Online Computer Vision and Patter Recognition Projects

Yuanqi Su

May 31, 2020

Project I: Neural Network

1 Model setting-up

Now it's time for you to set up the model for feature extraction, and combine it with linear classification for age prediction. Usually, we favor the end-to-end manner for the prediction. It means that we fed the input image from one side of the model and get the prediction from the other side of the model. Here, we need to set up a convolutional neural network for age prediction. In the part, you work with the 'network.ipynb'.

1.1 Preparing data

First please download image dataset(APPA-REAL DATABASE) from `http://chalearnlap.cvc.uab.es/dataset/26/data/45/description/`, and unzip it. Copy this folder to 'DATASET'. The files structure should looks like:

```
———Network
    |
    |———— DATASET
    |     |—— appa-real-release
    |     |—— valid
    |     |—— feature_test.npy
    |     |—— feature_train.npy
    |     ...
    |———— helperT.py
    |———— ignore_list.csv
    |———— networkCustom.ipynb
```

1.2 Model

In the part, you should implement the network in Table 1. It is a convolutional neural network that is composed of the convolutional, ReLU layer, max pooling and fully connected layers. The network has an image input size of 128-by-128. Here, we use the multi-class classification for the age prediction. For these part, you should set up the network, choose the SGD optimizer, and train the network with the fixed epochs.

As shown in Figure 1, the input to network is of fixed size 128×128 RGB image. The image is passed through a stack of convolutional layers, where the filters of different receptive fields were used. Except for the first 3 convolutional layers, all the other convolutional layers are with a very small receptive field:

Layer name	kernel_size	(fan_in, fan_out)	stride	padding
convolution	7	(3,64)	1	3
ReLU				
pooling	2		2	
convolution	5	(64,128)	1	2
ReLU				
convolution	5	(128,128)	1	2
ReLU				
pooling	2		2	
convolution	3	(128,256)	1	1
ReLU				
convolution	3	(256,256)	1	1
ReLU				
convolution	3	(256,256)	1	1
ReLU				
pooling	2		2	
convolution	3	(256,512)	1	1
ReLU				
convolution	3	(512,512)	1	1
ReLU				
convolution	3	(512,512)	1	1
ReLU				
pooling	2		2	
convolution	3	(512,512)	1	1
ReLU				
convolution	3	(512,512)	1	1
ReLU				
convolution	3	(512,512)	1	1
ReLU				
pooling	2		2	
linear		(512*4*4,4096)		
ReLU				
linear		(4096,4096)		
ReLU				
linear		(4096,101)		

Table 1: Network structure. fan_in and fan_out correspond to the input and output channel number.

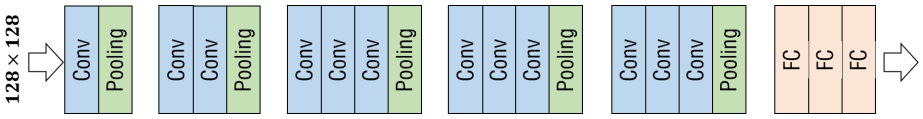

Figure 1: Network structure.

3×3. The receptive fields for the first 3 convolutional layers are 7×7, 5×5 and 5×5 respectively. The convolution stride is fixed to 1 pixel; the spatial paddings of convolutional layer make that the spatial resolution is preserved after convolution, i.e. the padding is 1-pixel for 3×3 convolutional layers, 2-pixel for 5×5 convolutional layer and 3-pixel for 7×7 convolutional layer. In the network, there are totally five max-pooling layers, which follow some of the convolutional layers. Max-pooling is all performed over a 2×2 pixel window, with stride 2.
After the stack of convolutional layers, there are three fully-connected (FC) layers. The first two have 4096 channels each, the third performs 101-way classification and thus contains 101 channels (one for each class). The final layer is the soft-max layer. Each of the convolutional and fully connected layers are followed by a rectified linear unit (ReLU) except the last fully connected layer. The detailed description of the network is summarized in Table 1.
To set up the model, the torch.nn module is the cornerstone of designing neural networks in PyTorch. This class can be used to implement a layer like a linear layer, a convolutional layer, a pooling layer, an activation function, and also an entire neural network. In 'networkCustom.ipynb', you are to fullfill the **AgeNet(nn.Module)** class. The class has two methods that you have to override.

- **__init__ function**. This function is invoked when you create an instance of the nn.Module. Here you will define the various parameters of a layer. Take the linear layer for example, you can define its input and output dimension, whether the bias is used.
- **forward function**. In the function, you define how your output is computed. This function doesn't need to be explicitly called, and can be run by just calling the nn.Module instance like a layer with the input as it's argument.

Besides, you are to implement the **train_ageNet** function where you do the following things.

- Instantiate an AgeNet;
- Move the network to Cpu or Gpu. (In Colab, you can use a GPU.)
- Define the criterion for the network.
- Set up the optimizer.

- Implement the forward and backward pass to update the parameters.

During the optimization process, you need to evaluate the parameter settings on validation set and save the setting with best validation performance.

Matters need attention

There are a lot of factors that influence your final precision. To make the comparison fair. **In the lab, the maximum training epoch is fixed to** 100**.**