COMP6721 Applied Artificial Intelligence (Fall 2020)

Project Assignment Part II

AI Face Mask Detector

Team Name: SS\_G11

Team members:

Lin Li 40044486

Wenhui Guo 40092514

*Index*

*I. Report of Project Part I*

(As confirmed by Professor Witte, we need to include report of Project Part I in this report)

**1. Dataset**

**2. CNN Architecture**

**3. Evaluation**

*II. Updates in Project Part II*

**1. Dataset**

**2. CNN Architecture improvement**

**3. Image Normalization**

**4. Visualize CNN performance by showing image with its expected label and predicted label**

*III. K-Fold Cross Validation*

**1. Definition**

**2. Implementation**

**3. Evaluation**

*IV. Reference Section*

**1. Reference Section for Project Part I**

**2. Reference Section for Project Part II**

*I. Report of Project Part I*

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).

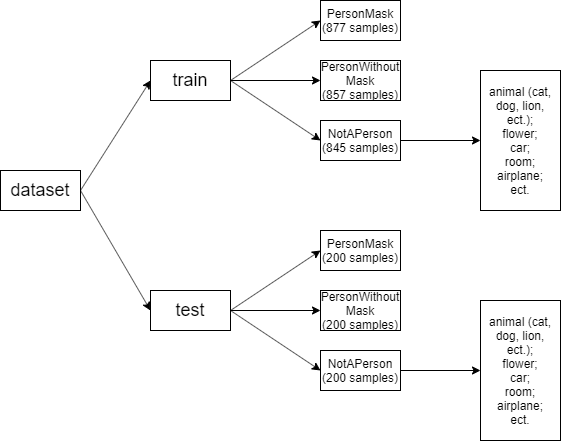


Fig.1 Project Part I Dataset’s size and structure

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:

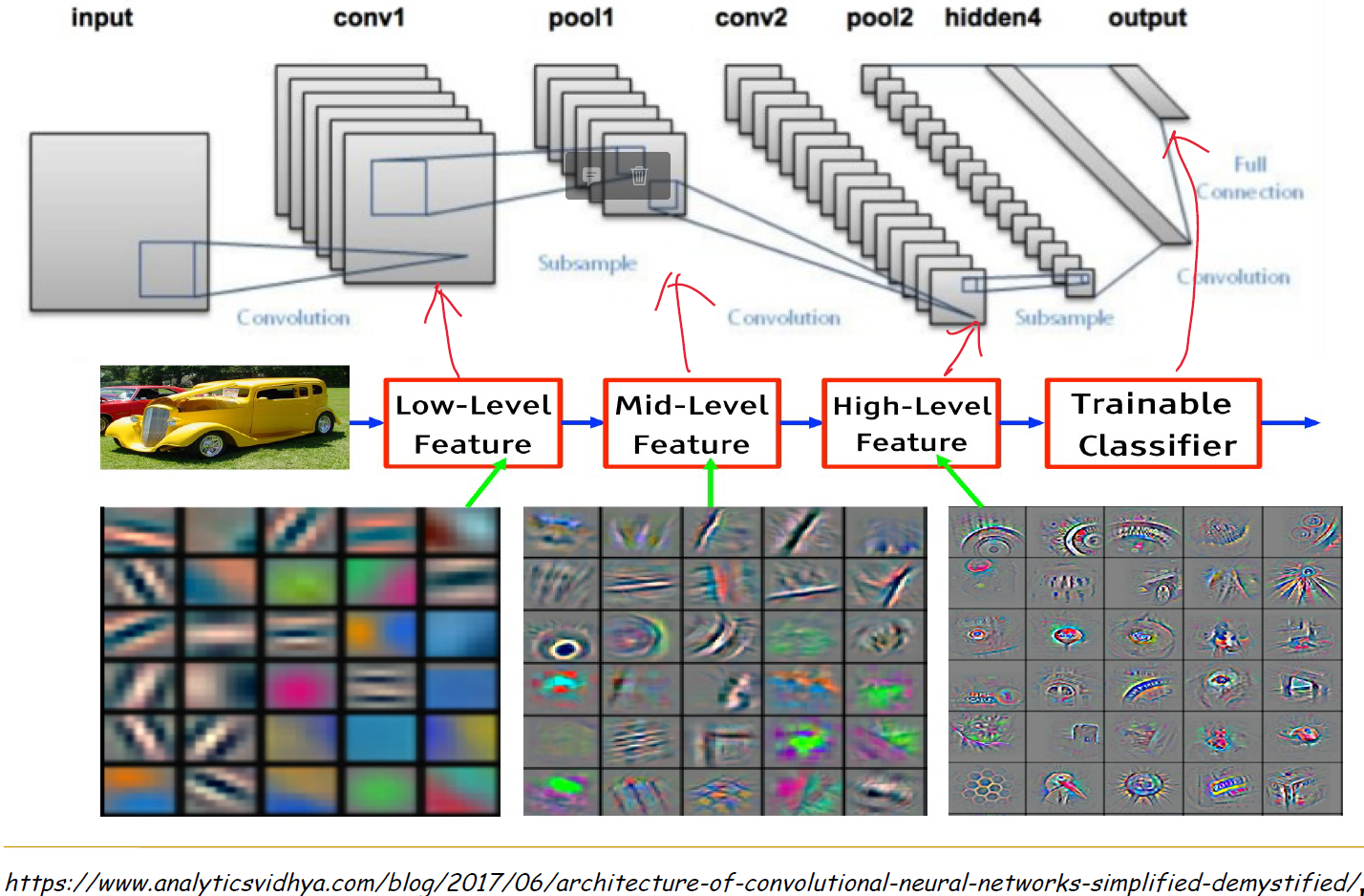


Fig.2 Convolutional Neural Network (CNN) architecture shown in lecture

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

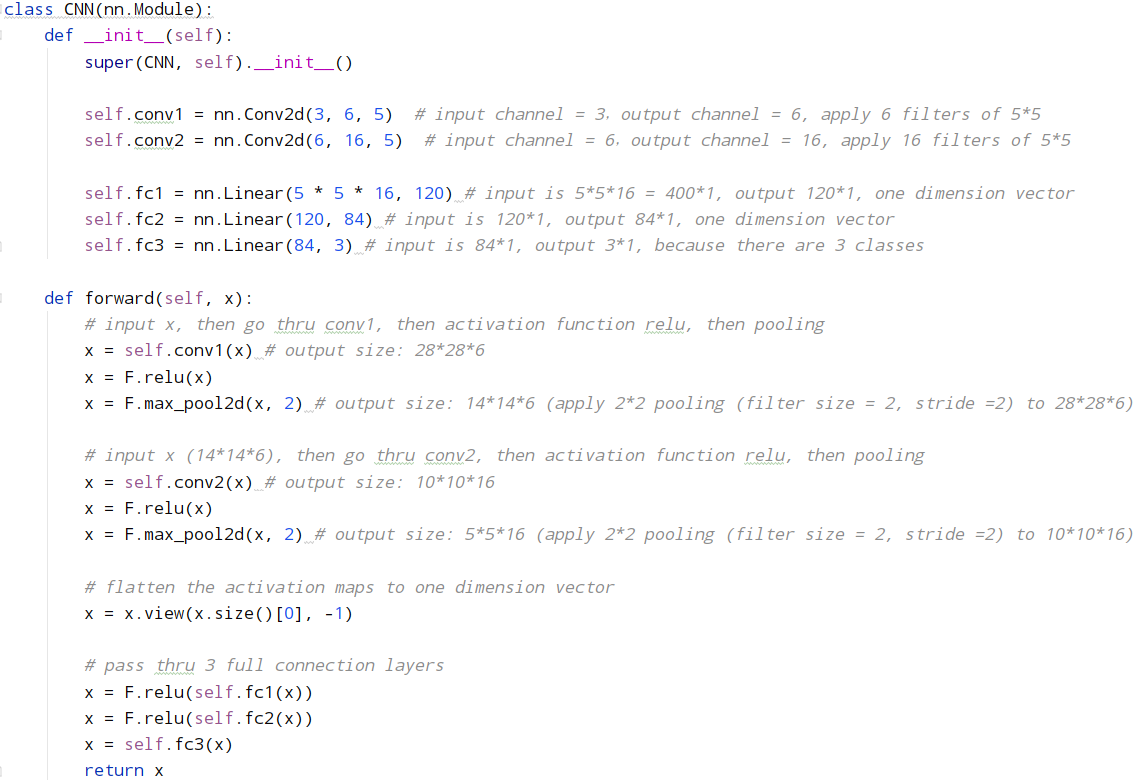


Fig.3 Our Convolutional Neural Network (CNN) setup

And diagram to show workflow:

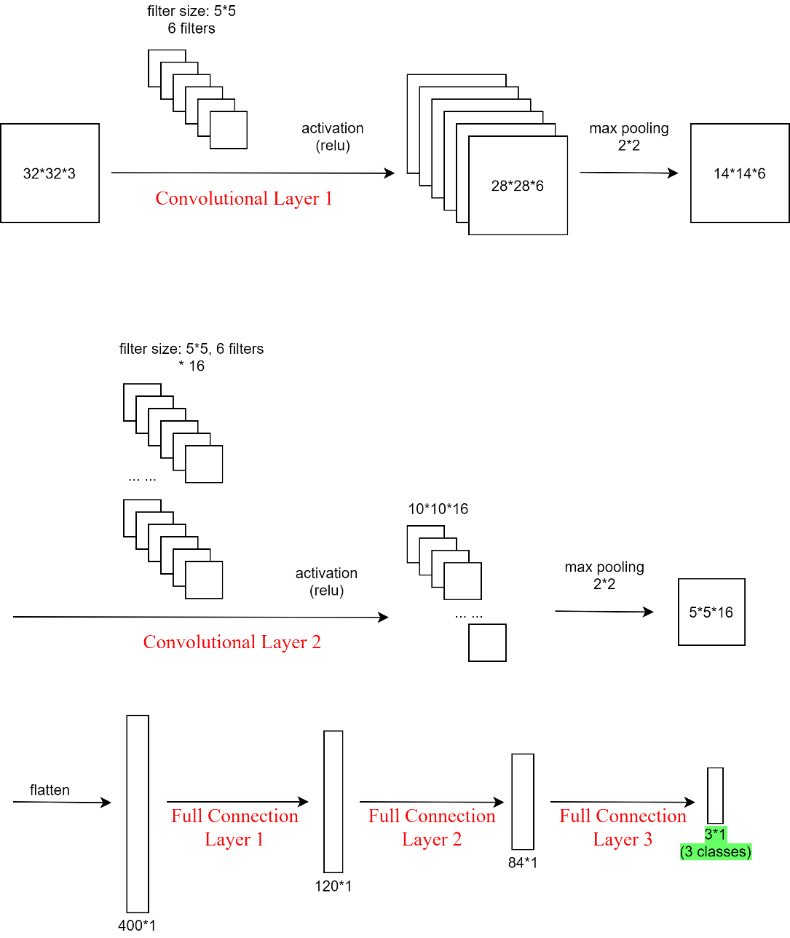


Fig.4 Our Convolutional Neural Network (CNN) setup diagram

***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

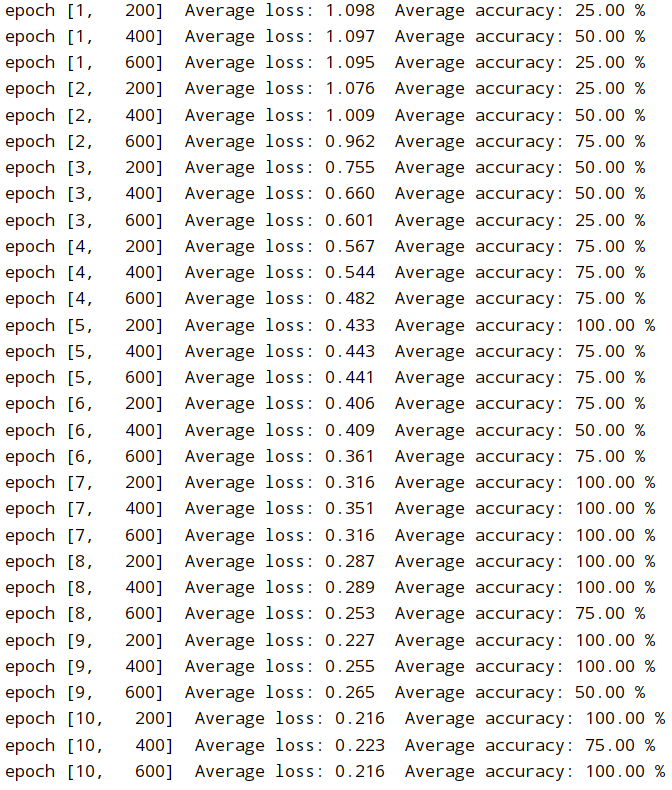


Fig.5 Printed training log

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***

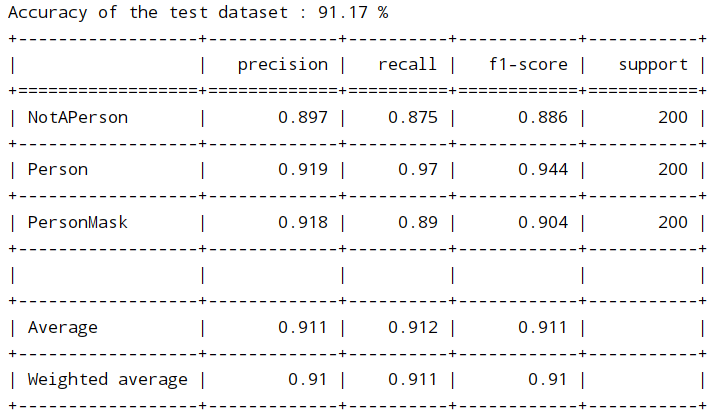


Fig.6 A table of Project Part I results showing the accuracy, precision, recall and F1-measure

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:***

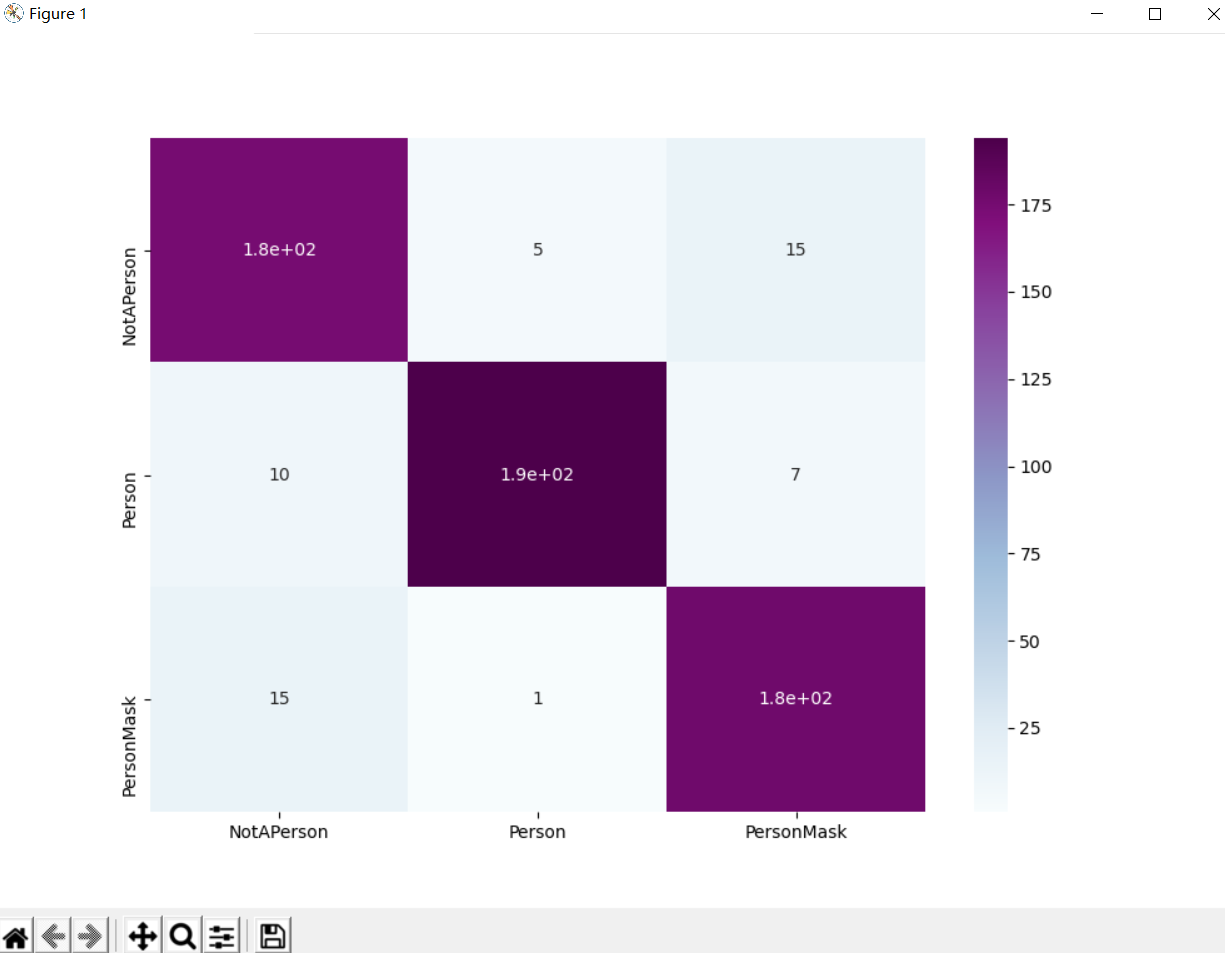


Fig.7 Confusion matrix

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.

*II. Updates in* *Project Part II*

Thanks for the feedback and guidance from our TA Mr. Soroosh Shahtalebi during the demo of the first part, in the Project Part II, accordingly we improved multiple aspects of our implementation of Convolutional Neural Network to recognize three different classes (Person without a face mask, Person with a face mask, Not a person), including: increased dataset, re-structured and improved CNN architecture, fixed image normalization, visualize CNN performance by showing image with its expected label and predicted label, etc.

As requested, we implemented a 10-fold cross-validation (with random shuffling) on our AI, and analyzed the performance evaluation in the report, including the result comparison with our previous, standard evaluation (fixed training/test split).

**1. Dataset**

**1.1 increase size of dataset**

In order to get better performance of our project, we collect much more dataset. The size of dataset for the second iteration is 22,500, that is, each class (Person, PersonMask, NotAPerson) has 7500 images inside.

The size of dataset for the first iteration is around 3000, that is, each class (Person, PersonMask, NotAPerson) has about 1050 images inside. Comparing with the dataset for the first iteration, for the second iteration, there are still six smaller dataset. The figures show as Fig.1 Project Part I Dataset’s size and structure and Fig.8 Project Part II Dataset’s size and structure

**1.2 split dataset into training (80%) and testing (20%). Out of the 80% training dataset, do a 10-fold cross validation**

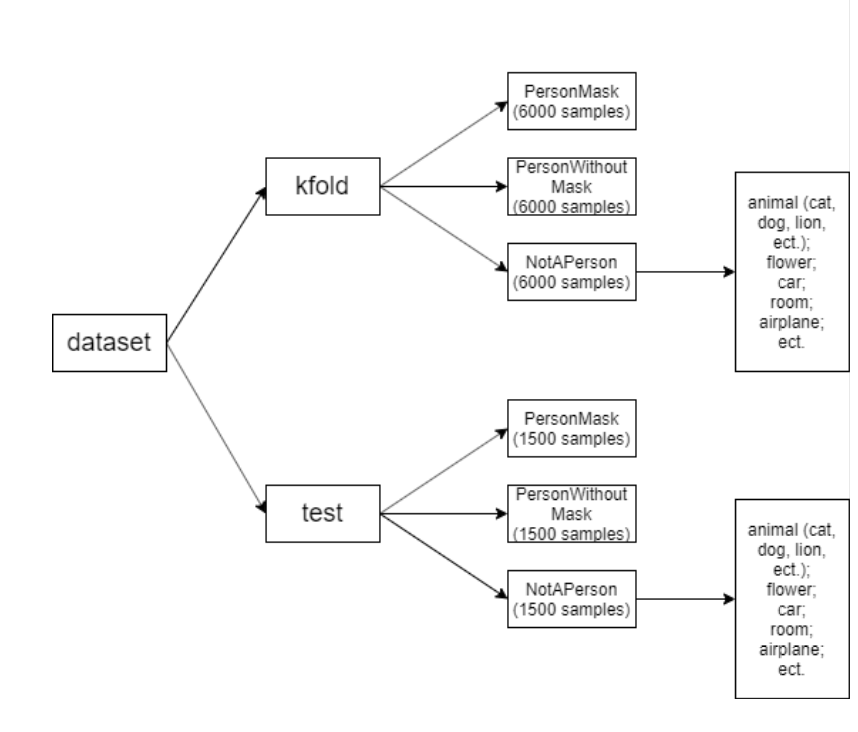


Fig.8 Project Part II Dataset’s size and structure

Some more details about the Project second iteration dataset:

* In the person with mask kfold dataset, there are 3100 female adult samples, 2800 male adult samples, and 100 child samples. There are 950 dark skin person samples. 5400 samples are face front photos, 600 samples are side-face photos. Masks of around 4800 samples are in the center of the photos.
* In the person with mask test dataset, for each character, there are around one quarter proportion with person with mask kfold dataset.
* In the person without mask kfold dataset, there are 2900 female adult samples, 2900 male adult samples, and 200 child samples. There are 1200 dark skin person samples. 5200 samples are face front photos, 800 samples are side-face photos. Masks of around 5000 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 kfold dataset.
* In the not person train dataset, there are 1000 cat samples, 850 dog samples, and 1900 other hair wild animals’ samples. There are 500 plant and fruit samples, 1636 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 kfold dataset.

**2. CNN Architecture improvement**

With the help of K-Fold Cross Validation and test effort, we are able to adjust hyperparameters of the project and the architecture of our Convolutional Neural Network to achieve better performance, based on evaluation.

**2.1 Increase the number of Convolutional Layers**

In part I of the project, we used 2 Convolutional Layers to train our AI, the purpose is to extract low level features, then high level features.

In part II of the project, we added 2 more Convolutional Layers for training. With the 4 Convolutional Layers, the system is able to extract features from images in 4 different levels, and this helps to improve the performance of our system.

**2.2 Add padding during processing**

In part II of the project, we applied padding during the training of the Convolutional Neural Network. By starting the filter outside the frame of the image, it gives the pixels on the border of the image more of an opportunity for interacting with the filter, more of an opportunity for features to be detected by the filter, and in turn, an output feature map that has the same shape as the input image.

Sample code of applying padding to Convolutional Layers:



**2.3 Add normalization to Convolutional Layers**

Batch normalization is a technique that can improve the learning rate of a neural network.

It does so by minimizing internal covariate shift which is essentially the phenomenon of each layer’s input distribution changing as the parameters of the layer above it change during training.

Sample code of applying normalization to Convolutional Layers:



**2.4 Randomly drop some nodes**

During training, randomly zeroes some of the elements of the input tensor with probability p. Each channel will be zeroed out independently on every forward call.

This has proven to be an effective technique for regularization and preventing the co-adaptation of neurons.

Sample code of applying random dropout:



**3. Image normalization**

As pointed out by our TA Mr. Soroosh Shahtalebi in Project Part I demo, we optimized our image process for its normalization.

When load images with method torchvision.datasets.ImageFolder, during transform, we pass in mean and standard deviation to method torchvision.transforms.Normalize(mean, std), normalize each channel of the input image (torch.\*Tensor) with formula:

output[channel] = (input[channel] - mean[channel]) / std[channel]

With this step, we normalize the dimensions of RGB (3 channels) image to [-1, 1].

In the testing phase, to visualize the performance of our CNN, we need to show the images together with their expected class and predicted class, so un-normalization is necessary to transform the image back to its RGB value. We calculated the corresponding RGB value of images [0, 255] in method def imshow(), then visualize it by plt.imshow() and plt.show().

**4. Visualize CNN performance by showing image with its expected label and predicted label**

In Project Part I, during test phase, we managed to print the expected labels and predicted labels, one batch size at a time, in console, for comparison and evaluation.

As suggested by our TA Mr. Soroosh Shahtalebi, we further improved this during the Project Part II, by adding the function of showing the images by methods plt.imshow() and plt.show(), with printing the expected labels and predicted labels beside each image.

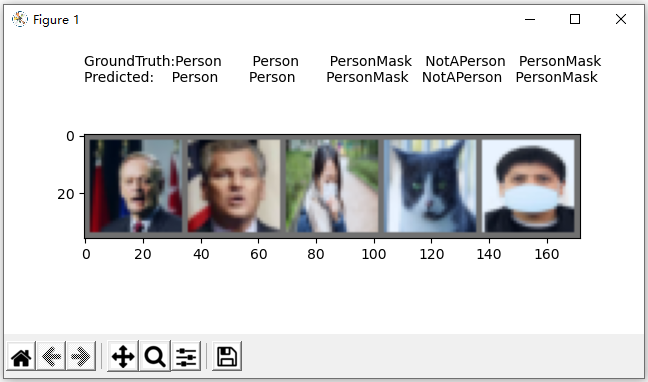
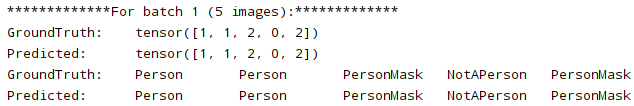


Fig.9 The expected labels and predicted labels beside each image

In the meantime, this information is printed to console too:



The approach of showing image with its expected label and predicted label facilitate visualization of the CNN performance, it provides an opportunity for us to evaluate the architecture and improve.

*III: K-Fold Cross Validation*

**1. Definition**

K-Fold Cross Validation is a resampling procedure used to evaluate machine learning models on a limited data sample. It generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

Process of applying K-Fold Cross Validation:

(1) Shuffle the dataset randomly;

(2) Split the dataset into k consecutive groups, called folds, of equal sizes (if possible).

(3) For each unique group:

Take the group as a hold out or test data set;

Take the K-1 groups as a training data set, the remaining 1 group as testing data set;

Fit a model on the training set and evaluate it on the test set;

Retain the evaluation score and discard the model;

(4) Summarize the skill of the model using the sample of model evaluation scores.

**2. Implementation**

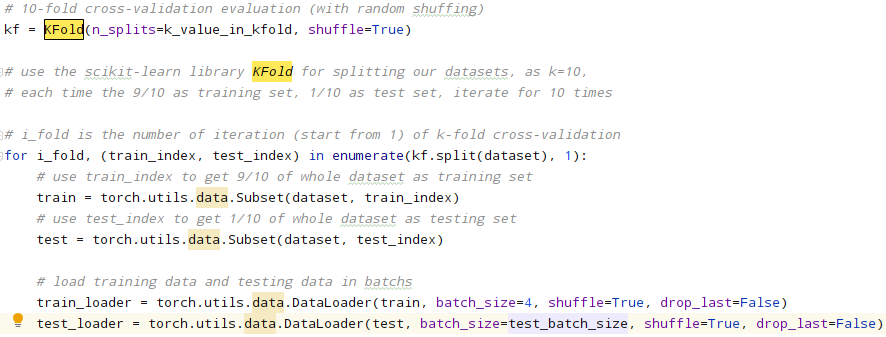
According to the communication with our TA Mr. Soroosh Shahtalebi, following his suggestion, firstly we split our whole dataset into training set (80%) and testing set (20%); then we apply 10-Fold Cross Validation to the training set. After that, we apply the result to the testing set. This is a robust way of developing ML models.

In the part of 10-Fold Cross Validation, after loading and pre-process images (resize, center crop, normalize, etc.), we use scikit-learn library (sklearn.model\_selection.KFold()) to split our dataset.

With random shuffling enabled, we are able to let our CNN get trained on fixed percentage but random pieces of our dataset, this is very important to lower the system bias.

With K=10 in K-fold, the system uses torch.utils.data.Subset() to assign 9/10 of whole dataset as training data, and 1/10 as testing data.

We use a for loop to enable the 10 iterations of the 10-fold cross validation, where the CNN is trained and tested for 10 times.



For each iteration, we obtain the evaluation measurements including accuracy, precision, recall, F1-measure, and confusion matrix. On completion of the 10 iterations, we calculate the average measurements of all 10 iterations and show the result.

Finally, we apply the result to the testing set.

**3. Evaluation**

**3.1 Compare the** **results of Project Part II with Project Part I**

Compare the results figure Fig.10 A table of Project Part II results showing the accuracy, precision, recall and F1-measure below with the Project Part I, which shows as Fig.6 A table of Project Part I results showing the accuracy, precision, recall and F1-measure.

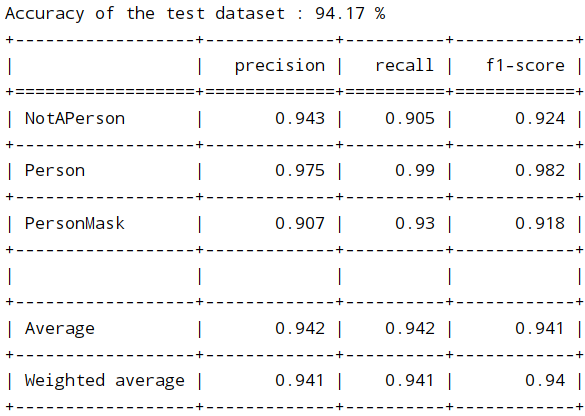


Fig.10 A table of Project Part II results showing the accuracy, precision, recall and F1-measure

As we can see from these two figures, the results of Project Part II behaviour much better than Project Part I. The Project Part I is implemented by using the standard evaluation, which has fixed training/test split. The Project Part II is implemented by using the K-Fold Cross Validation, which which split the dataset into k consecutive groups, then take the K-1 groups as a training data set, the remaining 1 group as testing data set.

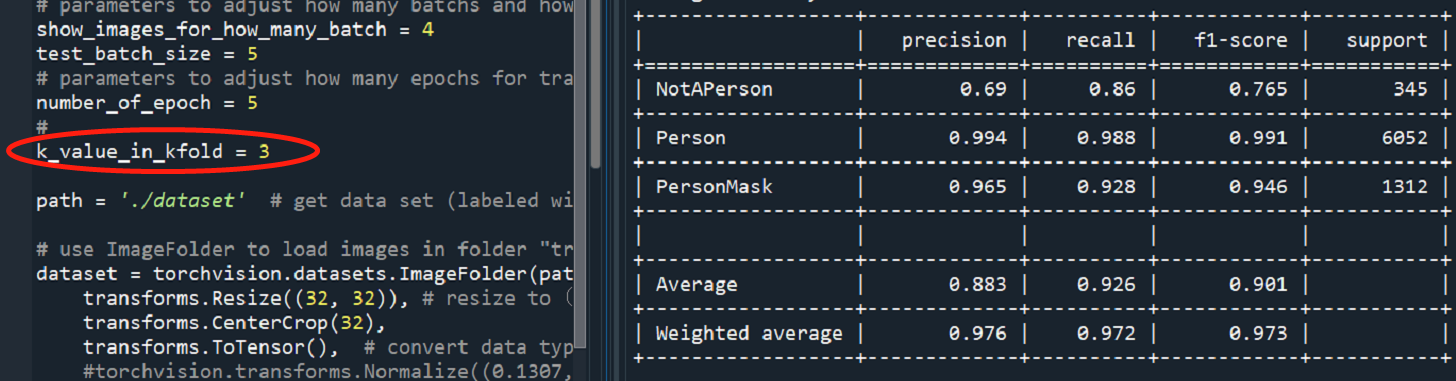
K-Fold Cross Validation helps us better use our data, and it gives us much more information about our algorithm performance. By doing cross-validation, we’re able to use all our examples both for training and for testing while evaluating our learning algorithm on examples it has never seen before. Moreover, less bias than our previous, standard evaluation method as training-set is larger when we use 10 folds. Because of larger training set reduced bias, reduced over-estimation of test-error, not as much compared to our previous, standard evaluation method.

With the advantage of K-Fold Cross Validation, all images in the dataset are equally used for training and testing, thus the performance is better.

Compare the results with our previous, standard evaluation (fixed training/test split):

Provide any insights regarding our system's performance in an analysis (i.e., why or why not there is a difference).

**3.1 Adjusted hyper-parameters during the K-fold**

****

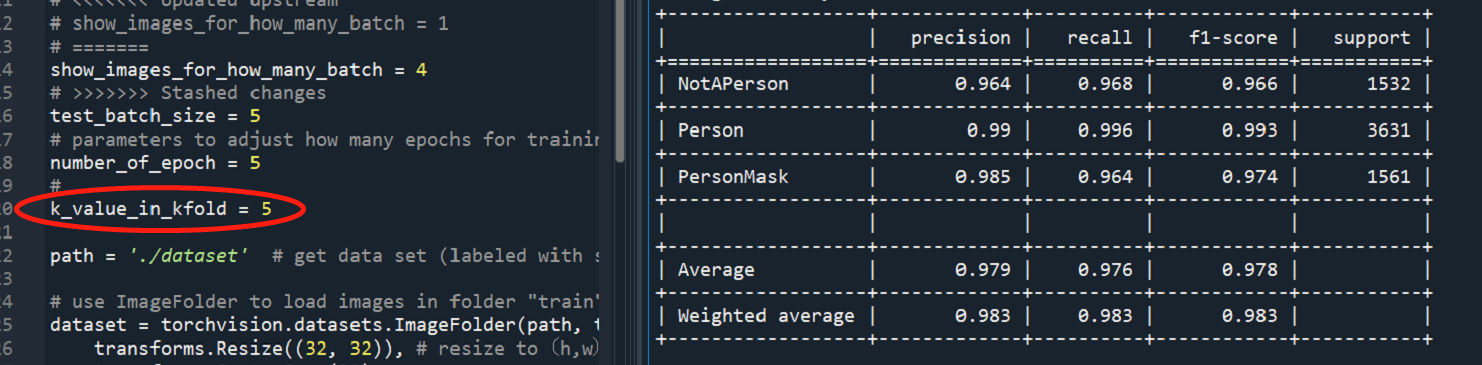
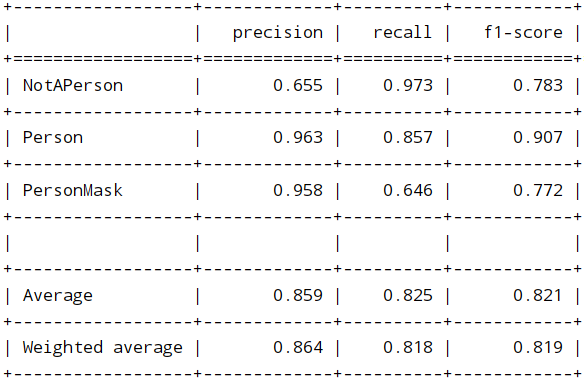
****

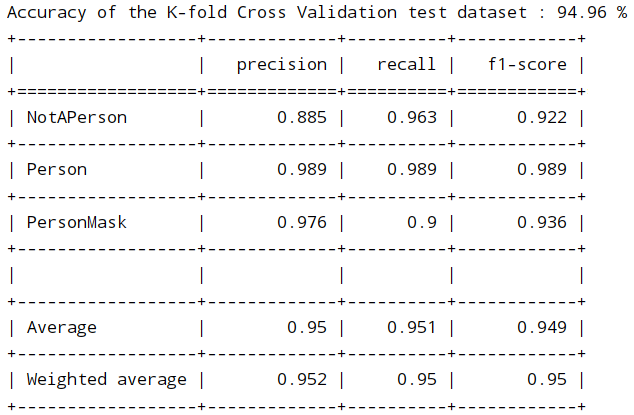
Fig.11 Adjusting K-value for Observing results

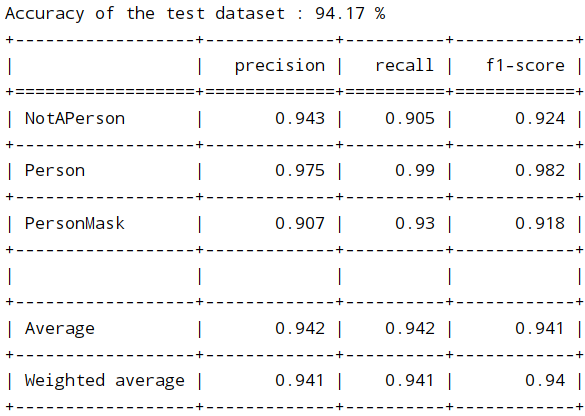
When we use 5 or more folds, there is less bias. Because of larger training set reduced bias, reduced over-estimation of test-error, not as much compared to our previous, standard evaluation method. So in our project, we finally choose 10 folds. (These two results just for observing performance when we adjust parameters, not real results.)

During the K-fold, adjusted hyper-parameters, get better result

Dataset好了运行一次，把结果贴过来（因为上面包含图片数量，现在还不能做）







*IV. Reference Section*

**For Project Part I:**

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/>

**For Project Part II:**

[1] k-fold CV, Maulik Patel, November 11, 2016, <https://rpubs.com/maulikpatel/226239>

[2] k-fold and other cross-validation techniques, [Renu Khandelwal](https://arshren.medium.com/?source=post_page-----6c03a2563f1e--------------------------------), Nov 2, 2018 <https://rpubs.com/maulikpatel/226239>

[3] MaskedFace-Net, cabani, Nov 2, 2018 <https://rpubs.com/maulikpatel/226239>

[4] MaskedFace-Net – A dataset of correctly/incorrectly masked face images in the context of COVID-19, <https://www.sciencedirect.com/science/article/abs/pii/S2352648320300362?via%3Dihub>

[5] Photo face mask, <https://www.etsy.com/market/photo_face_mask>

[6] Documentation of scikit-learn, <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html>

[7] Documentation of scikit-learn, <https://scikit-learn.org/stable/modules/cross_validation.html>

[8] Introduction to k-fold Cross-Validation, <https://machinelearningmastery.com/k-fold-cross-validation/>

[9] K-Fold Cross Validation, <https://medium.com/datadriveninvestor/k-fold-cross-validation-6b8518070833>

[10] Augmenting only the training set in K-folds cross validation, <https://stackoverflow.com/questions/57539567/augmenting-only-the-training-set-in-k-folds-cross-validation>

[11] What is the best way to apply k-fold cross validation in CNN?, <https://discuss.pytorch.org/t/what-is-the-best-way-to-apply-k-fold-cross-validation-in-cnn/15035>

[12] Documentation of pytorch: TORCHVISION.TRANSFORMS, <https://pytorch.org/docs/stable/torchvision/transforms.html>

[13] valid range for imshow with RGB data, <https://stackoverflow.com/questions/49643907/clipping-input-data-to-the-valid-range-for-imshow-with-rgb-data-0-1-for-floa>

[14] convert a float32 image to an uint8 image, <https://stackoverflow.com/questions/53235638/how-should-i-convert-a-float32-image-to-an-uint8-image/53236206>

[15] Neural Network Programming - Deep Learning with PyTorch , <https://deeplizard.com/learn/video/lu7TCu7HeYc>

[16] Documentation of pytorch: CONV2D, <https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html?highlight=padding>