CV Homework 4 Report

— 106062202 陳騰鴻

— Dec 28, 2019

Q1: Binary Classification

Q1-1

```
Training complete in 4m 15s
Best val Acc: 0.830000
```

Q1-2

```
Training complete in 5m 24s
Best val Acc: 0.875000
```

Q1-3

```
Training complete in 5m 26s
Best val Acc: 0.717500
```

Q1-4

Comparisons

Model	Pretrained	Backprop Parameters	Performance
Feature Extractor	Yes	1 Linear Classifier	Second
Fine-tuned	Yes	All Layers + 1 Linear Classifier	**Best**
AlexNet	No	All layers	Last

Why Feature Extractor Performs Better than AlexNet:

The feature extractor benefits from the pretrained parameters, which are trained on an extensive amount of images.
Due to the effect of transfer learning, the feature extractor yields better performance.

Why Fine-tuned Model Performs Better than Feature Extractor:

Rather freezing the pretrained parameters, the fine-tuned model performs backpropagation on all parameters, enabling to fit the training dataset better.
Thus, the fine-tuned model yields better results.

Q1-5

```
Training complete in 3m 33s
Best val Acc: 0.782500
```

> Validation accuracy worsened

Q1-6

```
Training complete in 4m 45s
Best val Acc: 0.892500
```

> Validation accuracy improved

Q1-7

Method

The resizing interval of `transforms.RandomResizedCrop` is tweaked to contain the whole face for most pictures.

```
'train': transforms.Compose([
    transforms.RandomResizedCrop(224, scale=(0.75, 1.0)),
    transforms.RandomHorizontalFlip(),
    transforms.ToTensor(),
    transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
]),
```

Why The Accuracy Worsened for The Feature Extractor

Due to the randomly cropping transformation, the model learned features that are unrelated to the face, which accidentally increased the accuracy.
Therefore, using proper data, the actual transfer learning effect is not as dramatic as Q1-1 demonstrates.

Why The Accuracy Improved for The Fine-tuned Model

When the dataset provides consistent features, i.e. the face, the fine-tuned model can tune its parameters more precisely, rather than trying to generalize upon badly cropped pictures.

Q1-8

Implementation

1. Data Augmentation: `AutoAugment` and `Cutout` are imported from repository 4uiiurz1/pytorch-auto-augment (https://github.com/4uiiurz1/pytorch-auto-augment).

```
'train': transforms.Compose([
        transforms.ColorJitter(brightness=.04, contrast=.04,
        saturation=.04, hue=.04),
        transforms.RandomResizedCrop(224, scale=(0.75, 1.0)),
        transforms.RandomHorizontalFlip(),
        AutoAugment(),
        Cutout(),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        transforms.Normalize([0.485, 0.456, 0.406],
                             [0.229, 0.224, 0.225])
]),
```

2. Model: EfficientNet-B2, loaded from [rwightman/gen-efficientnet-pytorch](https://github.com/rwightman/gen-efficientnet-pytorch) (https://github.com/rwightman/gen-efficientnet-pytorch).
   The model has a one-layer classifier with output feature = 1000, so `nn.Linear(1000, 2)` is added.

```
model_ft = torch.hub.load('rwightman/gen-efficientnet-pytorch',
                          'efficientnet_b2', pretrained=True)
model_ft.classifier = nn.Sequential(model_ft.classifier,
                                    nn.Linear(1000, 2))
```

3. Optimizer: Adam with learning rate = 0.001.

```
optimizer_ft = optim.Adam(model_ft.parameters(), lr=0.001)
```

Q1-9

Result

```
Training complete in 14m 12s
Best val Acc: 0.907500
```

How I Chose The Modifications

1. `transforms.ColorJitter` is a commonly used transformation.
2. Auto-augmentation is [one of the most implemented](https://paperswithcode.com/task/image-augmentation) (https://paperswithcode.com/task/image-augmentation) image augmentation techniques. Augmentation for CIFAR_10 is used in hope of increasing the performance.
3. EfficientNet is chosen because of its [great performance](https://paperswithcode.com/sota/image-classification-on-imagenet) (https://paperswithcode.com/sota/image-classification-on-imagenet) and relatively small number of parameters.
   Despite EfficientNet-B7 having similar number of parameters as AlexNet does, it takes way too long to train.
   The final decision was to use EfficientNet-B2, only 9.2 M parameters (0.15x of AlexNet).
4. Optimizer was changed to Adam in attempt to increase the speed of convergence.

Q2: Semantic Segmentation

Q2-1

Before Removing Skip Connections

```
The highest mIOU is 0.4295 and is achieved at epoch-41
The highest pixel accuracy  is 0.8523 and is achieved at epoch-41
```

After Removing Skip Connections

```
The highest mIOU is 0.4055 and is achieved at epoch-20
The highest pixel accuracy  is 0.8384 and is achieved at epoch-17
```

Q2-2

Implementation

I deleted the addition of `x4` and `x3` in `FCN8s.forward`, which are the two skip connections in the neural network.

```
def forward(self, x):
    output = self.pretrained_net(x)
    x5 = output['x5']

    score = self.relu(self.deconv1(x5))
    score = self.bn1(score)
    score = self.relu(self.deconv2(score))
    score = self.bn2(score)
    score = self.bn3(self.relu(self.deconv3(score)))
    score = self.bn4(self.relu(self.deconv4(score)))
    score = self.bn5(self.relu(self.deconv5(score)))
    score = self.classifier(score)

    return score
```

Are skip connections quantitatively beneficial?

According to the experimental results, the neural network with skip connections performs better in both measurements by 2%.
Therefore, skip connections are quantitatively beneficial.

Why adding skip connections improves performance

Deep features can be obtained when data is passed through more layers.
However, spacial location information is also lost due to convolution.(Paraphrasing a quote from this article (https://towardsdatascience.com/review-fcn-semantic-segmentation-eb8c9b50d2d1).)
Hence, by adding skip connections, the spacial location information (Extracted when the network is still shallow) is preserved and improves the performance.

Q2-3

Results

```
The highest mIOU is 0.8261 and is achieved at epoch-20
The highest pixel accuracy  is 0.9629 and is achieved at epoch-20
```

Implementation

- Number of segmentation classes is changed to 3.

```
num_class = 3
```

- Labels of each class can be found in `save_result_comparison` .

```
if output_np[i,j] == 0:  # Sky
    im_seg_RGB[i,j,:] = [128, 128, 128]
elif output_np[i,j] == 1:  # Building
    im_seg_RGB[i,j,:] = [128, 0, 0]
elif output_np[i,j] == 2:  # Pole
    im_seg_RGB[i,j,:] = [192, 192, 128]
elif output_np[i,j] == 3:  # Road
    im_seg_RGB[i,j,:] = [128, 64, 128]
elif output_np[i,j] == 4:  # Pavement
    im_seg_RGB[i,j,:] = [0, 0, 192]
elif output_np[i,j] == 5:  # Tree
    im_seg_RGB[i,j,:] = [128, 128, 0]
elif output_np[i,j] == 6:  # Sign Symbol
    im_seg_RGB[i,j,:] = [192, 128, 128]
elif output_np[i,j] == 7:  # Fence
    im_seg_RGB[i,j,:] = [64, 64, 128]
elif output_np[i,j] == 8:  # Car
    im_seg_RGB[i,j,:] = [64, 0, 128]
elif output_np[i,j] == 9:  # Pedestrian
    im_seg_RGB[i,j,:] = [64, 64, 0]
elif output_np[i,j] == 10:  # Bicyclist
    im_seg_RGB[i,j,:] = [0, 128, 192]
```

- Labels are changed according to the specs, using boolean indexing.

```
label = np.array(label_image)

# Class 1
mask_class1 = (label > 0) & (label < 8) | (label > 10)
label[mask_class1] = 1

# Class 2
mask_class2 = (label > 7) & (label < 11)
label[mask_class2] = 2
```

Q2-4

The Comparison Model

> Q2-3. Please try to **FURTHER** reduce the number of classes from 11 to 3 ...

Thus, our comparison model should be the one **without** skip connections.

Why The Mean IoU And The Accuracy Increased Dramatically

- With less classes, the ground truth segmentations would have simpler contours, thus making it an easier task for a model to learn.
- The following are the IoUs at maximum performance:

Classifier	Result
11-class	`pix_acc: 0.8306, meanIoU: 0.3932,` `IoUs: [0.8899 0.7452 0. 0.9159 0.7039 0.738 0. 0.0034 0.2936 0.0353 0.]`
3-class	`pix_acc: 0.9629, meanIoU: 0.8261, IoUs: [0.9049 0.9559 0.6175]`

  As we can see, the 11-class classifier fails to detect class 10, which greatly reduces the mean IoU.
- As all the other factors (hyperparameters, data augmentation, training epochs ...) are controlled, the performance increases.