Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs

Deep Learning Project Update 2

Spring 2023

Team

Likhith Sai Chaitanya Vadlapatla

Ajay Kumar Ganipineni

Dataset:

Data Collection:

Images are Captured with mobile handset across the university of New Haven campus.

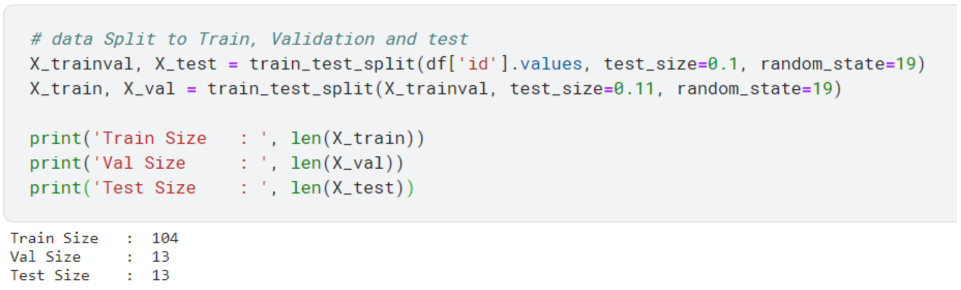
Data Pre-Processing:

Our captured images are of size between 5MB to 12MB and of different mobile handset format, we pre-processed the images to reduce image size and to desired format. We annotated with 5 labeled classes (building, car, traffic sign, chair, person). We used MATLAB to annotate the images.

Step:1

Then we divided the datasets into three parts:

training dataset (80% of the data), validation dataset (10% of the data), testing dataset (remaining 10% of the data).



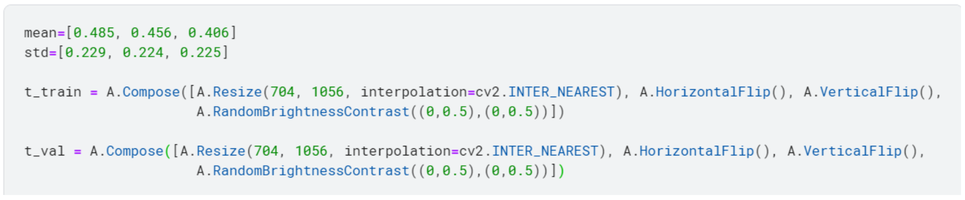
```
# data Split to Train, Validation and test
X_trainval, X_test = train_test_split(df['id'].values, test_size=0.1, random_state=19)
X_train, X_val = train_test_split(X_trainval, test_size=0.11, random_state=19)

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

```
Train Size   :  104
Val Size     :  13
Test Size    :  13
```

Step 2:

Data Augmentation:

In the real world, it is very difficult to collect huge amounts of data. Hence if we flip the data horizontally and vertically. Hence as we are increasing the size of the dataset by adding similar types of data, it reduces overfitting.



```
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.HorizontalFlip(), A.VerticalFlip(),
                     A.RandomBrightnessContrast((0,0.5),(0,0.5))])

t_val = A.Compose([A.Resize(704, 1056, interpolation=cv2.INTER_NEAREST), A.HorizontalFlip(), A.VerticalFlip(),
                   A.RandomBrightnessContrast((0,0.5),(0,0.5))])
```

Methodology:

- We are using Unet -- mobilenet_v2 for transfer learning.
- We replaced the output channels to 5 to match our dataset

```
model = smp.Unet('mobilenet_v2', encoder_weights='imagenet', classes=5, activation=None,
encoder_depth=5, decoder_channels=[256, 128, 64, 32, 16])
```

- **Architecture of U-Net:**

  U-Net is U shaped architecture which is symmetric. It consists of two major parts: the left path is contracting path, which is constituted by general convolution process whereas the right path is expansive path, which is constituted by transposed 2d convolution layers.

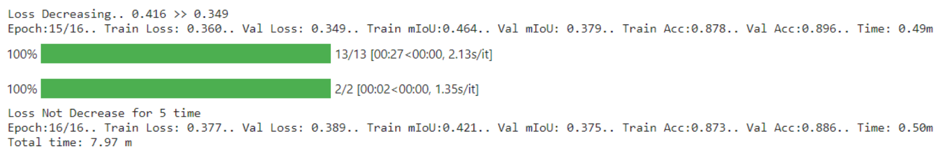
- We are using a batch size of 8. As we are using 104 training images. In each mini batch we will have 8 images.
- **Hyperparameters Used:**

  Learning rate = 1e-3

  epoch = 16

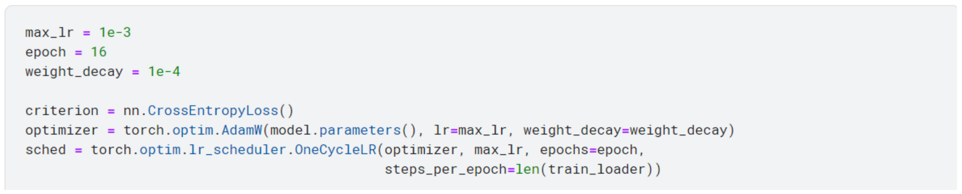
  weight_decay = 1e-4

- We have used cross entropy loss function which resulted in loss of 0.389.



```
Loss Decreasing.. 0.416 >> 0.349
Epoch:15/16.. Train Loss: 0.360.. Val Loss: 0.349.. Train mIoU:0.464.. Val mIoU: 0.379.. Train Acc:0.878.. Val Acc:0.896.. Time: 0.49m
100% 13/13 [00:27<00:00, 2.13s/it]

100% 2/2 [00:02<00:00, 1.35s/it]
Loss Not Decrease for 5 time
Epoch:16/16.. Train Loss: 0.377.. Val Loss: 0.389.. Train mIoU:0.421.. Val mIoU: 0.375.. Train Acc:0.873.. Val Acc:0.886.. Time: 0.50m
Total time: 7.97 m
```

- We used Adam optimizer as it converges very fast.

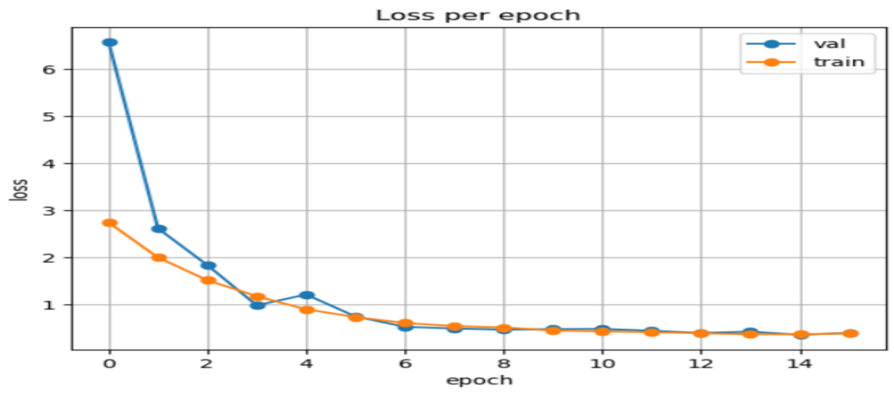


```
max_lr = 1e-3
epoch = 16
weight_decay = 1e-4

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))
```

Evaluation:

Loss:

As we can see from the image, validation and train loss have decreased with the increase in number of epochs.

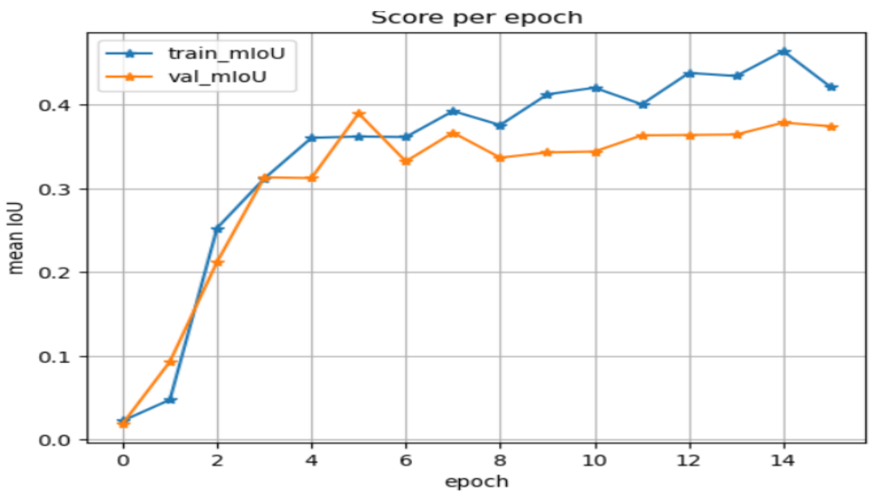


MIoU:

Mean IOU calculates the ratio of area overlapped by two bounding boxes. The two observations are real and predicted observations.

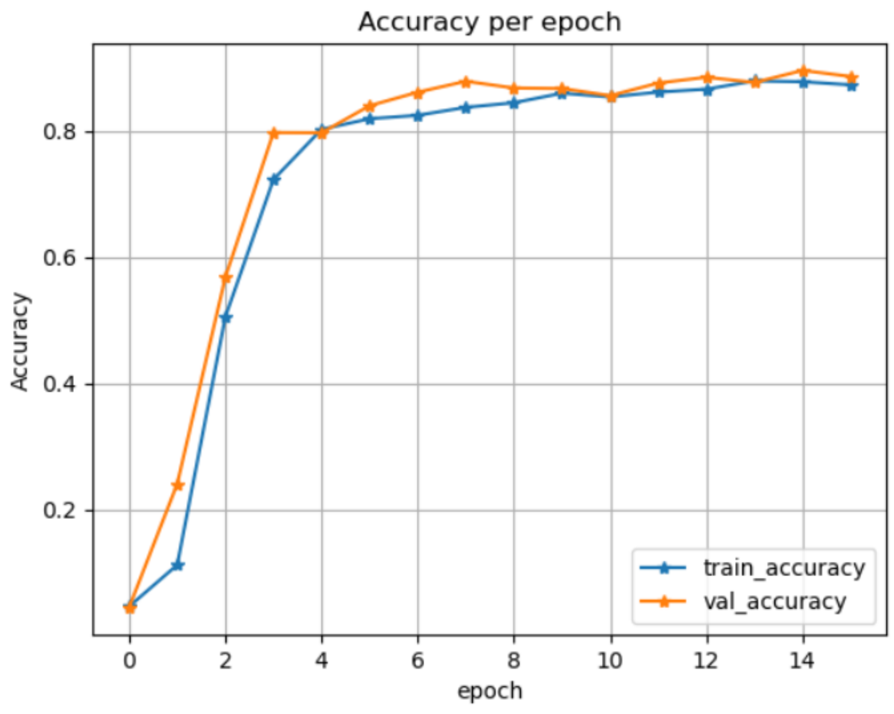
$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}.$$

If the prediction is completely correct it means IoU will be 1.

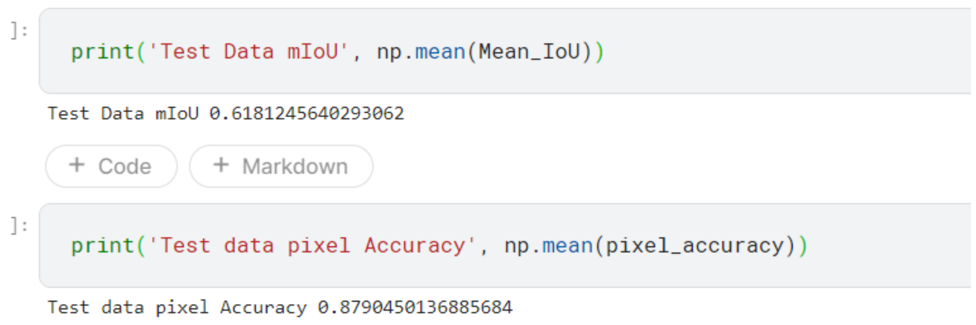


Accuracy:

As we can see from the image that, both train and validation accuracy are increasing with increase in number of epochs.



Pixel Accuracies and MIoU for Test Data:


```
]:  print('Test Data mIoU', np.mean(Mean_IoU))
```

```
Test Data mIoU 0.6181245640293062
```

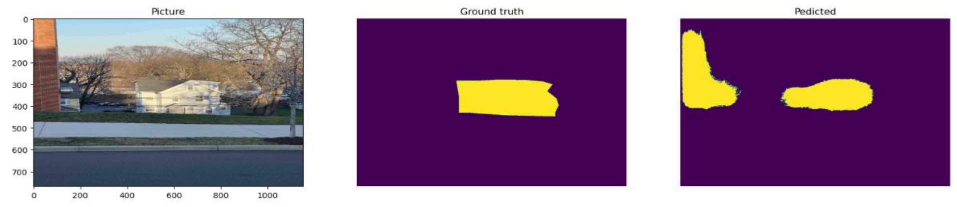
```
]:  print('Test data pixel Accuracy', np.mean(pixel_accuracy))
```

```
Test data pixel Accuracy 0.8790450136885684
```

Predictions on test data:

Sample 1

```python
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('Pedicted')
ax3.set_axis_off()
```



Sample 2

```python
image2, mask2 = test_set[8]
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('predicted')
ax3.set_axis_off()
```