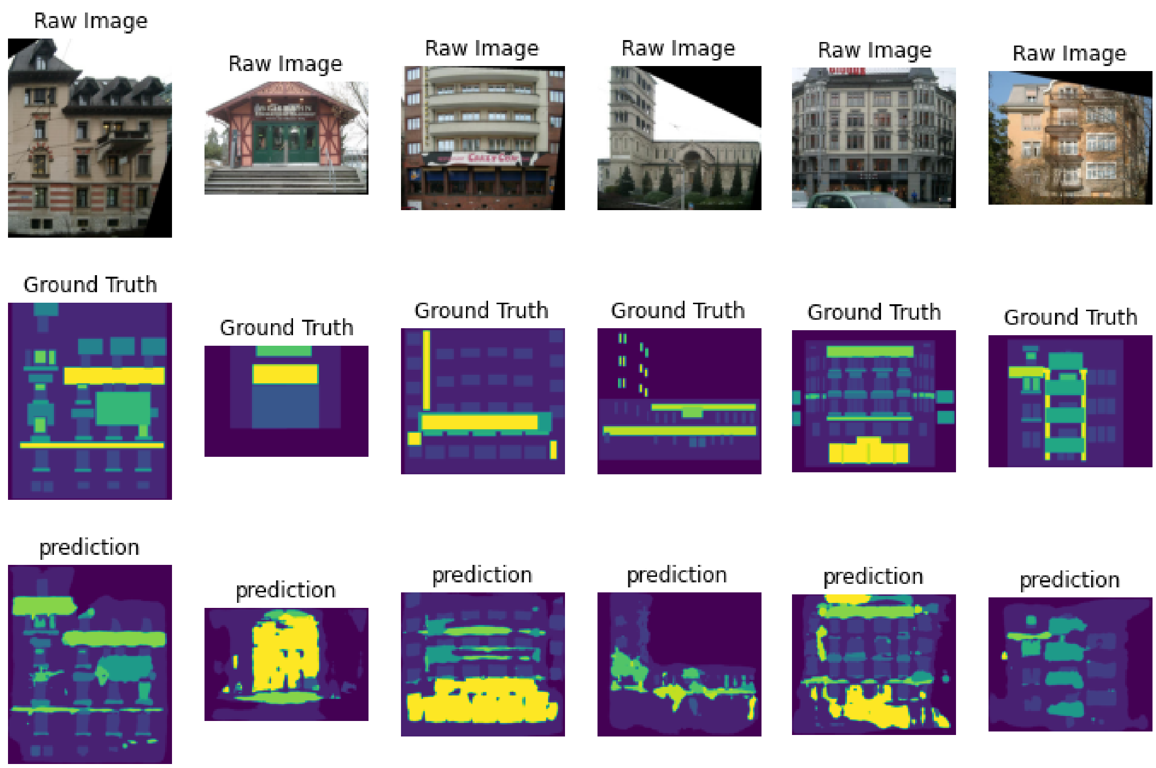
In [10]:

```
import torchvision
vgg16 = torchvision.models.vgg16(pretrained=True)

model = FCN8s(n_class=12)
model.copy_params_from_vgg16(vgg16)
model.to(device)

optim = torch.optim.Adam(
    model.parameters(),
    lr=lr,
    weight_decay=weight_decay,
#     momentum=momentum,
)

from torch.optim.lr_scheduler import ReduceLROnPlateau
scheduler = ReduceLROnPlateau(
    optim, 'min', patience=3,
    min_lr=1e-10, verbose=True
)
loss_func = nn.CrossEntropyLoss()
best_model_fcn8s = Train(
    model,
    loss_func,
    optim,
    scheduler,
    25,
    train_loader,
    val_loader,
    test_loader
)
```

```
Avg Loss: 0.2293, Avg Acc: 0.7322, Mean IoU: 0.5
Epochs: 22
100 / 500, Current Avg Loss:0.2256
200 / 500, Current Avg Loss:0.2297
300 / 500, Current Avg Loss:0.2294
400 / 500, Current Avg Loss:0.2255
Avg Loss: 0.2205, Avg Acc: 0.7407, Mean IoU: 0.4877

Epochs: 23
100 / 500, Current Avg Loss:0.2019
200 / 500, Current Avg Loss:0.1998
300 / 500, Current Avg Loss:0.2049
400 / 500, Current Avg Loss:0.2026
Avg Loss: 0.2034, Avg Acc: 0.7368, Mean IoU: 0.4996
Epochs: 24
100 / 500, Current Avg Loss:0.2196
200 / 500, Current Avg Loss:0.2101
300 / 500, Current Avg Loss:0.2161
400 / 500, Current Avg Loss:0.2134
Avg Loss: 0.2115, Avg Acc: 0.7377, Mean IoU: 0.4858
```

In [13]:
```
visualize(best_model_fcn8s, test_loader)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or [0..255] for integers).
```



Inline Questions(30%):

Inline Question 1: Why using pretrained model to initialized our model(FCN-32s) helps a lot? Please give at least two specific reasons

Your Answer:

Inline Question 2: Compare the performance and visualization of FCN-32s and FCN-8s. Please state the differnece, and provide some explanation. You can visualize more images than we provide, if it's necessary for you to see the difference.

Your Answer:

In [17]:
```
#Question 1:
#Pretrained helps a lot because the models have already been trained
#on previous large datasets so we can already
#use the wieghts obtained from already trained model.
#Since the majority of the weights and model are already
#pretrained to fit our data, our FCN-32 model can learn more
#fine detail and has more epochs/time to make the model
#more precise to our data.
```

In [15]:
```
#Question 2:
#The performance of FCN-8s are a little better than the pretrained
#FCN-32s as the FCN-8s has test accuracy of 0.7322 and a
#Test Mean IoU of 0.5 while the FCN-32 has a test accuracy
#of 0.7237 and a test mean IoU of 0.4789. However the prediction
#for some raw images are better for FCN-32s, such as the second
#raw image's prediction is better for FCN-32 while the
#prediction of the first raw image is better for FCN-8s.We get
#better results for FCN-8s in general because we skip connection
```