CISC3024 Pattern Recognition Project

by Zhang Huakang D-B9-2760-6

> The notebook file of this project has been published on kaggle.

Member Name	Contribution Percentage
ZHANG HUAKANG	100%

1. Data Loading

In this part, the image will be loaded from disk with its corresponding labels. After loading images and lables, a customized class `SatalliteDataset` inherited from `torch.utils.data.Dataset` is used to stored this data. When `__getitem__()` function is called, processed images and labels are returned.

```
# Image processing for training data
train_tran = transforms.Compose([
                    transforms.RandomRotation(5),
                    transforms.RandomHorizontalFlip(0.5),
                    transforms.RandomCrop(64, padding=2),
                    transforms.ToTensor(),
                ])
# Image processing for testing and val. data
tran = transforms.Compose([
                    transforms.ToTensor(),
                ])
```

```
# SatalliteDataset
class SatalliteDataset(Dataset):
    def __init__(self,images,labels, is_train):
        self.images_list=images
        self.labels_list=labels
        self.is_train=is_train

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

    def __getitem__(self,idx):
        # Image Processing
        if self.is_train:
            return train_tran(self.images_list[idx]).to(device),
self.labels_list[idx]
        return tran(self.images_list[idx]).to(device),
self.labels_list[idx]
```

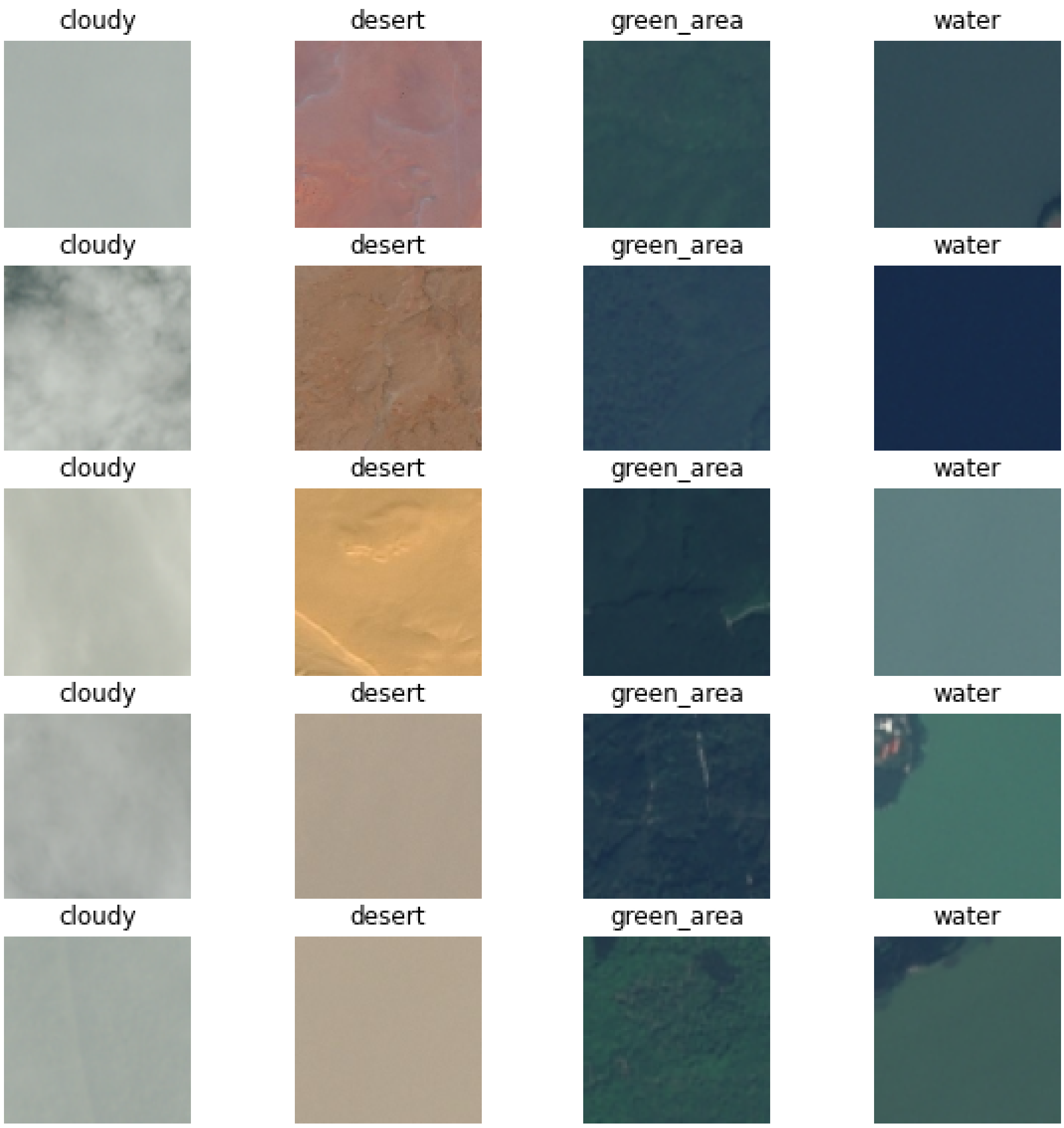
Then, `train`, `test` and `val` data will be encapsulated in `torch.utils.data.Dataloader` object to make it easier to get data when training the model.

```
train_iterator = data.DataLoader(SatalliteDataset(X_train,y_train, True),
batch_size=BATCH_SIZE)
```

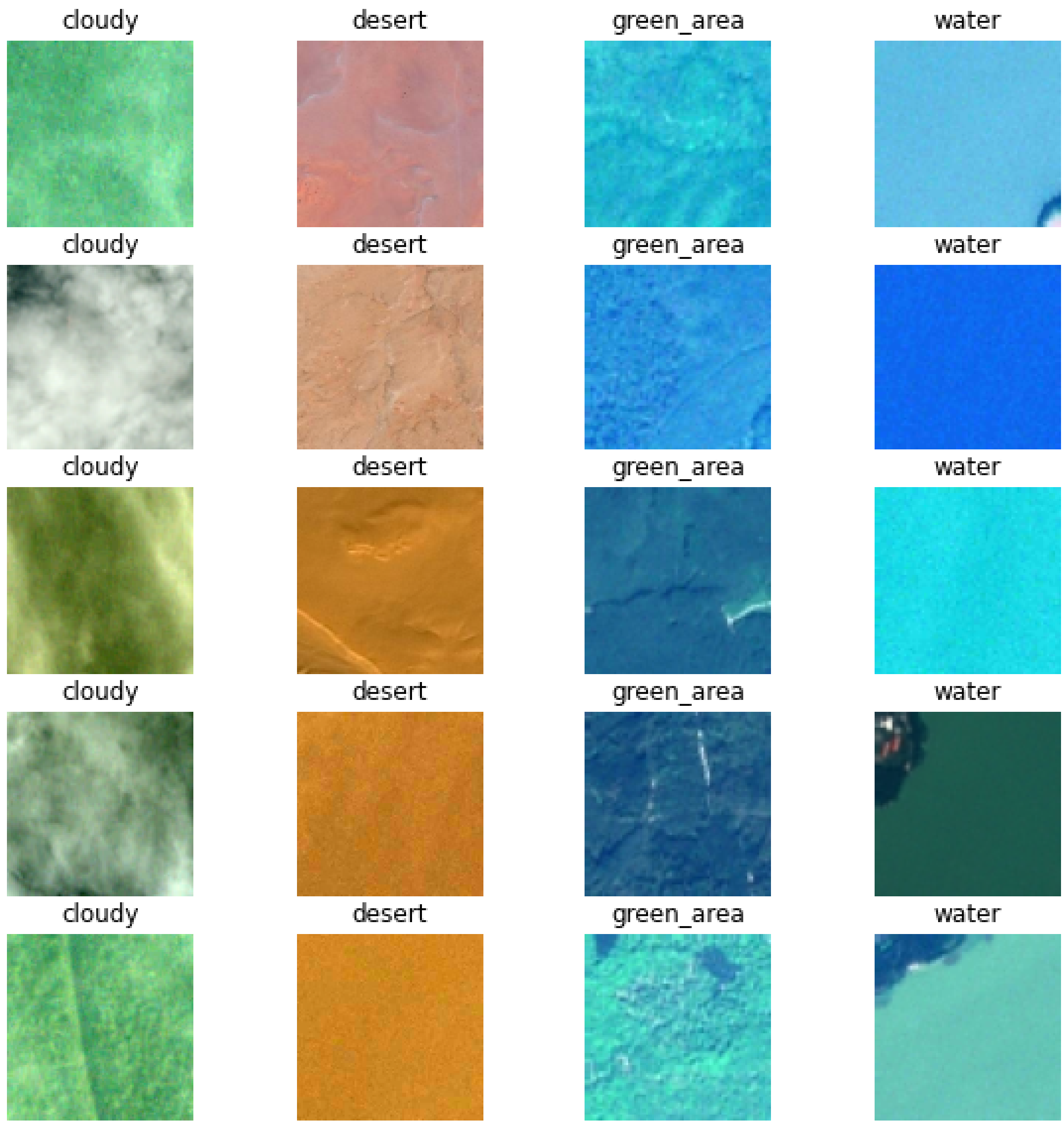
Example Image

Here is some example of dataset.

Unprocessed:

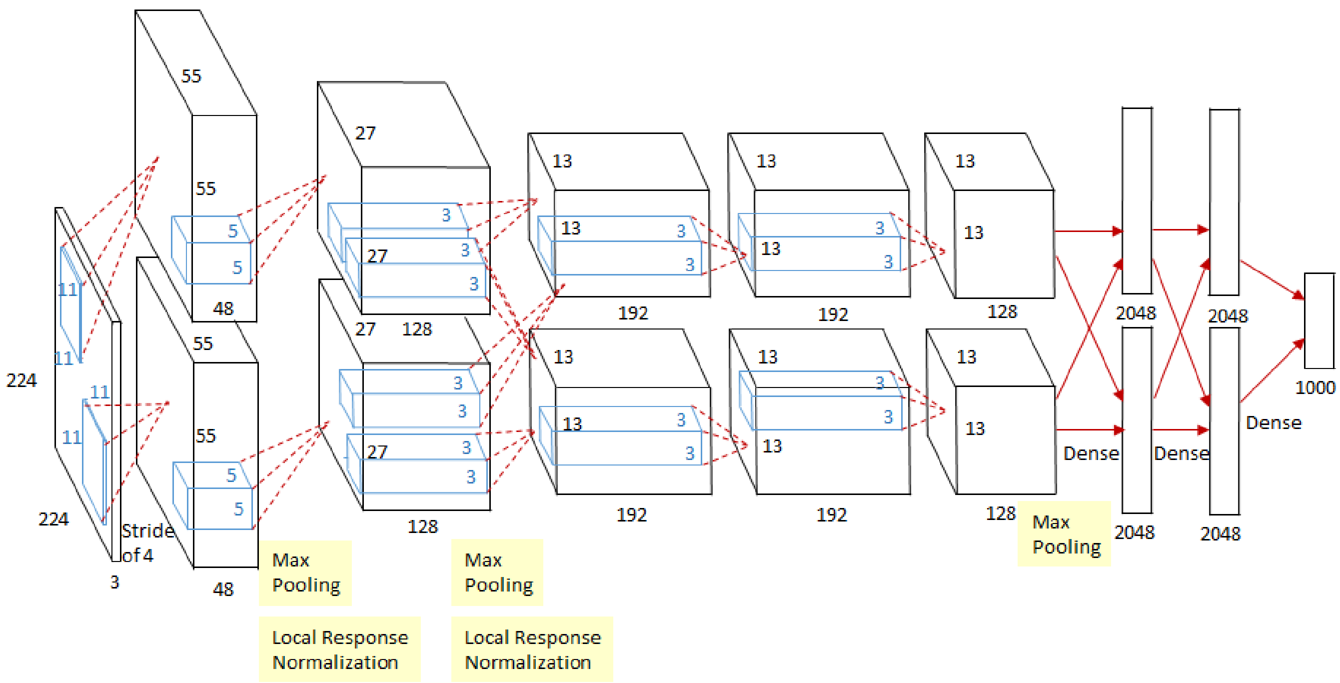


Normalization:



Model Design

AlexNet



code version:

```
class AlexNet(nn.Module):
    def __init__(self, output_dim):
        super().__init__()
        self.features = nn.Sequential(
            nn.Conv2d(3, 64, 3, 2, 1),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True),
            nn.Conv2d(64, 192, 3, padding=1),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True),
            nn.Conv2d(192, 384, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(384, 256, 3, padding=1),
            nn.ReLU(inplace=True),
            nn.Conv2d(256, 256, 3, padding=1),
            nn.MaxPool2d(2),
            nn.ReLU(inplace=True)
        )
        self.classifier = nn.Sequential(
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Dropout(0.5),
            nn.Linear(4096, 4096),
            nn.ReLU(inplace=True),
            nn.Linear(4096, output_dim),
        )
    def forward(self, x):
        x = self.features(x)
        h = x.view(x.shape[0], -1)
        x = self.classifier(h)
        return x, h
```

Model Training

Find the best learning rate

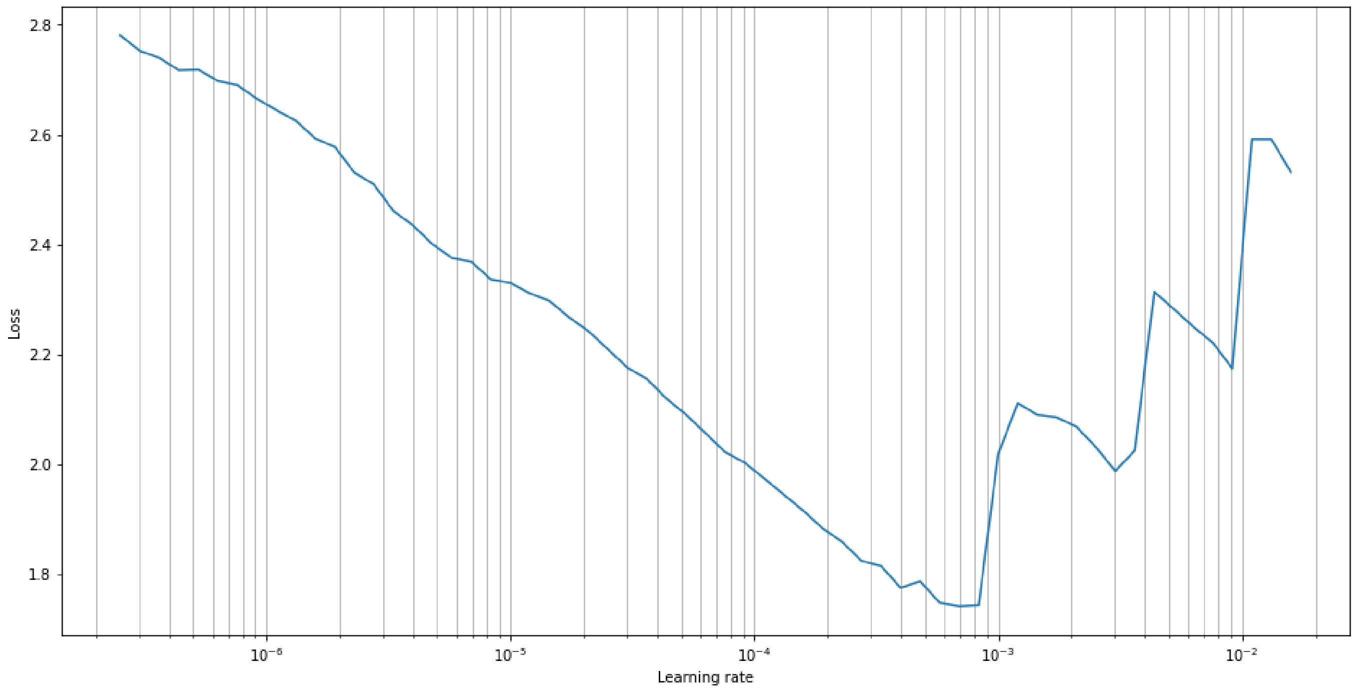
In this part, I build a class `LRFinder` to find the best learning rate. In the function `range_test()`,

```
def range_test(self, iterator, end_lr=10, num_iter=100,
                   smooth_f=0.05, diverge_th=5):
        lrs = []
        losses = []
        best_loss = float('inf')
        lr_scheduler = ExponentialLR(self.optimizer, end_lr, num_iter)
        iterator = IteratorWrapper(iterator)
        for iteration in range(num_iter):
            loss = self._train_batch(iterator)
```

```
            lrs.append(lr_scheduler.get_last_lr()[0])
            # update lr
            lr_scheduler.step()
            if iteration > 0:
                loss = smooth_f * loss + (1 - smooth_f) * losses[-1]
            if loss < best_loss:
                best_loss = loss
            losses.append(loss)
            if loss > diverge_th * best_loss:
                print("Stopping early, the loss has diverged")
                break
        # reset model to initial parameters
        model.load_state_dict(torch.load('init_params.pt'))

        return lrs, losses
```

This is the result:



From this image, we can see that when `LR=10e-4`, loss decreas fastest. SO we choose `10e-4`as the learning rate.

Training the model

This is the code of training:

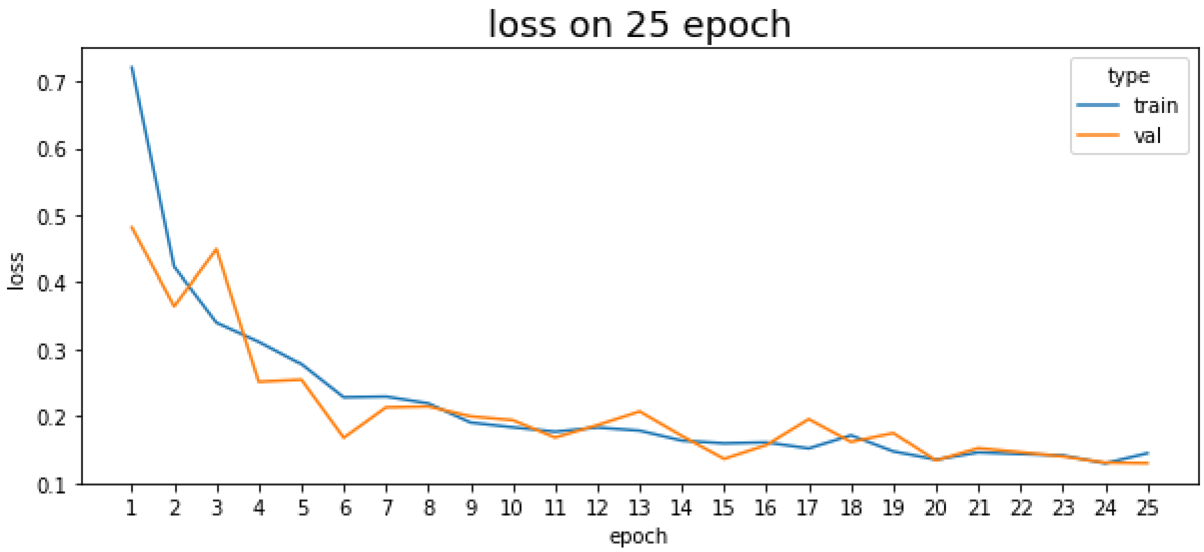
```
def train(model, iterator, optimizer, criterion, device):
    epoch_loss = 0
    epoch_acc = 0
    model.train()
    for (x, y) in tqdm(iterator, desc="Training", leave=False):
        x = x.to(device)
        y = y.to(device)
```

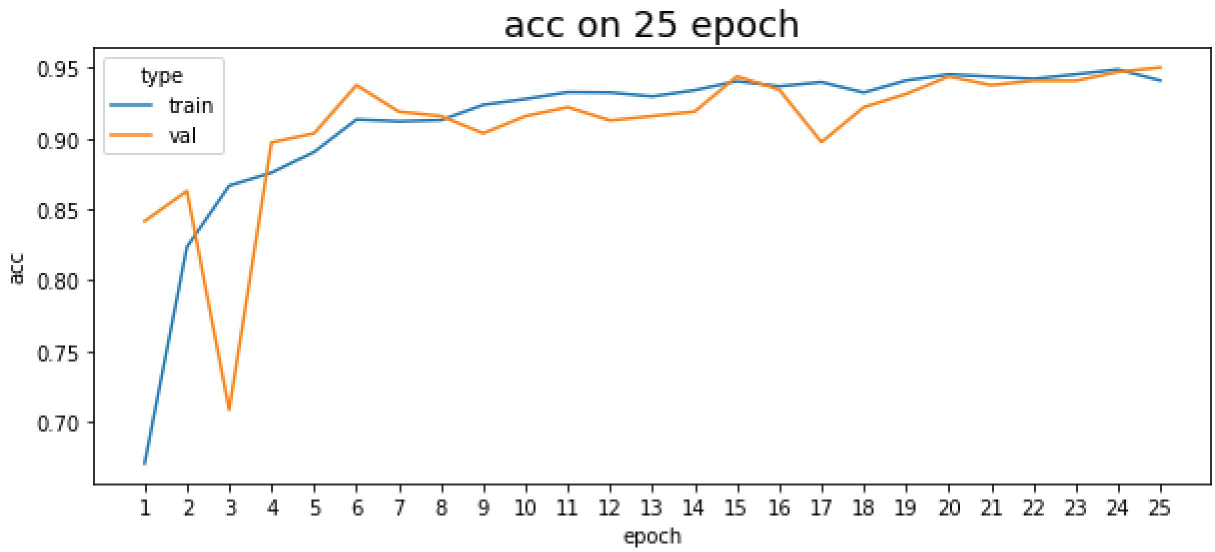
```
        optimizer.zero_grad()
        y_pred, _ = model(x)
        loss = criterion(y_pred, y)
        acc = calculate_accuracy(y_pred, y)
        prec,recall = precision_recall(y_pred, y, average='macro',
num_classes=4)
        loss.backward()
        optimizer.step()
        epoch_loss += loss.item()
        epoch_acc += acc.item()
    return epoch_loss / len(iterator), epoch_acc / len(iterator),
prec.to('cpu'), recall.to('cpu')
```

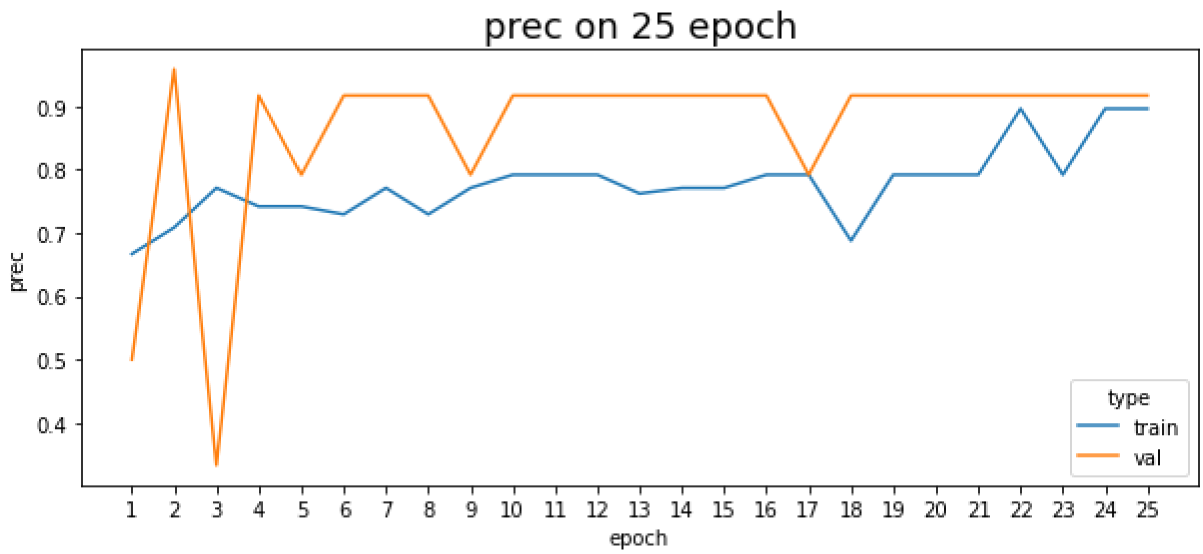
Model Performance Evaluation

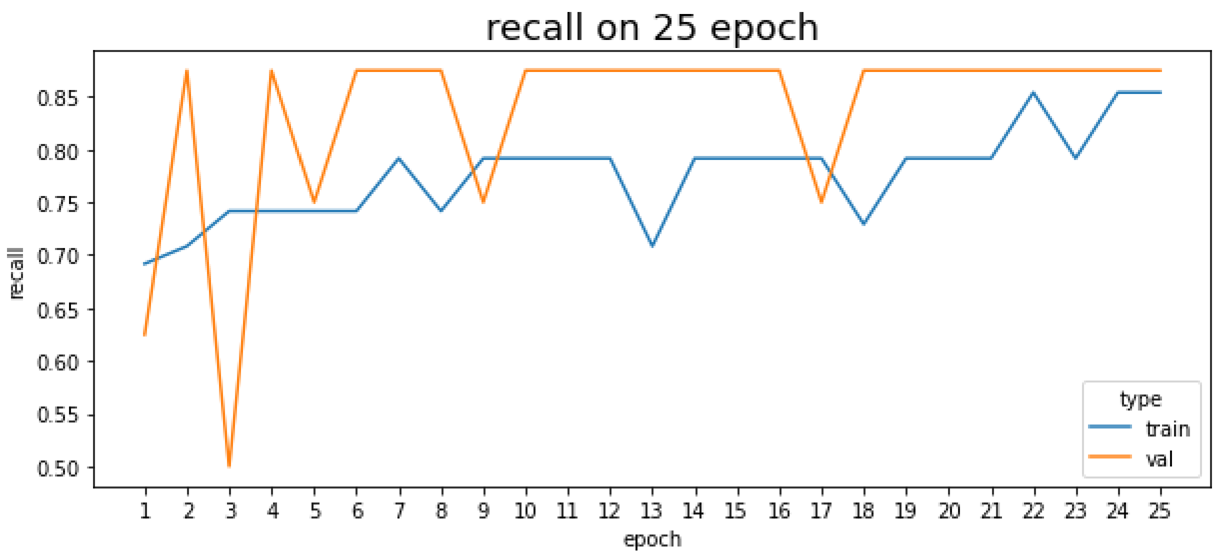
After 25 epoches of training,

```
Epoch: 25 | Epoch Time: 0m 6s
    Train Loss: 0.144 | Train Acc: 94.11% | Train Prec: 89.58% | Train
Recall: 85.42%
     Val. Loss: 0.129 |  Val. Acc: 95.03% |  Val. Prec: 91.67% |  Val.
Recall: 87.50%
```

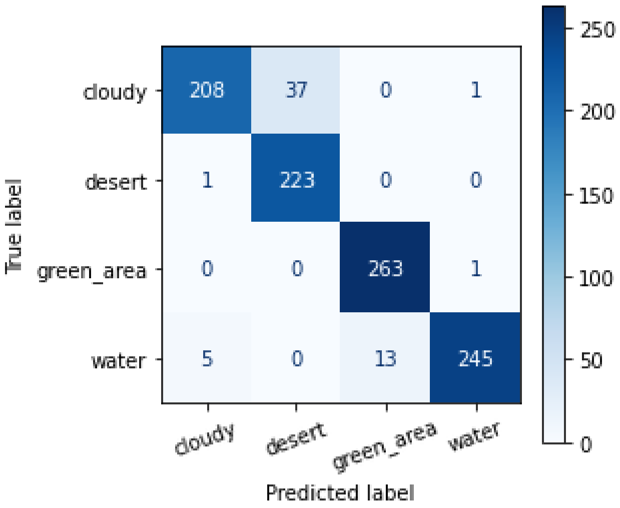








And for test data, the prediction result is shown below.



True label
Predicted label
cloudy
desert
green_area
water
208
37
0
1
1
223
0
0
0
0
263
1
5
0
13
245
250
200
150
100
50
0