COMP6721 Applied Artificial Intelligence (Fall 2020)

Project Assignment Part I

AI Face Mask Detector

Team Name: SS_G11

Team members:

Lin Li 40044486

Wenhui Guo 40092514

Index

0. Introduction

In this project, we developed a Deep Learning Convolutional Neural Network (CNN) using PyTorch and train it to recognize three different classes:

(1) Person without a face mask,
(2) Person with a face mask, and
(3) Not a person (i.e., any other image)

In this first phase of the project, we focused on a proper design of our datasets, collecting suitable training data, setting up the complete AI learning & evaluation process and gathering first results. We will further improve it in the second phase of the project.

1. Dataset

1.1 Create dataset

In this part we will describe how we built our dataset, the source of collected images and provide details on the dataset.

In order to get better performance for our project, we decided to build our own dataset by collecting images from existing several datasets rather than directly re-use existing datasets. We collected images from public datasets, CelebFaces Attributes Dataset (CelebA); from online community of data scientists and learning practitioners, Kaggle; from personal source code datasets, Git. More details about the source in the Reference Section.

1.2 Analyse dataset

Our dataset has two parts, train dataset part and test dataset part. Each part includes three classes.

- Person with a face mask,
- Person without a face mask, and
- Not a person (i.e., any other image).

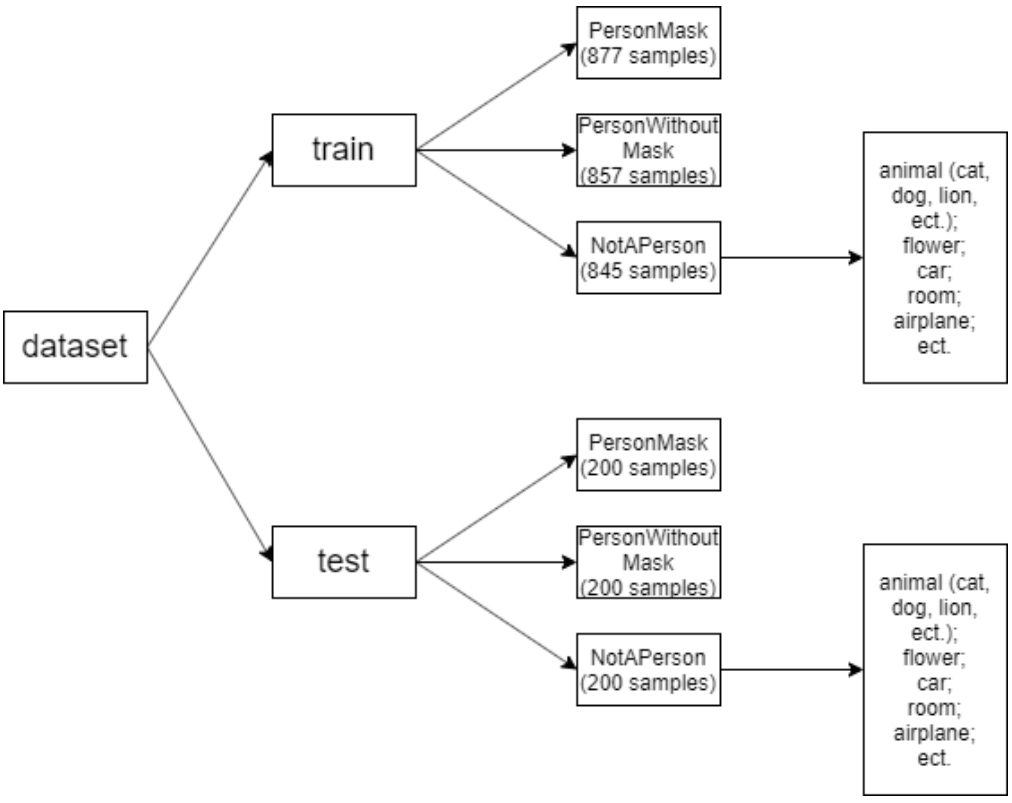


Fig.1 Dataset's size and structure

Some more details about the dataset:

- In the person with mask train dataset, there are 420 female adult samples, 400 male adult samples, and 57 child samples. There are 185 dark skin person samples. 743 samples are face front photos, 132 samples are side-face photos. Masks of around 696 samples are in the center of the photos.
- In the person with mask test dataset, there are 103 female adult samples, 86 male adult samples, and 11 child samples. There are 44 dark skin person samples. 151samples are face front photos, 42 samples are side-face photos. Masks of around 178 samples are in the center of the photos.
- In the person without mask train dataset, there are 398 female adult samples, 409 male adult samples, and 38 child samples. There are 210 dark skin person samples. 812 samples are face front photos, 45 samples are side-face photos. Masks of around 785 samples are in the center of the photos.
- In the person without mask test dataset, for each character, there are around one quarter proportion with person without mask train dataset.
- In the not person train dataset, there are 174 cat samples, 151 dog samples, and 352 other hair wild animals' samples. There are 58 plant and fruit samples, 96 transport tools (car, moto, airplane) samples, 14 room photos.
- In the not person test dataset, for each type, there are around one quarter proportion with not person train dataset.

1.3 Pre-process dataset

When we install PyTorch, we also install torchvision library which contains models and transformation operations generally used in the image pre-processing.

To pre-process the data, by using transform method in torchvision, we do resize and center crop when loading each image. The processed dataset is saved to variables "trainset" or "testset", to be used later.

1.4 Load dataset

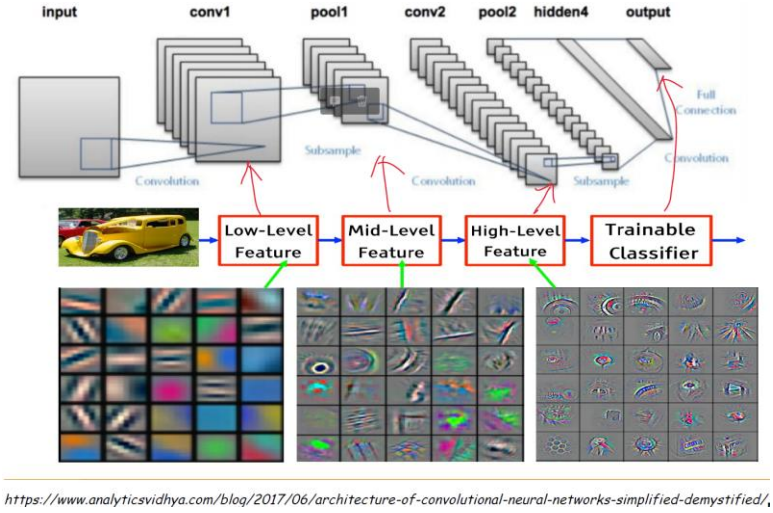
We use DataLoader in pytorch library to provide API for loading datasets. By creating a DataLoader object, we load the previously created variables "trainset" or "testset" which contain pre-processed images, and feed to our Convolutional Neural Network (CNN) model for training and testing purposes.

2. CNN Architecture

In this part we will describe the architecture of our CNN and provide details on the training.

2.1 CNN class for Convolutional Neural Network architecture

In order to create a suitable Convolutional Neural Network (CNN) architecture, implement it in PyTorch, and train it using our dataset, we first create a class inheriting from the nn.Module to define different layers of the network based on provided network architecture above.
We set 2 Convolutional Layers with activation and max pooling, the flatten it to one-dimension vector, then pass it through 3 Full Connection layers, like shown in the teaching material:



https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/,

Fig.2 Convolutional Neural Network (CNN) architecture shown in lecture

In our code, we have provided comment for every step, as shown below:

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()

        self.conv1 = nn.Conv2d(3, 6, 5)  # input channel = 3, output channel = 6, apply 6 filters of 5*5
        self.conv2 = nn.Conv2d(6, 16, 5)  # input channel = 6, output channel = 16, apply 16 filters of 5*5

        self.fc1 = nn.Linear(5 * 5 * 16, 120) # input is 5*5*16 = 400*1, output 120*1, one dimension vector
        self.fc2 = nn.Linear(120, 84) # input is 120*1, output 84*1, one dimension vector
        self.fc3 = nn.Linear(84, 3) # input is 84*1, output 3*1, because there are 3 classes

    def forward(self, x):
        # input x, then go thru conv1, then activation function relu, then pooling
        x = self.conv1(x) # output size: 28*28*6
        x = F.relu(x)
        x = F.max_pool2d(x, 2) # output size: 14*14*6 (apply 2*2 pooling (filter size = 2, stride =2) to 28*28*6)

        # input x (14*14*6), then go thru conv2, then activation function relu, then pooling
        x = self.conv2(x) # output size: 10*10*16
        x = F.relu(x)
        x = F.max_pool2d(x, 2) # output size: 5*5*16 (apply 2*2 pooling (filter size = 2, stride =2) to 10*10*16)

        # flatten the activation maps to one dimension vector
        x = x.view(x.size()[0], -1)

        # pass thru 3 full connection layers
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Fig.3 Our Convolutional Neural Network (CNN) setup

And diagram to show workflow:

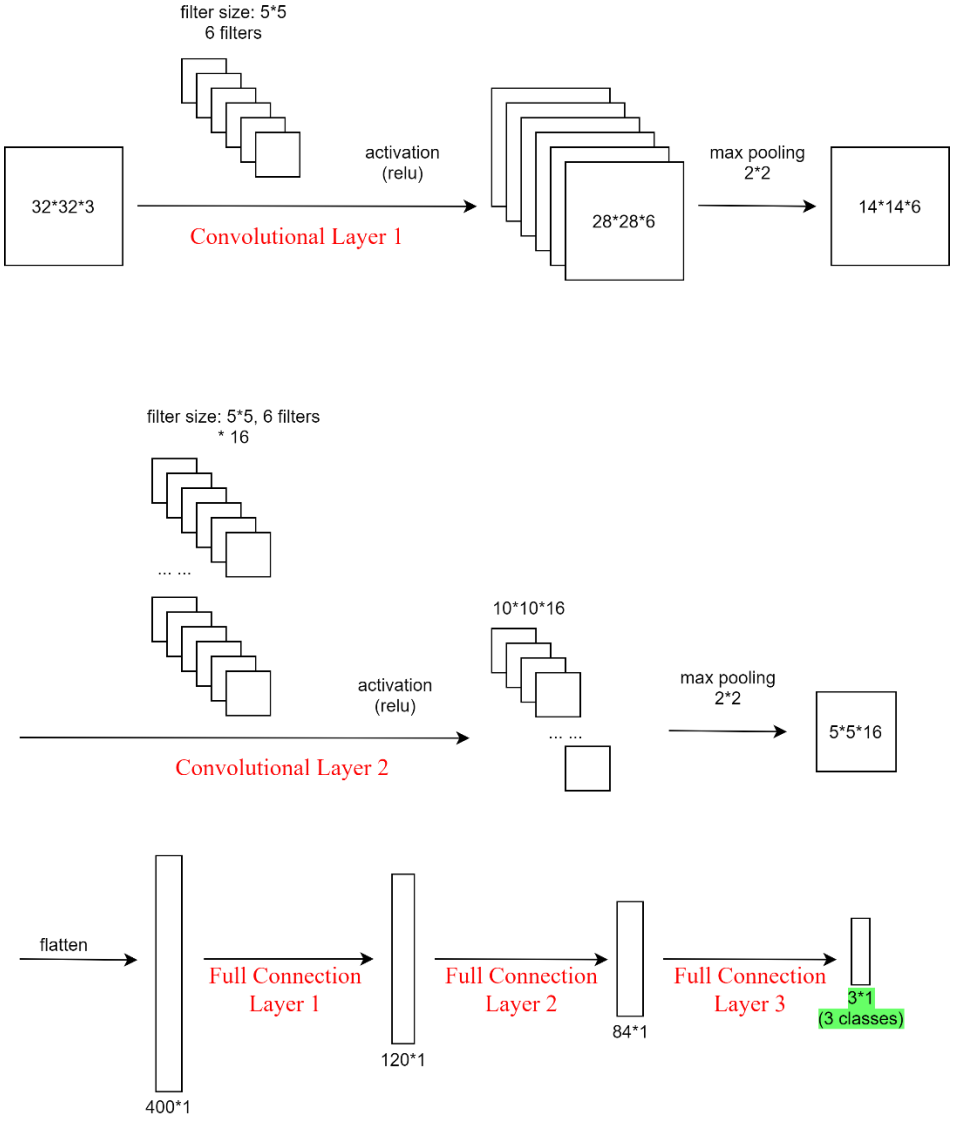


Fig.4 Our Convolutional Neural Network (CNN) setup diagram

2.2 Train using our CNN model

In the training phase, we use “train_loader” to load the images from given path, the images in each sub-folder is labeled with the sub-folder name.

Following what we learned from the lecture and lab, we create an instance of the Convolution class we defined in previous part, then define the optimizer and loss function.

We set the learning rate to be 0.001, this number is small so the weight won’t get changed violently.

We train the model for 10 epochs, for now we can get a final accuracy of 91.17%.

In second phase of the project we can increase this number to achieve better performance.

When training phase is completed, the parameters of our CNN are saved to 2 files:

net.pkl

net_params.pkl

This will be read by the testing phase to evaluate our model.

2.3 Test using our CNN model

In the testing phase, we use torch.load() method from pytorch library to load the parameters of our CNN which was generated from previous training phase.

Then we use “test_loader” to load the images from given path, the expected labels are the name of sub-folders, this information will be used to compare with the predicted labels, to evaluate the performance of our CNN model.

3. Evaluation

We provide the data as below to analyze and evaluate our model:

3.1 Log for training our CNN:

Average loss and accuracy of each 200 images

```
epoch [1,   200]  Average loss: 1.098  Average accuracy: 25.00 %
epoch [1,   400]  Average loss: 1.097  Average accuracy: 50.00 %
epoch [1,   600]  Average loss: 1.095  Average accuracy: 25.00 %
epoch [2,   200]  Average loss: 1.076  Average accuracy: 25.00 %
epoch [2,   400]  Average loss: 1.009  Average accuracy: 50.00 %
epoch [2,   600]  Average loss: 0.962  Average accuracy: 75.00 %
epoch [3,   200]  Average loss: 0.755  Average accuracy: 50.00 %
epoch [3,   400]  Average loss: 0.660  Average accuracy: 50.00 %
epoch [3,   600]  Average loss: 0.601  Average accuracy: 25.00 %
epoch [4,   200]  Average loss: 0.567  Average accuracy: 75.00 %
epoch [4,   400]  Average loss: 0.544  Average accuracy: 75.00 %
epoch [4,   600]  Average loss: 0.482  Average accuracy: 75.00 %
epoch [5,   200]  Average loss: 0.433  Average accuracy: 100.00 %
epoch [5,   400]  Average loss: 0.443  Average accuracy: 75.00 %
epoch [5,   600]  Average loss: 0.441  Average accuracy: 75.00 %
epoch [6,   200]  Average loss: 0.406  Average accuracy: 75.00 %
epoch [6,   400]  Average loss: 0.409  Average accuracy: 50.00 %
epoch [6,   600]  Average loss: 0.361  Average accuracy: 75.00 %
epoch [7,   200]  Average loss: 0.316  Average accuracy: 100.00 %
epoch [7,   400]  Average loss: 0.351  Average accuracy: 100.00 %
epoch [7,   600]  Average loss: 0.316  Average accuracy: 100.00 %
epoch [8,   200]  Average loss: 0.287  Average accuracy: 100.00 %
epoch [8,   400]  Average loss: 0.289  Average accuracy: 100.00 %
epoch [8,   600]  Average loss: 0.253  Average accuracy: 75.00 %
epoch [9,   200]  Average loss: 0.227  Average accuracy: 100.00 %
epoch [9,   400]  Average loss: 0.255  Average accuracy: 100.00 %
epoch [9,   600]  Average loss: 0.265  Average accuracy: 50.00 %
epoch [10,  200]  Average loss: 0.216  Average accuracy: 100.00 %
epoch [10,  400]  Average loss: 0.223  Average accuracy: 75.00 %
epoch [10,  600]  Average loss: 0.216  Average accuracy: 100.00 %
```

Fig.5 Printed training log

It is obvious that along with the process of training, the average loss decreases, while average accuracy is augmented.

3.2 A table of results showing the accuracy, precision, recall and F1-measure

```
Accuracy of the test dataset : 91.17 %
+------------------+-------------+----------+------------+-----------+
|                  |   precision |   recall |   f1-score |   support |
+==================+=============+==========+============+===========+
| NotAPerson       |       0.897 |    0.875 |      0.886 |       200 |
+------------------+-------------+----------+------------+-----------+
| Person           |       0.919 |     0.97 |      0.944 |       200 |
+------------------+-------------+----------+------------+-----------+
| PersonMask       |       0.918 |     0.89 |      0.904 |       200 |
+------------------+-------------+----------+------------+-----------+
|                  |             |          |            |           |
+------------------+-------------+----------+------------+-----------+
| Average          |       0.911 |    0.912 |      0.911 |           |
+------------------+-------------+----------+------------+-----------+
| Weighted average |        0.91 |    0.911 |       0.91 |           |
+------------------+-------------+----------+------------+-----------+
```

Fig.6 A table of results showing the accuracy, precision, recall and F1-measure

The results of accuracy, precision, recall and F1-measure are acceptable and promising, means we learned the knowledge and skill to build a CNN to do classification.

3.3 Confusion matrix:

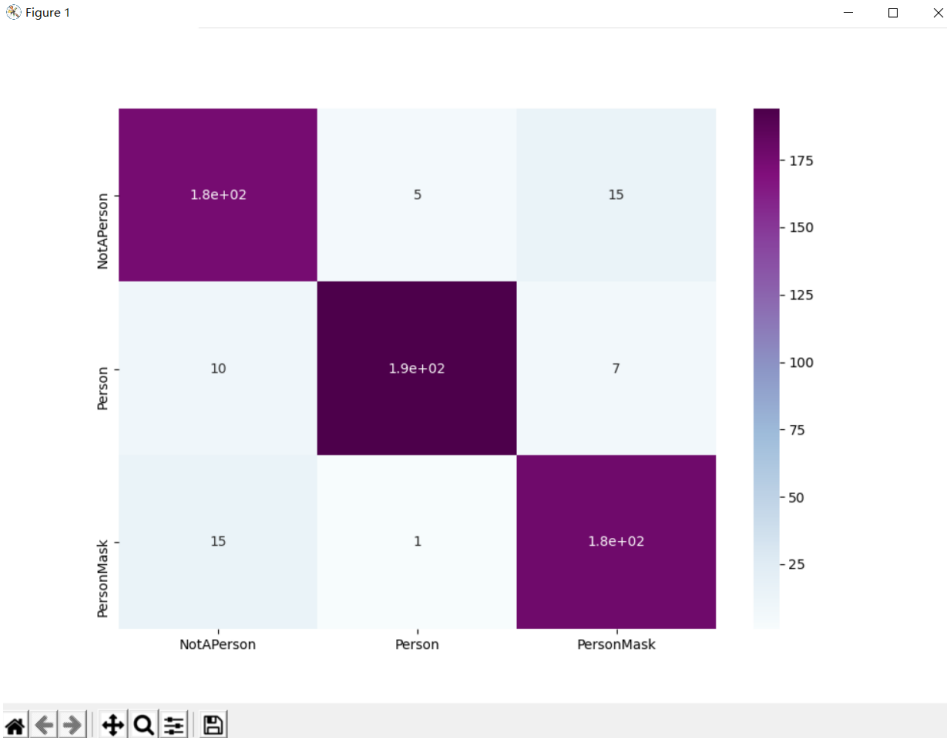


Fig.7 Confusion matrix

As we can see from the Confusion matrix, our CNN model successfully classified majority of the test data.

3.4 Running time

Training phase takes: 0:03:02.110196
Testing phase takes: 0:00:41.731781

3.5 Discuss the results and explain how and where we want to improve during the second phase of the project.

- We could improve on the structure of our CNN:

- For structure:
  - We could increase the number of Convolutional Layers so that the accuracy will be further improved;
  - We could add normalization to some steps in Convolutional Layers;
  - We could add the step of “dropout random nodes” in Full Connection Layers to avoid certain feature has huge impact on the result.

- For parameters:
  - We could change the kernel size of the Convolutional Layers (currently always 5*5);
  - We could add padding (currently no padding);
  - We could try other activation functions (currently relu).

- For training phase:
  - We could try different loss function (currently using CrossEntropyLoss());
  - We could use other optimizer (currently using optim.SGD);
  - We could change the learning rate to see if the performance can be further improved (current learning rate = 0.001);
  - We could increase the number of images in training set, until it reaches the top of the learning curve (where accuracy hardly could be improved by increasing the size of the training set);
  - We could also increase the number of epochs to minimize the loss, and maximize accuracy, precision, recall and F1-measure.

4. Reference Section

This part contains citations to all relevant resources that we have consulted, even if it was just to inspire us.

[1] CelebFaces Attributes Dataset, by Ziwei Liu, Ping Luo, Xiaogang Wang, Xiaoou Tang, Multimedia Laboratory, The Chinese University of Hong Kong http://mmlab.ie.cuhk.edu.hk/projects/CelebA.html

[2] Animal face, by Larxel, https://www.kaggle.com/andrewmvd/animal-faces

[3] Face Mask Detection, by Larxel, https://www.kaggle.com/andrewmvd/face-mask-detection

[4] With/Without Mask, by Niharika Pandit, https://www.kaggle.com/niharika41298/withwithout-mask

[5] Face Mask Classification, by Dhruv Makwana, https://www.kaggle.com/dhruvmak/face-mask-detection

[6] Face Mask Detection, by Edward Zhang, https://www.kaggle.com/sshikamaru/face-mask-detection

[7] Mask detection, by abdelatif, https://www.kaggle.com/moussaid/mask-detection

[8] COVID-19 Mask Detector, by Niharika Pandit, https://www.kaggle.com/niharika41298/covid-19-mask-detector

[9] LFW-People (Face Recognition), by Atul Anand{Jha}, https://www.kaggle.com/atulanandjha/lfwpeople

[10] Real-World-Masked-Face-Dataset, by X-zhangyang, https://github.com/X-zhangyang/Real-World-Masked-Face-Dataset

[11] Face-Mask-Detection, by chandrikadeb7, https://github.com/chandrikadeb7/Face-Mask-Detection/tree/master/dataset/with_mask

[12] 60,000+ Images of cars, by Paul, https://www.kaggle.com/prondeau/the-car-connection-picture-dataset

[13] Flower Color Images Set for Classification, by Olga Belitskaya, https://www.kaggle.com/olgabelitskaya/flower-color-images
[14] STL-10 Image Recognition Dataset, by Jessica Li, https://www.kaggle.com/jessicali9530/stl10
[15] Documentation of pytorch, https://pytorch.org/
[16] Build our own datasets and load to pytorch, https://pytorch.org/docs/stable/torchvision/datasets.html
[17] Lab 7 of COMP6721, refer to moodle
[18] https://www.youtube.com/watch?v=YRhxdVk_sIs
[19] Youtube video: How Convolutional Neural Networks work, https://www.youtube.com/watch?v=FmpDIaiMIeA&ab_channel=BrandonRohrer
[20] "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way", https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
[21] Precision, recall, and F1measure, From Wikipedia: https://en.wikipedia.org/wiki/Precision_and_recall
[22] A Comprehensive Tutorial to learn Convolutional Neural Networks from Scratch (deeplearning.ai Course #4), https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/
[23] Documentation of pytorch: TRAINING A CLASSIFIER, https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
[24] Neural Network Programming - Deep Learning with PyTorch, https://deeplizard.com/learn/video/0LhiS6yu2qQ
[25] Documentation of pytorch , https://pytorch-lightning.readthedocs.io/en/latest/metrics.html
[26] Architecture of Convolutional Neural Networks (CNNs) demystified, https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/