License Plate Recognition

阮柏愷 bkruan.ee11@nycu.edu.tw

黃柏綸 kevin503.ee12@nycu.edu.tw

Special Thanks to Yi-Lun Wu 🤗

Outline

- Problem Description
- Dataset
- Tasks
    - Seting up Environment
    - Build Dataset
    - Define Model
    - Save Model

Problem Description

- Input: An image of a License plate.
- Output: A string of length 7.
- We can make some assumptions:
    - All license plates are centered in the images.
    - The number of characters in a license plate is fixed to 7.
    - Only characters `0~9` and `A~Z` are legal.

Dataset

- Our dataset was taken from this work with a few modifications.
- Training set:
    - It contains 8 different folders with a total of 8762 images.
    - `trainValNew.csv` contains all the image paths of the training set.
- Testing set
    - It contains 4 different folders with a total of 299 images.
    - `trainValNew.csv` contains all the image paths of the testing set.
- The CSV format is as follows:

```
track_id , image_path             , lp      , train
s01_l01_1, s01_l01_unique/1_1.png, 7C24698, 1
s01_l01_3, s01_l01_unique/3_1.png, 5AR2347, 1
...
```

Source Code

- SSH into an arbitrarily server.
- Clone the source code from Github. Note, **if the source code has been cloned in the previous lab, there is no need to clone it again this time**.

```
$ git clone https://github.com/Justin900429/hcc-ml
```

- Make sure that your working directory is `hcc-ml/lab2`

```
$ cd hcc-ml/lab2
```

Or you can open the folder `hcc-ml/lab2` in VS Code.

Create Virtual Environment

- Create virtual environment for Lab2.

  ```
  $ python3 -m venv {name_of_your_venv}
  ```

  For example:

  ```
  $ python3 -m venv lab2env
  ```

How to Use?

- Activate your virutal environment.

  ```
  $ source [name_of_your_venv]/bin/activate
  ```

  For example:

  ```
  $ source lab2env/bin/activate
  (lab2env) $
  ```

  Once activated, the environment name will be displayed in your terminal prompt.
- Install all python packages.

  ```
  (lab2env) $ pip install -r requirements.txt
  ```

- To use Nvidia GPU in PyTorch, you need to install the Nvidia driver and the CUDA toolkit. These packages have already been installed on all servers.

Data Preprocessing

- Download the dataset from the Google Drive. link
  Alternatively, you can run `download.py` script to download the dataset from the terminal.

```
$ python download.py
```

- Unzip the dataset.

```
$ unzip dataset.zip
```

Data Preprocessing

- The structure of the `./dataset` directory should look like this:

```
dataset
├── train
│     ├── s01_l01_unique
│     ├── s01_l02_unique
│     ├── ...
│     └── trainValNew.csv
└── val
      ├── crop_m1_unique
      ├── crop_m2_unique
      ├── ...
      └── trainValNew.csv
```

Customize dataset in PyTorch

- The base dataset is defined in `./dataset.py`.

```
class LicensePlateDataset(torch.utils.data.Dataset):
    def __init__(self, csv_path, transform=None):
        self.transform = transform
        csv = pd.read_csv(csv_path)
        root = os.path.dirname(csv_path)
        self.paths = csv['image_path'].apply(
            lambda p: os.path.join(root, p)).values
        self.labels = csv['lp'].apply(
            lambda lp: [CODES_TO_INT[c] for c in lp]).values

    def __len__(self):
        ...
    def __getitem__(self, idx):
        ...
```

Customize dataset in PyTorch

- The base dataset is defined in `./dataset.py`.

```
class LicensePlateDataset(torch.utils.data.Dataset):
    def __init__(self, csv_path, transform=None):
        ...

    def __len__(self):
        # return the size of the dataset
        return len(self.paths)

    def __getitem__(self, idx):
        # return i-th element in the dataset
        image = Image.open(self.paths[idx]) # load on demand
        if self.transform is not None:
            image = self.transform(image)
        return image, torch.tensor(self.labels[idx])
```

Training Set

- Derive a training set from the `LicensePlateDataset`:

```
class TrainDataset(LicensePlateDataset):
    def __init__(self,
                 csv_path='./dataset/train/trainValNew.csv'):
        transform = transforms.Compose([
            transforms.ColorJitter(0.2, 0.2, 0.2, 0.2),
            transforms.RandomRotation(3),
            transforms.RandomResizedCrop(
                (HEIGHT, WIDTH),
                scale=(0.7, 1.0),
                ratio=[2., 2.8]),
            transforms.ToTensor()
        ])
        super().__init__(csv_path, transform)
```

Checkpoint 1 - Testing Set

- To complete the testing set, you need to add a resize transformation to the `transform` object in `./dataset.py`

```
'''
Checkpoint 1: Resize image to (HEIGHT, WIDTH)
Hint: Add resize transformation to the begining of transfrom
'''
transform = transforms.Compose([
    ???
    transforms.ToTensor()
])
```

- `HEIGHT` and `WIDTH` are global variables in `dataset.py`, so you don't need to define them again.
- You can find the `Resize` operation in the documentation.

Checkpoint 1

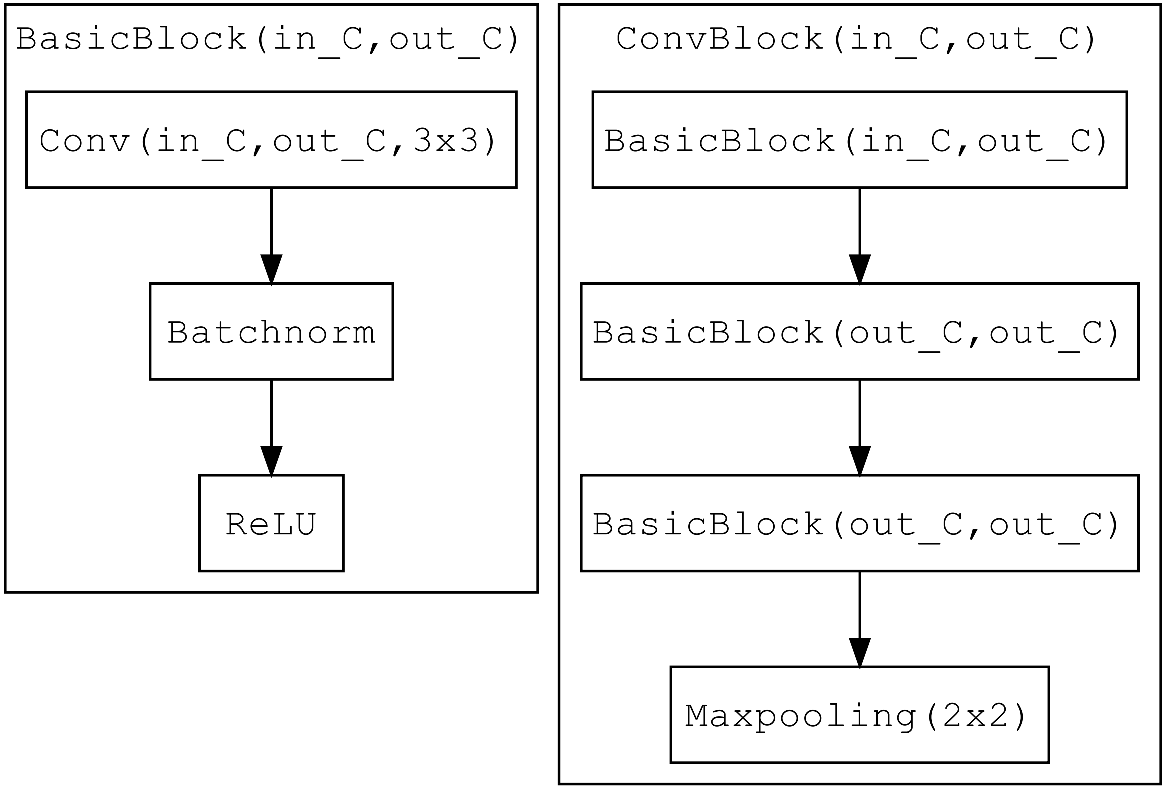
- Execute `dataset.py` to test your modification.

```
$ python dataset.py
```

- Show the resolution of the dumped image `val.jpg`.

```
$ file val.jpg
```

Schematic of 2 Basic Modules



Step 1. Define `BasicBlock` in PyTorch

- Here is a standard implementation of `BasicBlock` in PyTorch:

```python
class BasicBlock(nn.Module):
   def __init__(self, in_channels, out_channels):
       super().__init__()
       self.conv = nn.Conv2d(
           in_channels, out_channels, kernel_size=3, padding=1)
       self.bn = nn.BatchNorm2d(out_channels)
       self.relu = nn.ReLU()

   def forward(self, x):
       x = self.conv(x)
       x = self.bn(x)
       x = self.relu(x)
       return x
```

Step 1. Alternative Approach for `BasicBlock`

- Alternatively, you can rewrite `BasicBlock` using `nn.Sequential` like this:

```
class BasicBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.main = nn.Sequential(
            nn.Conv2d(
                in_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU())

    def forward(self, x):
        return self.main(x)
```

Step 2. Define `ConvBlock` in PyTorch

- Here is a standard implementation of `ConvBlock` in PyTorch:

```python
class ConvBlock(nn.Module):
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.conv1 = BasicBlock(in_channels, out_channels)
        self.conv2 = BasicBlock(out_channels, out_channels)
        self.conv3 = BasicBlock(out_channels, out_channels)
        self.maxpool = nn.MaxPool2d((2, 2))

    def forward(self, x):
        x = self.conv1(x)
        x = self.conv2(x)
        x = self.conv3(x)
        x = self.maxpool(x)
        return x
```

Checkpoint 2

- You have to rewrite `ConvBlock` using `nn.Sequential`.

```python
class ConvBlock(nn.Module):
    '''
    Checkpoint 2:
        Use `nn.Sequential` to rewrite `ConvBlock`.
    '''
    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.main = ???

    def forward(self, x):
        return ???
```

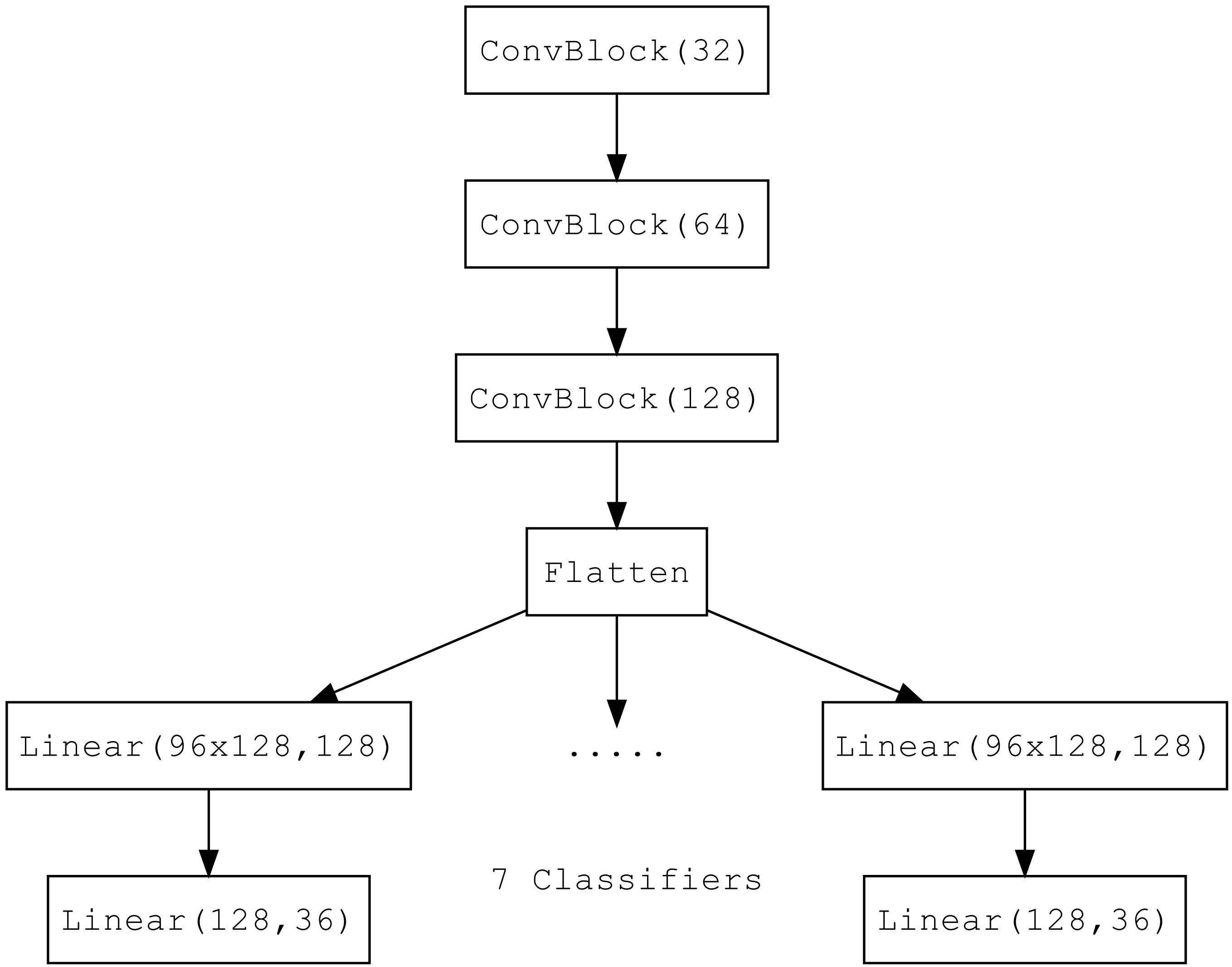
Checkpoint 2

- To test the modules, please run the following command in the terminal after removing the old `ConvBlock`:

```
$ python model.py
```

- Show the output in the terminal.

Schematic of Full Model



Define Full Model

```
class LPRModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.encoder = nn.Sequential(
            ConvBlock(3, 32),          # [B, 32, H / 2, W / 2]
            ConvBlock(32, 64),         # [B, 64, H / 4, W / 4]
            ConvBlock(64, 128),        # [B, 128, H / 8, W / 8]
            nn.Flatten(start_dim=1))  # [B, 128 x H / 8 x W / 8]
        self.classifiers = nn.ModuleList()
        for _ in range(7):
            classifier = nn.Sequential(
                nn.Linear(HEIGHT // 8 * WIDTH // 8 * 128, 128),
                nn.ReLU(),
                nn.Linear(128, 36))
            self.classifiers.append(classifier)
    ...
```

Checkpoint 3

Please complete the `forward` function of the full model as follows:

```python
class LPRModel(nn.Module):
    ...
    def forward(self, x):
        x = self.encoder(x)
        outputs = []
        for classifier in self.classifiers:
            output = classifier(x)           # [B, 36]
            output = output.unsqueeze(1)     # [B, 1, 36]
            outputs.append(output)
        '''
        Checkpoint 3:
            Concatenate `outputs` along the axis 1.
            The shape of the result must be [B, 7, 36].
        '''
        # output = torch.cat(????)
        return output
```

Checkpoint 3

- Download `pretrain.zip` from Google Drive. You can find the link here.

> If you have already used download.py to download the dataset, you will not need to download pretrain.zip as it has already been downloaded.

- To unzip pretrain.zip, use the following command:

```
$ unzip pretrain.zip
```

- Then, execute the following command:

```
$ python detect.py \
    --checkpoint ./pretrain/ckpt100.pt \
    --image ./example/9B52145.png
```

- Finally, show the predictions by showing the following message:

```
Loading image from ./example/9B52145.png
weight has been loaded
predictions: 9B52145
```

Training Loop

```
Declare Dataset
Declare Model
Declare Optimizer
For epoch in 1..epochs:
    For batch in Dataset:
        Calculate loss by forward propagate
        Calculate gradients by backward propagation
        Update model
    end For
    Evaluate Model
    Save the checkpoint
end For
```

Training Implementation in PyTorch

```
train_loader = DataLoader(dataset=TrainDataset() ...)
val_loader = DataLoader(dataset=ValDataset() ...)
model = LPRModel()
optim = torch.optim.Adam(model.parameters() ...)
...
for epoch in range(args.epochs):
   with tqdm(total=len(train_loader)) as pbar:
       for image, label in train_loader:
           predict = model(image)
           loss = loss_fn(predict.transpose(1, 2), label)
           optim.step()
           ...
   with torch.no_grad():
       for image, label in val_loader:
   if epoch == 0 or (epoch + 1) % args.checkpoint_period == 0:
       torch.save(model.state_dict(), path)
```

Checkpoint 4

- To execute the training script, run the following command:

```
$ python train.py
```

- As the training process takes around half an hour to complete, it is sufficient to display the training progress only. Here's an example of what it would look like:

```
Epoch   1/100 train_loss: 2.3468, val_loss: 2.1914, train_acc: ...
Epoch   2/100 train_loss: 1.8277, val_loss: 1.8763, train_acc: ...
Epoch   3/100 train_loss: 1.2882, val_loss: 1.4427, train_acc: ...
Epoch   4/100 train_loss: 0.9773, val_loss: 1.4076, train_acc: ...
Epoch   5/100 train_loss: 0.7762, val_loss: 1.0444, train_acc: ...
...
```

The End