COL 780
Computer Vision
Report - Assignment 2

Submitted By :

Arjun Bhardwaj
2016ME10754

Tanmay Goyal
2016ME20757

Link to the Google Drive folder containing output images :
https://drive.google.com/drive/folders/1qivSwPeUCYEdj0hdEFSihfPBuGWGhUps?usp=sharing

Dataset Creation

1. **Resizing** :-
   - The webcam image is rectangular (640x480 in our case), to fit the assignment specifications, the image was cropped into a square.
   - The image size defined as the input for neural net is 50x50 pixels. The square image (480x480) was scaled down to 50x50. The image was blurred before scaling down to avoid aliasing.

2. **Labeling** :-
   - We created a directory with one of the class names. Now we open the video through web camera and after every 6 frames save a frame as jpg image in the directory. This is done for each class (“Previous”, “Next”, “Stop” and “Other”) to create the dataset.

Training

1. **Initialising the code** :-

   First of all the OpenCV, Numpy and array modules are imported. We also imported torch, torchvision and torch.utils.

2. **Neural network** :-

   The 5 layer neural net had 3 convolution layers and 2 fully connected layers. The total trainable parameters for the network are
   1st Layer = 3 * 6 * 3 * 3 = 162
   2nd Layer = 6 * 8 * 7 * 7 = 2352
   3rd Layer = 8 * 4 * 3 * 3 = 288
   4th Layer = 4 * 2 * 2 * 20 = 320
   5th Layer = 20 * 4 = 80
   Total = 162 + 2352 + 288 + 320 + 80 = 3202
   There are two max pooling layers and the activation function is chosen as ‘relu’.

The hyperparameters of the network are discussed later.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 6, 3)         # 162
        self.pool1 = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(6, 8, 7)         # 2646
        self.pool2 = nn.MaxPool2d(3, 3)
        self.conv3 = nn.Conv2d(8, 4, 3)         # 405
        self.fc1 = nn.Linear(4* 2 * 2, 20)     # 1600
        self.fc2 = nn.Linear(20, 4)             # 80

    def forward(self, x):
        x = self.pool1(F.relu(self.conv1(x)))
        x = self.pool2(F.relu(self.conv2(x)))
        x = self.pool1(F.relu(self.conv3(x)))
        x = x.view(-1, 4 * 2 * 2)
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x
```

Fig. 1 Neural Network

3. **Training Dataset Loader** :-
   To load the images into train_dataset_loader, we create gesture_dataset by transforming images toTensor(). It is done by parsing the directory of all the training images by datasets.ImageFolder loader provided by pytorch. This directory has photos sorted into 4 folders - other, next, previous and stop. In this way we get 4 classes and images sorted in 4 classes.

4. **Loss criterion, Optimizer** :-
   We created an object of neural net. The criterion is chosen as the CrossEntropyLoss() criterion as this is considered the best for training softmax output. The optimizer has a learning rate equal to 0.001.

5. **Training neural network** :-
   We have chosen a batch size of 100. This batch size is sufficiently large for stochastic backpropagation to work and can also be processed very fast as compared to the dataset size (i.e ~ 15000).
   We have trained the neural net for 100 epochs, because the loss seemed to converge after 100 epochs. We then saved the trained neural network in model_path directory.

6. **Loss vs Epoch Graph** :-
   We then wrote the losses at each epoch in a csv file showing training quality.

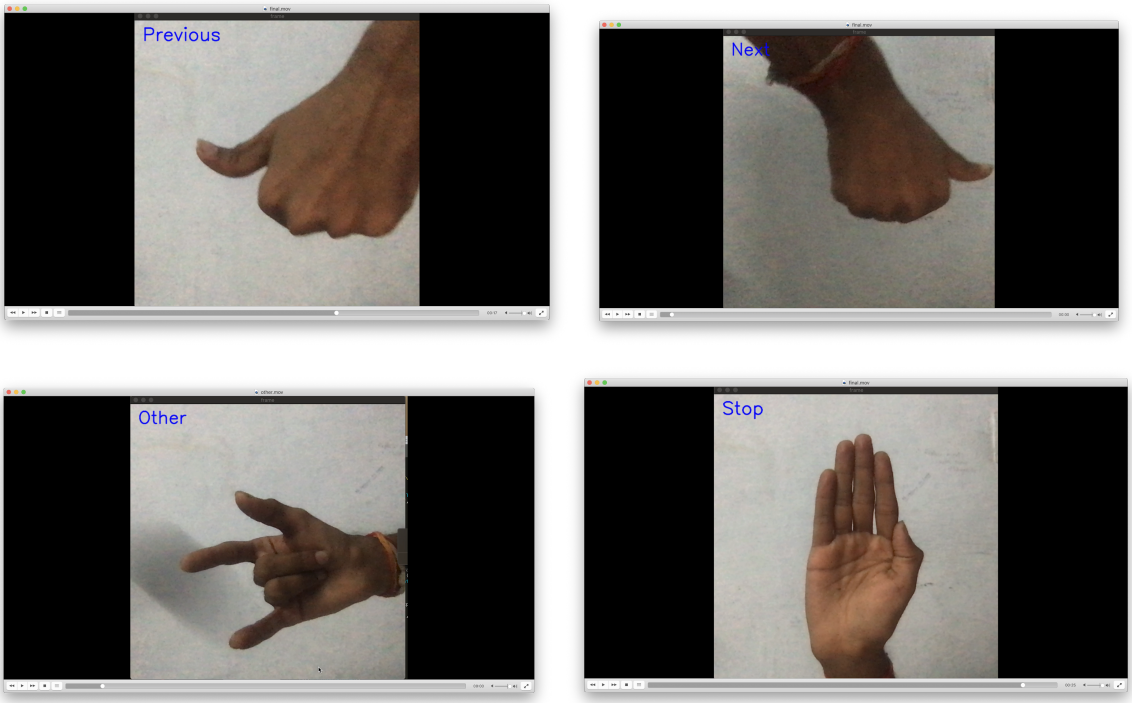
Validation

1. **Validation Set Loader:-**
   To load the images into test_dataset_loader, we create test_set by transforming images toTensor(). It is done by parsing the directory of all the training images to datasets.ImageFolder similar to the training

2. **Label Names**:-
   We defined 4 labels = ["Next", "Other","Previous", "Stop"].

3. **Loading the saved net** :-
   We called the save trained data (netPath) into load_state_dict to predict the input images.

4. **Outputs** :-
   We then parsed all the test images to neural net and the outputs corresponds to detected image.



5. **Accuracy** :-
   The percent of images labelled correctly out of the total validation set gives the accuracy of the network. The higher the accuracy the better the learning.

Real Time Webcam Inference

1. By default, the input is taken from a webcam (Device 0). Since the inference takes some time, 5 out of 6 frames are skipped and one frame is passed through the net.

2. The output received after passing through the net is a 4x1 tensor. The maximum value of the values is the detected class.

3. The detected class label is displayed on the original image.

Selecting the Model Hyperparameters

1. The total no. of images are around 15000, therefore, if the parameters are more than 15000, the learning would be merely fitting to the training data without any generalization. So, the trainable parameters should be strictly less than 15000.

2. For good generalization, the ratio of the no. of images to the trainable parameters should be roughly around 5. Therefore, the parameters should be around 3000.

3. The first layer should detect features such as edges, so a 3x3 kernel would be optimal for edge detection. The no. of features are chosen to be 6

4. Then max pooling is done, by a 2x2 kernel, to reduce the layer size.

5. The next layer should combine features from different parts of the image, so the kernel is kept somewhat big.

6. The fully connected layers will combine the information from all the parts of the image.

7. The output layer has 4 nodes since four output classes are possible.