
Final Project Report

Student: Cao Viet Hai Nam
ID: 20200817

I. OVERVIEW

- For the task of the project to do semantic segmentation of photographic images, I was able to achieve the highest mean Intersection of Union (mIoU) of **0.6215** on the test dataset. The training elapsed time is **3 hours and 4 minitues**.

```
2021-06-07 02:48:50 || [99/100], train_loss = 4.3356, valid_loss = 5.7314, train_mIoU = 0.8684, val_mIoU = 0.5241, , best mIoU = 0.5241

End of training, elapsed time : 184.0 min 37.20937156677246 sec.
```

- The model I used for this project is UNet based on the research paper: *"U-Net: Convolutional Networks for Biomedical Image Segmentation"* *(1)* with some specific modifications that I've mentioned in detail in **section III** of the report**:** Model architecture.

- In the **last section**, I describe some of my attempts to reach the final performance and how I employed different methods to achieve it. I also demonstrate methods used for data preprocessing process in **section II :** Dataset.

II. DATASET

General assessment: The original images in the dataset were quite varied in terms of size where large images' size at around 500x300 and small images' size only around 100x200.

Preprossesing Images:

Increasing the crop size for transformation seems to give error while training because of some small size images. So I have to resize images to fix size initially.
- Resizing images to fixed size (370, 481), which is the mean size of total dataset.

- Realizing the fact that the size of image is quite large, Croping only the center would waste a lot of information and ineffective. Instead, I cropped the corners as well.
For each image, cropping the given image and returning five images using transform.FiveCrop(). The cropping size is 256 which is much larger than 128. This helps to increase the total images for training (from **900 to 4500 images**) and the quantity of dataset is better (more information as crop size 256 > 128)

Original image

Five Crop Images

- Normalizing: I added a **get_mean_std** function in file util.py to compute mean and std of all training images and using that to normalizing the data with transform.Normalize().

- Custom transform: I used *Random Horizontal Flip* to add more randomness to the dataset which helps the network the generalize better. Doing this the image and label pair may show different locations, different flips,…Therefore, I modify **__getitem__ function** in the util.py file by randomly choose the seed and apply it to both input_transforms and target_transforms.

```
seed = np.random.randint(2147483647)  # make a seed with numpy generator

torch.manual_seed(seed)
if self.input_transform is not None:
    image = self.input_transform(image)
torch.manual_seed(seed)
if self.target_transform is not None:
    label = self.target_transform(label)
```

- Finally, save train dataset and validation dataset with Images and Labels locally in seperate directory and push the custom dataset files to content drive.

```
img.save(os.path.join(train_image_root, str(idx) + '.jpg'))
lab.save(os.path.join(train_label_root, str(idx) + '.png'))
```

In the images directory, there are 4500 train images with corresponding labels and 100 validation images.

III. MODEL ARCHITECTURE: UNET

This architecture consists of three sections: the contraction (down part), the bottleneck, and the expansion section (up part).

My main modification from original UNet are as follows:

- The contracting path follows the typical architecture of a convolutional network. It consists o the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit (ReLU) and a 2x2 max pooling operation with stride 2 for downsampling. Noteably, I **employed regularization method** by adding a batch normalization after each 3x3 convolutions from the original set up in modelbaseline.py given.

```
class DoubleConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(DoubleConv, self).__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(in_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(out_channels, out_channels, 3, 1, 1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True),
        )
```

Double Conv Layer.

- At each downsampling step we double the number of feature channels.
Instead of 4 double conv layers as in the original paper (with feature channels output: [64, 128, 256, 512], I **increases the depth of the network** by increasing number of double conv layers to 6 (feature channels output: [32, 64, 128, 256, 512, 1024]).

- Every step in the expansive path consists of an upsampling of the feature map followed by a **Transposed 2D convolution** (“up-convolution”) size 2, stride 2 that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and a double conv layer

- At the final layer a **1x1 convolution** is used to map each 32- component feature vector to 10 classes.

IV. MY ATTEMPTS

1. First attemp

* Model: UNet in the given model baseline
* Dataset: original dataset (1000 images) center crop size 128
* Final mIoU validation result of **0.20**

I first tried to extend the depth of the network to help extract more complex features from the image. For upsampling, I used transposed conv2D instead of Bilinear Interpolation. It does notuse a predefined interpolation method. It has learnable parameters so It allow the network to learn how to up-sample optimally.

2. Changing model.

***** Original dataset
* Model: Increasing the depth of Unet to 4 double ConV layer in contraction and expansion sections [64, 128, 256, 512] with drop out.
* Result: final validation mIoU **0.2833**

As I noticed going to final epochs, the model had no progress so I decreased n_epoch to 100 and employed some transformations on the data. I also increased the learning rate to boost up the training time a little bit.

3. Transforming dataset and tunning hyperpramter. (no change model)

* Transforming train dataset : more train images, transform includes resize((256, 256)), randomcrop size 128, normalizing data by calculating mean and std.
* Hyperprameter: num_epoch = 100, lr = 1e-2,
* Result: final valid mIoU **0.31**

At this point, I tried to generate more images to feed the network. I also increased the depth even further.To increase the speed of the networks train, batch normalization was applied to make weights easier to initialize. Since bias is no longer need, I turned off the bias for each conv layer. It helps to speed up the process as well because it has less parameters

I also found out that although drop-out can help to prevent overfitting while training the network. However, applying dropout increases the training time significantly. Sometimes, it took forever to even start the first epoch. Therefore, I skip the drop-out from the model.

4. Generate more training images and change the model.

***** Gernating custom trainining dataset previously described in Dataset section
* Model: Increasing the depth of Unet to 6 double ConV layer in contraction and expansion sections [36, 64, 128, 256, 512 1024] with no drop out. Batch normalization after each convolution in double conv layer. (Current best model: detaild in model architecture section)
* Result: final validation mIoU **0.52**

```
2021-06-07 02:48:50 || [99/100], train_loss = 4.3356, valid_loss = 5.7314, train_mIoU = 0.8684, val_mIoU = 0.5241, , best mIoU = 0.5241

End of training, elapsed time : 184.0 min 37.20937156677246 sec.
```

5. Keep improving the performance

- I loaded the best model state dict to keep training my current model:

```
model = my_UNet().cuda()
model.load_state_dict(torch.load(f"{filepath}/{experiment}/best_model_state_dict.pt"))
```

As I found the validation mIoU quite good, I tried to change some hyperparameters to booste the performance including changing the learning rate to 1e-3, decrease n_epoch, increasing batch size, changing the optimizer method to rmsprop, adding more custom transforms: colorjitter, random rotation. ..

However, there seemed to be not much progress. I also noticed train loss sometimes increases significantly for continuous epochs and sometimes the model was heavily overfitting with high accuracy for train dataset but low for validation set. So I **employed learning rate scheduler** and **early stopping** the training process if val_mIoU achieved decent result to prevent overfitting.

```
#Using learning rate scheduler
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optim, patience=5, verbose=True)
```

This scheduler reads a metrics quantity and if no improvement is seen for a 'patience' number of epochs, the learning rate is reduced.

* num epochs = 100, lr = 1e-3, batch size = 16
* Result: final validation mIoU **0.56**, final training mIoU **0.87**
* Score on public test data Codalab: mIoU **0.6215**

V. DISCUSSION

Adaptive learning rate with Adam optimizer seems to be the best option for the model.

Increasing the depth of encoder and decorder allows the network to learn higher levels of complex abstraction but also means increasing training time significantly. Therefore, different techniques are required to help decrease the time for training process.

The model with the use of transposed convolution for upsampling seems to make the model learning process more slowly. Combinations of interpolation, subpixel and deconvolutional tec hniques can be further experimented with for exploring fast learnable upsampling.

VI. REFERENCE

- (1): *"U-Net: Convolutional Networks for Biomedical Image Segmentation"* https://arxiv.org/abs/1505.04597
- (2) : *Up sampling with transposed convolution* https://towardsdatascience.com/up-sampling-with-transposed-convolution-9ae4f2df52d0