Project Report on

Human Segmentation for Face Recognition

Deep Vision
Heidelberg Collaboratory for Image Processing (HCI)

By
Achita Prasertwaree
Sumet Khumphairan

Ruprecht-Karls-Universität Heidelberg
Summer Semester 2019

Project Committee:

Prof.Dr. Björn Ommer, Lecturer

Artsiom Sanakoyeu, Director of Project

Abstract

Over the past two decades, face recognition has been one of the most interesting and important research fields because its many (possible) applications in real life. Moreover, since deep learning models have been developed for semantic segmentation, we are interested in to work on these two computer vision problems.

In this project, which is part of the Deep Vision Course at Universität Heidelberg conducted by Professor, we have used a number of deep models. U-Net with ModeilNetV2 is implemented as learning background removal model. For the face recognition, we have used a residual neural network (ResNet50) as our main model.

In this report, it is separated into two main parts. First part is related to human segmentation for background removal. Finally, the face recognition will be discussed. In both paths, we trained models on sufficiently large epochs. After that, the trained models' performance has been measured. We found that they are able to performed quite reasonably well under time limit condition.

Table of Contents

PART 1: Semantic Segmentation:

Human Segmentation for Background Removal

Achita Prasertwaree

July 2019

1.1 Introduction

Semantic segmentation is one of computer vision tasks to classify all pixels in which classes they belong to, can be simply described as the pixelwise classification. Many applications of doing segmentations are also from medical image analysis such as extracting interested organs [4] or specific cells [9].

Deep learning gains the big impact on computer vision. The modification of deep neural network for each specific task is now the prerequisite. The common idea of segmentation architectures is to construct the network with the encoder and the decoder followed afterward. Downsampling the data is the main task for the encoder, so we could observe how our data captures the information. Then, the decoder does the upsampling path. Different architectures use different methods for recovering the data. In 2015, SegNet [13] provides the unpooling function, based from the idea of max pooling function, used to upsampling the data. In the same year, U-Net [9] has improved the upsampling method by using transpose convolution function. In 2016, Tiramisu [10] used the idea from DenseNet [5] in the construction of encoder. However, the payoff is the bigger number of parameters compared to SegNet and U-Net.

In this project, we consider to do background removal of human by doing semantic segmentation with the proper number of parameters for the selected model to achieve the desired accuracy. We use CelebHair [1] dataset to be the corresponding label for CelebA [6] dataset and choose U-Net to be our model with MobileNetV2 [8] as its backbone. This network can be used as the features extraction to be the pretrained model for our classification model.

1.2 Semantic Segmentation

1.2.1 Models

1.2.1.1 U-Net

U-Net is the deep convolutional network, primarily used for biomedical image segmentation, developed from University of Freiburg. Data augmentation is utilized in the prototype. Consequently, it is feasible to train the network with limited data. The model mainly consists of the contracting path and expansive path. The outstanding idea is the usage of the skip-connection, implemented as a concatenation function, between downsampling path and upsampling path to recover the information from downsampling. Moreover, we could also learn the upsampling by transpose convolution function, originated from the traditional convolution.

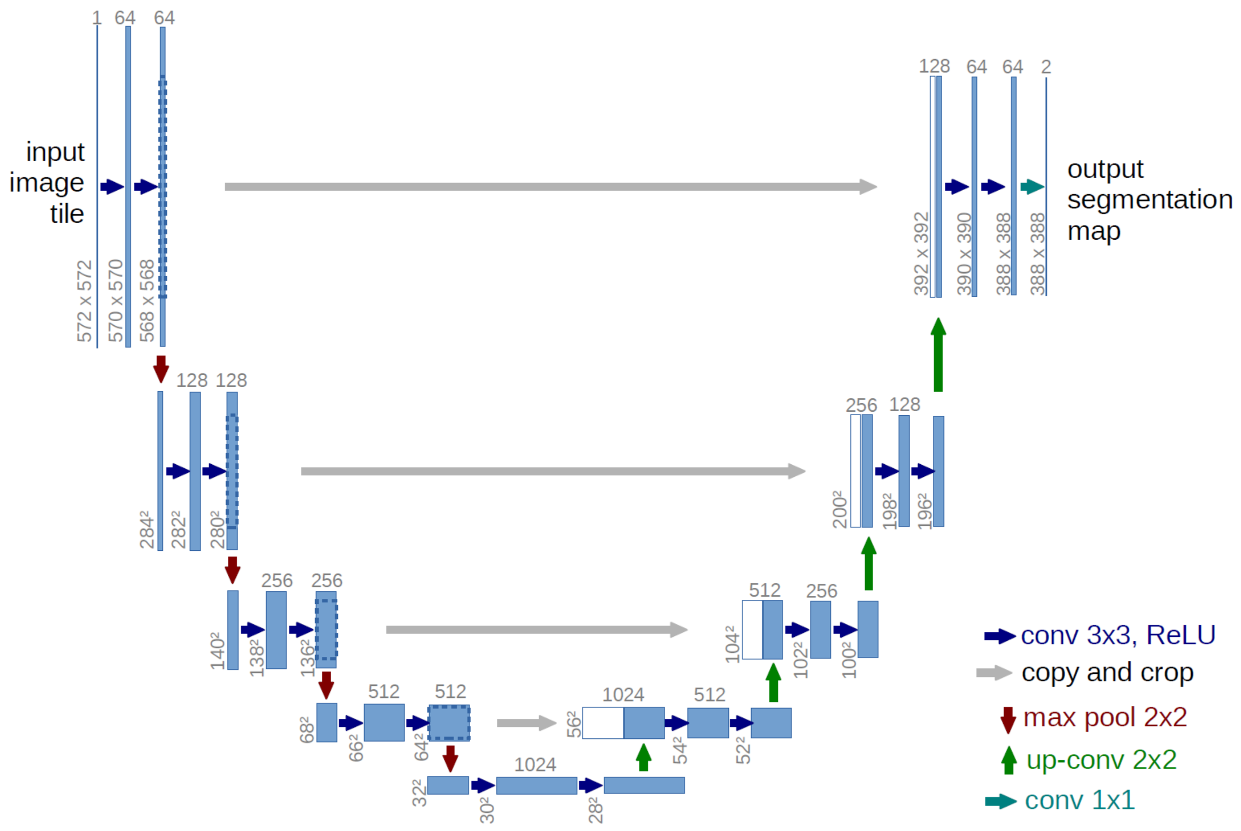

Figure 1: The diagram of U-Net Architecture

Description: The number on the top of each box stands for the number of channels and the different arrows are functions labeled and described in the lower right corner. Source: [9]

1.2.1.2 MobileNetV2

MobileNetV2 is the improved model of MobileNet from Google for mobile vision tasks, including semantic segmentation. The network is based on inverted residual structure, improving the ability to backpropagate and memory efficiency. The model is developed in a way that they could preserve as much as data from being destroyed by non-linearities and also fasten the computation. ReLU6 is one of the solutions, dealing with the robustness problem, used as the nonlinearity for this architecture.

$$ReLU6(x) \;=\; min(max(0,x),6)$$

The concept of separable depthwise convolution, which is depthwise convolution and pointwise convolution, chosen to make the computation more efficient. The depthwise convolution responds for lightweight filtering and pointwise convolution works as new features builder. They are used and arranged as the **Figure2-3**.

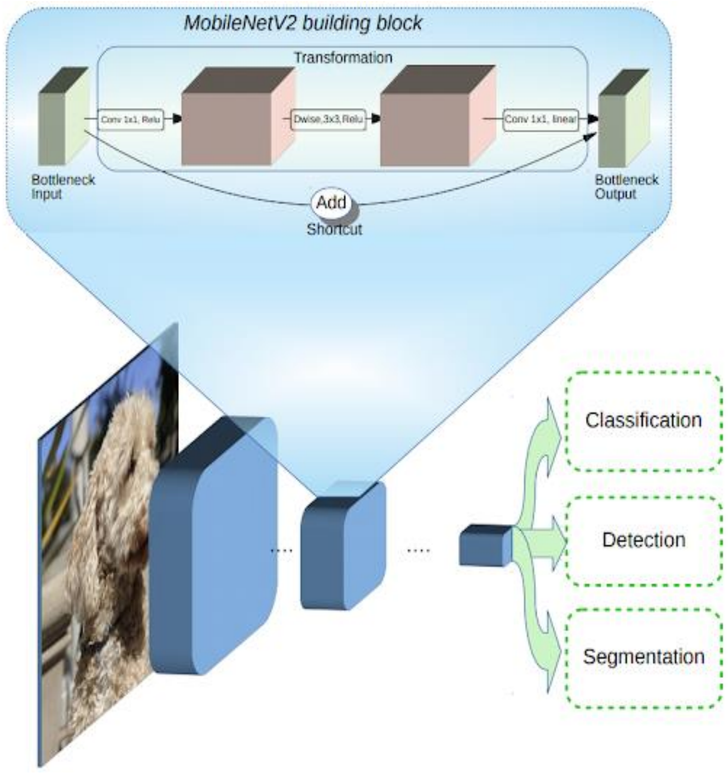

Figure 2: The diagram of MobileNetV2 Architecture

Description: The bottleneck block consists of the depthwise and pointwise convolution with the shortcut between other bottleneck layers and has inverted residual structure. Source : https://ai.googleblog.com/2018/04/mobilenetv2-next-generation-of-on.html

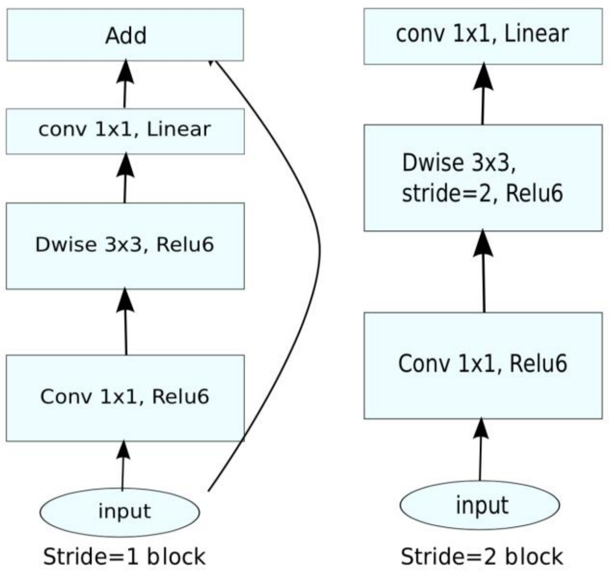

Figure 3: The inverted residual block of MobileNetV2 (Source: [8])

1.2.2 Loss function

1.2.2.1 Dice Loss

The objective function, based on dice coefficient, was firstly introduced in V-Net [4], used for volumetric medical image segmentation. The loss is designed for binary classification, especially for solving the unbalance foreground and background. It is usually processed after the softmax function has been applied. The range of this operation is between 0 and 1 and we determine the maximum value for the perfect overlap. This can be represented statistically by the set operations as follows

$$Diceloss(A,B) = 1 - \frac{2\,|A \cap B|}{|A| + |B|},$$

where A is the prediction, B is the target.

In the case of imbalanced segmentation, which we are dealing with, this function can be generalized for the multi-class classification, proposed in [2] and by applying also the Wasserstein distance, known as Earth Mover's Distance, suggested from [7].

1.2.3 Evaluation

1.2.3.1 Intersection over Union

Intersection over Union(IoU) is one of the metrics used for semantics segmentation. We choose the definition of IoU from [10]. That is, for a given class c, predictions (o_i) and targets (y_i), the metric is defined by

$$IoU(c) = \frac{\sum_i (o_i == c \wedge y_i == c)}{\sum_i (o_i == c \vee y_i == c)}$$

, where $\wedge$ is the logical and operation, $\vee$ is the logical or operation and the summation is computed over all pixels.

1.3 Experiments

1.3.1 Training

Our code is based on the repository, Human-Segmentation-PyTorch [11]. The preparation is to run the command **pip install -r requirements.txt.** Then, The training progress is processed by the command **python train.py --config config/config_UNet.json --device 0**. The hardware we selected is written in **Table 1.**

Hardware	**Model**
CPU	Intel(R) Core(TM) i7-8750H CPU @ 2.20 GHz
Memory	16 GB
GPU	NVIDIA GeForce RTX 2070 @ 8GB

Table 1: Information of the hardware

In our training, we select PyTorch as the major library. We use U-Net architecture for this task with MobileNetV2 as the backbone. The entire model contains 202 layers with the total number of parameters is 4,683,331 which is all trainable. We use CelebA dataset as the training data and CelebHair as the corresponding label. We choose the first 3,000 pictures according to the sorted label as the training set. Then, we apply the next 200 pictures for its validation set and consider the following 200 pictures to be our testing set. We do the data augmentation, which has been applied in the prototype of U-Net model, by inputting noise, flipping horizontally, rotating and random cropping, respectively. Note that we use our own designed function for the data augmentation. Next, we process our data by resizing them into 320 pixels and padding. For the training process, we use the number 16 as the batch size and train our model for 300 epochs. We use SGD as the optimizer with learning rate 1e-2. We apply cross validation with the prepared validation set. Hence, we would make sure that we will not overfit our model. We select dice loss for the loss function and mIoU(mean IoU) for the evaluation metric.

The summary of our setup is written in the **Table 2-4**. The whole setting can be found in the file **config_UNet.json.**

Variable	**Value**
Data augmentation	
noise_std	3
flip_hor	0.5
rotate	0.0
angle	10
crop_range	[0.90, 1.0]
Data preprocessing	
resize	320
padding_value	0

Table 2: The illustration how we process our data

Optimizer	
Type	SGD
learning rate	1e-2
momentum	0.9
weight decay	1e-8

Table 3: The initialization the optimizer

Lr_scheduler	
Type	StepLR
step_size	100
gamma	1.0

Table 4: The setup for the learning rate scheduler

We show the sample of the input data with several transformations in **Figure 4** and also plot the loss function of training set and its corresponding validation set. With TensorBoard, the results are shown in **Figure 5-6** respectively.

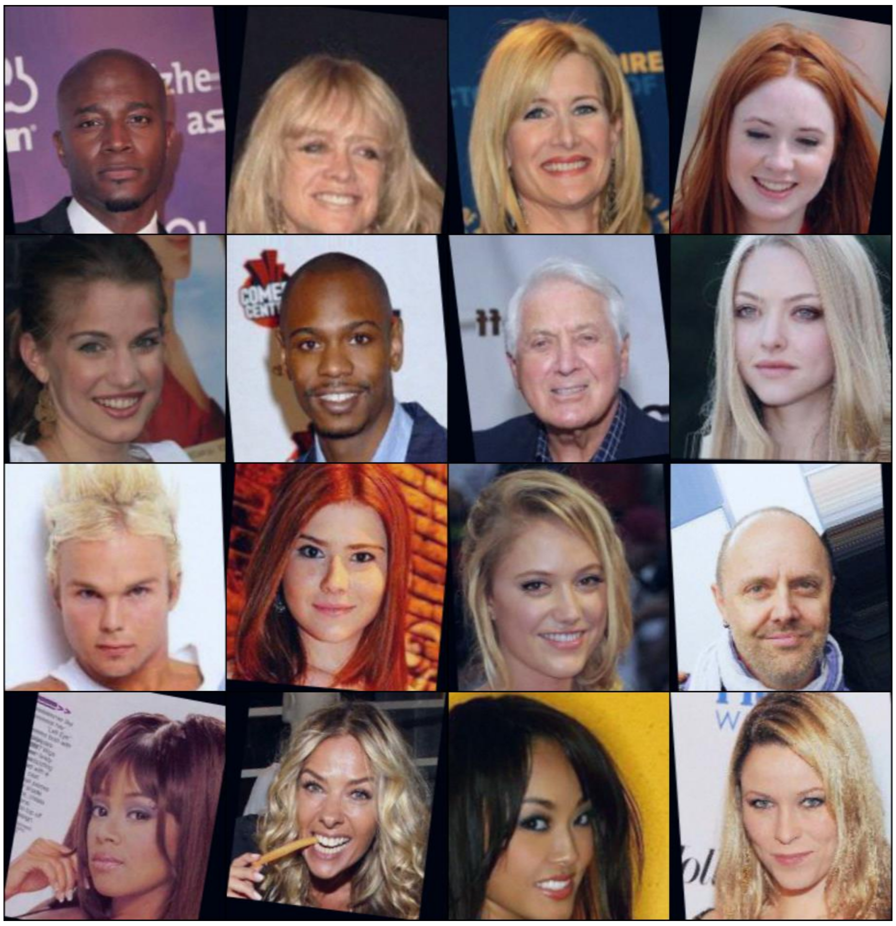

Figure 4: The sampling with applied transformation

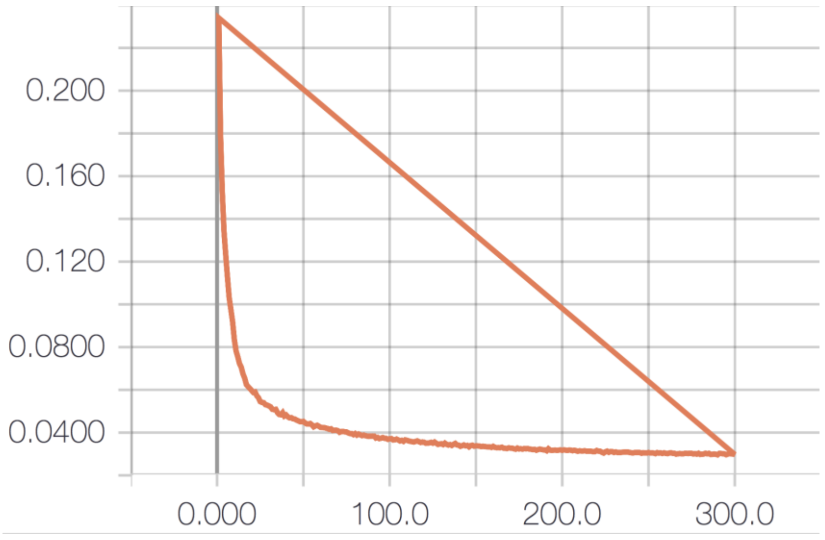

Figure 5: The training loss

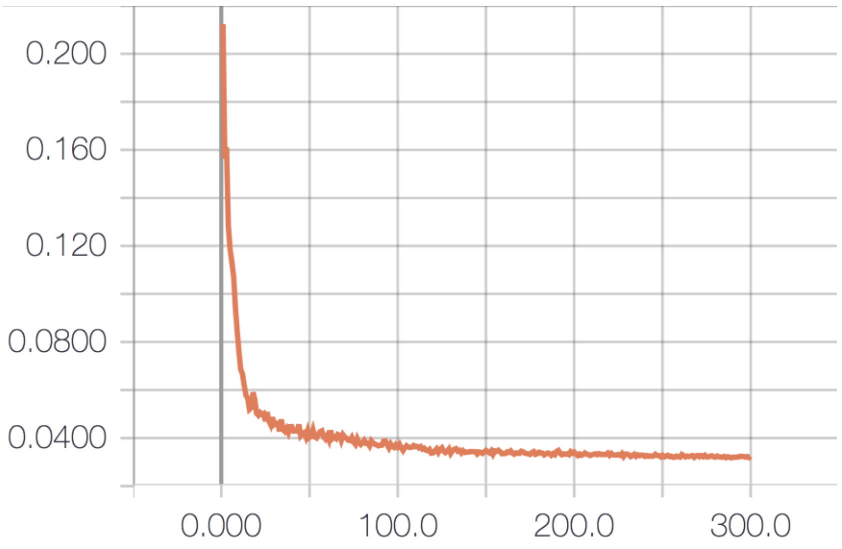

Figure 6: The validation loss

1.3.2 Testing

We obtain the satisfying result is that our model achieve the mIoU accuracy of 98.90%. We sample the picture which is the collection off the first ten pictures from the test set. The example is illustrated in **Figure 7**

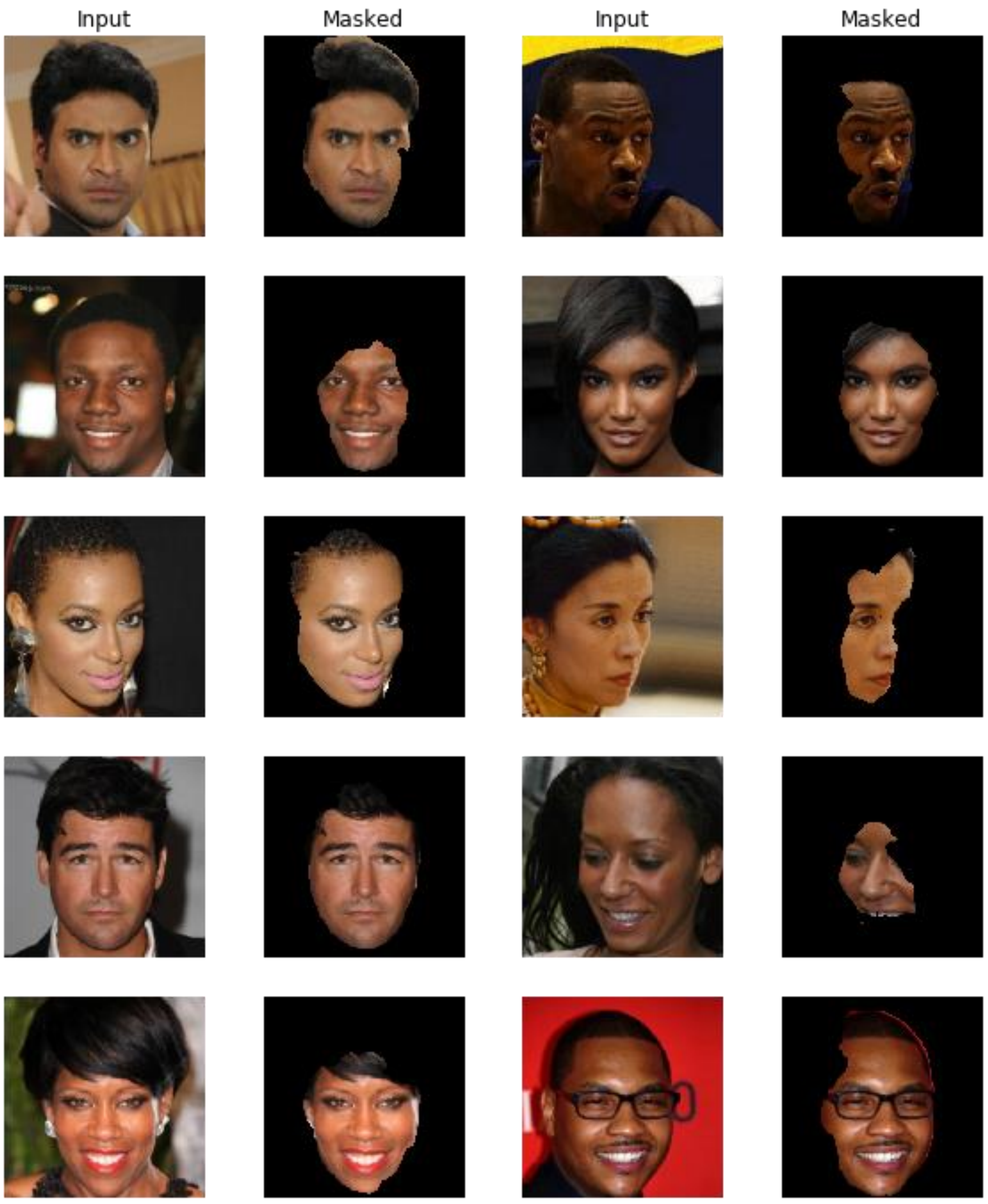

Figure 7: The illustration of the samples from our test set.

We tried also the partner's dataset to make sure whether the model works well or not. Unfortunately, the dataset is not provided for the segmentation task. The result is illustrated in **Figure 8**.

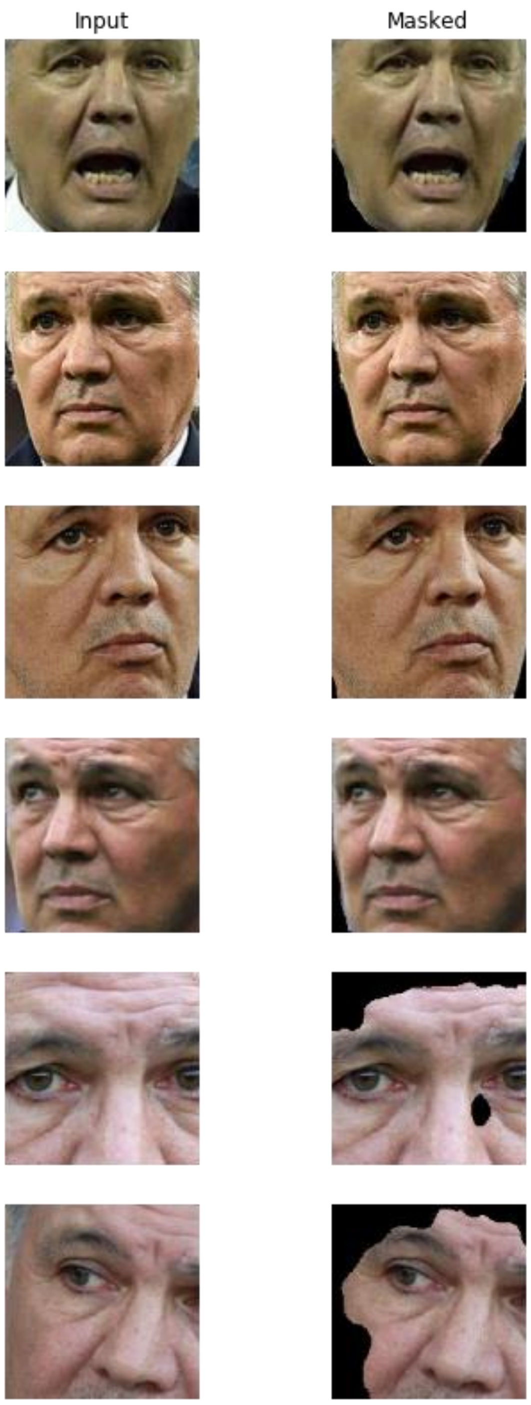

Figure 8: The illustration of the samples from our partner's dataset.

1.3.3 Difficulties

The neural network model for semantic segmentation task is usually constructed with deeper architecture compared to other computer vision tasks. Selecting the proper network could shape the better performance in terms of accuracy and computation cost.

Formerly, we select COCO [12] dataset from Microsoft for this task. The dataset contains more than 200,000 labeled pictures, generally used for the segmentation tasks. The annotation is well-written and sophisticated. However, the dataloader which can be found is the only one provided from PyTorch, **torchvision.datasets.CocoDetection.** By using the mentioned dataloader, the result is that we could not process any transformation for the corresponding label, especially cropping or resizing for inputting to our model. Moreover, in our project, our consideration is to determine human segmentation. However, from working with COCO dataset, we confront also the problem about labelling and is illustrated in **Figure 9-10**.

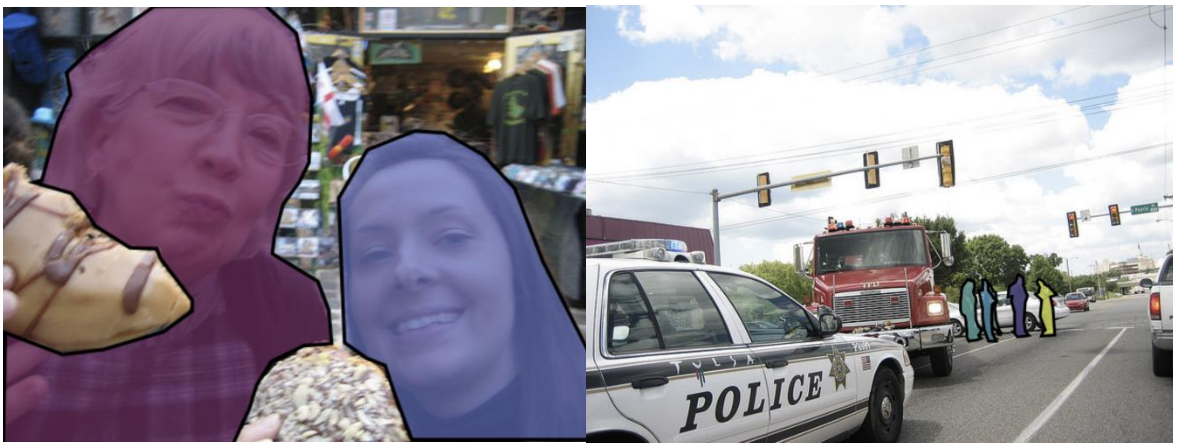

Figure 9: The comparison of two samples from COCO dataset.

Description: The left picture is appropriate of doing background removal for face detection, which the inputs for this task contain mostly human faces similarly as the picture. The right picture will be regarded as the undesired picture for our segmentation task.

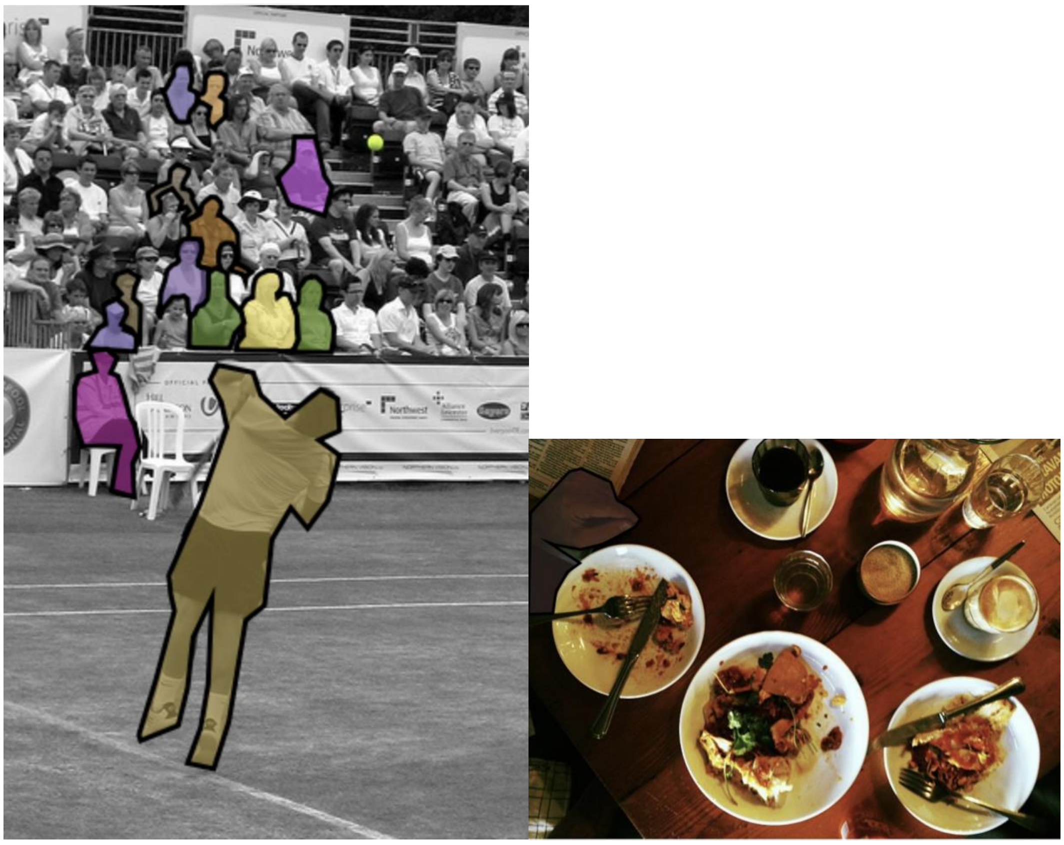

Figure 10: Two bad samples from COCO dataset.

Description: The left picture contains too many people for our background removal task. Moreover, the label does not cover for all people in the picture. For the right picture, there is no face in the picture containing human as the label.

Thus, our strategy is to identify the suitable repository first, so that we could decide the desired dataset. Nonetheless, we couldn't discover the dataloader for the datasets which the creator claimed they used in repository. Moreover, one of the datasets is written for MATLAB. Hence, our conclusion comes to CelebHair dataset, a corresponding label of CelebA dataset for doing hair and face segmentation. The same problem happens is that the dataloader for CelebHair could not be found even from the repository https://github.com/YBIGTA/pytorch-hair-segmentation which is from the authors of CelebHair. The advantage of CelebA and CelebHair is the datasets are the collections of pictures in .jpg, .png and .bmp files, so for us, who are using only PyTorch, we could resolve this problem easier than the files written in .txt or .mat files. We also considered to use CelebAMask-HQ [3] dataset, which contains 30,000 labels and also 19 classes of labels such as nose, eyes, ears, mouth. However, it is the corresponding dataset for the post-processed CelebA dataset. In our task, we consider that our dataset should be as much as realistic, so we consider that this dataset is not appropriate.

At the beginning, Tiramisu is the chosen model because of complicated network and the outstanding result compared to U-Net provided from the author of the following link https://towardsdatascience.com/background-removal-with-deep-learning-c4f2104b3157 who are doing the background removal task with also the guidance from the repository https://meetshah1995.github.io/semantic-segmentation/deep-learning/pytorch/visdom/2017/06/01/semantic-segmentation-over-the-years.html, the history of semantic segmentation using convolutional neural network. However, we could not find the repository of Tiramisu using for our specific task. The next intention is that we would like to cover some techniques applying in Tiramisu such as the Dense Block from DenseNet network to build the new neural network from U-Net architecture. However, we consider that this is time consuming since the repository we used is written in a professional script. That is we have to spend our time also for learning the written files from the repository. Moreover, my computer has no CUDA supported GPUs. Therefore, due to the amount of the dataset and the performance of my computer, we decided to use U-Net model, as in the paper mentioned that they could manage the problem of small dataset by using data augmentation. Furthermore, the result is that the number of parameters of our model is reduced, so is the computation time.

However, by the computer written in **Table 5**, the computation time costs approximately 13 hours per epoch. We decided to use my partner's platform. The result is that for each epoch, it spends only 3 minutes and this makes the better result of background removal.

Hardware	**Model**
CPU	2.3 GHz Intel Core i5
Memory	8 GB 2133 MHz LPDDR3
GPU	Intel Iris Plus Graphics 655 1536 MB

Table 5: Information of the hardware

1.4 Conclusion

In this project, we have studied U-Net and MobileNetV2 architectures, basically their infrastructures, how they actually work and also the advantages of using them. The computation is very fast when we select the proper platform. The model has an outstanding performance by achieving high test accuracy even though we worked on the limited data.

1.5 Future Work

We have considered different methods for improving our model. The first thing we should do if we have more time is to combine our dataset with different sources of datasets. Successively, we could obtain the model with better proficiency, in terms that the network is not restricted to remove the background for the specific resolution but with also numerous types of noise, level of brightness, etc. For another reason we use CelebHair dataset as our corresponding label is to extract the features whether the hair label is included in the face label or not. We are also interested in the accuracy with different scaling between hair label and face label and how can this affect the accuracy of the face recognition model. Furthermore, we consider to modify the infrastructure of the architecture such as using DenseNet, applied in Tiramisu, or try to add the new backbone for our model such as ResNet, provided in the repository and compare the performance with respect to MobileNetV2. Finally, we would like to compare the converging with other loss functions and also using various evaluation metrics to compare the accuracy.

PART 2: Face Recognition with VGGFace2

Sumet Khumphairan

July 2019

2.1 Introduction to Face Recognition

We have known that face recognition has been one of the most interesting and important research fields in the past two decades [22]. The first practical example is about identification for travelling both domestically and internationally, e.g. when someone travels to other regions, face recognition helps us verify personal identity at the immigration office. It should be noted that security is not just how to encrypt the password but our face can also be substituted as the password; therefore, another indisputable example is FaceID from apple enables us to increase the security of ID verification. Regarding to fraud in banking and payment methods, this identification algorithm is an essential tool to enlarge the possibility of personal security and personal identification data. With all of these reasons, we share our interest in and mainly focus on a technique of face identification including image segmentation. In this report, we will go through general ideas and structures of face recognition, datasets and pre-processing, training procedures, experimental results, conclusion and then future works.

It should be remarked that this report focuses only on color-image-based face recognition. Given a picture taken from a digital camera, we would like to know if there is any person inside, where his/her face locates at, and who he/she is. To achieve this goal, the face recognition procedure is generally separated into three steps: Face Detection, Feature Extraction, and Face Recognition [14].

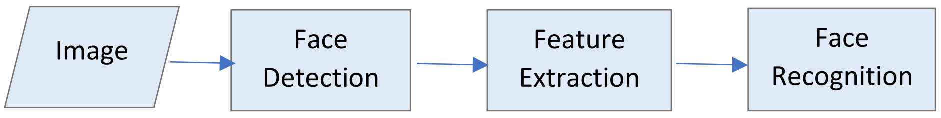

Figure 11: Configuration of a general face recognition structure

2.1.1 Face Detection

The main goal of this step is to find out whether human faces appear in a given image and also to tell us where those faces' locations are. The anticipated outputs of this step are bounding boxes containing faces in the given image. Since sometimes the bounding boxes are not suitable to be used in face recognition problem (e.g. the bounding boxes are too large), face alignment is performed to adapt the scales of these boxes. Moreover, this scheme helps us to make face recognition system more robust. In addition to serving as the pre-processing for face recognition, face detection could also be used for region-of-interest detection, video and image classification, etc.

2.1.2 Feature Extraction

After the face detection procedure, human-face bounding boxes are extracted from images. Directly implementing these boxes for face recognition have some disadvantages.

First, each box sometimes contains over 1,000 pixels which are too large to build a robust recognition system. Next, face boxes may be taken from different camera arrangements with different face expressions. To overcome these obstacles, feature extraction comes as a role for the recognition system in order to obtain information packing, dimension reduction and also noise cleaning. After this process, a face box is usually transformed into a vector with fixed dimension.

2.1.3 Face Recognition

After getting the representation for each person-face, the last step is to recognize the identities of these faces. To accomplish constructing recognition system, a face database is needed. For each identity, several images are taken and then their features are extracted and stored in the database. So, when an input image comes in, we perform face detection and feature extraction, and compare its feature to each face class stored in the database. There are two general types of face recognition. One is called identification and the other one is called verification. Face identification means given a face-image, we want the system to tell who the person is. In contrast, face verification is a problem that given a face image and a guess of the identification, we want the system to tell whether it is true or false for the guess.

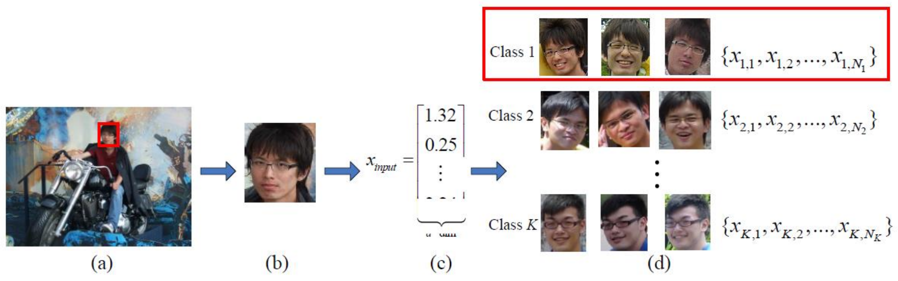
Class 1
$\{x_{1,1}, x_{1,2}, \dots, x_{1,N_1}\}$
Class 2
$\{x_{2,1}, x_{2,2}, \dots, x_{2,N_2}\}$
Class K
$\{x_{K,1}, x_{K,2}, \dots, x_{K,N_K}\}$
$x_{input} =$
1.32
0.25
(a)
(b)
(c)
(d)

*Figure 12: (a) input image, (b) the cropped image, (c) the feature vector, (d) do classification (*source: [15]*)*

2.2 Datasets and Preprocessing

In publications including daily news, we have seen and realized there has been the rapid development of deep Convolutional Neural Networks (CNNs). To feed those data-starving models, many datasets have been collected by the effort of many researchers. In general, as we know that a suitable dataset for face recognition should have the intra- and inter-class variation. The intra-class focuses on many images of one subject while the other one considers about many subjects with limited images per subject. The dataset will be more appropriate for our problem if it considers pose and age variation.

2.2.1 VGGFace2 and its Details

In this work, a large-scale face dataset named VGGFace2, has been implemented to train our model for recognizing face problem. The dataset was introduced by a visual geometry group, University of Oxford in 2017. This dataset contains 3.31 million images of identities, with an average of 362.6 images for each identity. It means the dataset has covered a large number of identities including a large number of images for each subject. In addition to the large size of dataset, images have large variations in pose, age, ethnicity and profession.

*Figure 13: A sampled dataset from VGGFace2 dataset (*source: [16]*)*

2.2.2 Review on Datasets

We shortly review relating datasets in this subpart. The first dataset for face recognition was the original Labellelled faces in the Wild (LFW), which was released in 2007. This dataset had 13,000 images with 5,749 identities. Other dataset is the CelebFaces+ which was realeased in 2007, with 10,177 subjects and 202,599 images in total. In 2015, the VGGFace dataset containing 2.6 million images covering 2,622 subjects was issued. One year later, Microsoft made the large MS-Celeb-1M dataset public with 10 million images from 100 thousands of celebrities. Although there has been an improvement in terms of the size of the dataset, these datastes still have two limitations, the number of images per identity and the label noise. Actually, the VGGFace2 dataset [17] was collected to break these restrictions.

2.2.3 Preprocessing for Training and Testing

In this subsection, we describe about how to load and preprocess the VGGFace2 dataset. First of all, we downloaded the data from the official site. When a programmer tries to solve any machine learning problem, his/her effort in solving that task goes in for the data preparation. Since PyTorch provides many tools to make loading data conveniently, we put a sample (72,283 images as training set and 4,634 images as testing set) from the large-scale dataset as the input for the class (VGG_Faces2) which is actually modified from the class (torch.utils.data.Dataset). Moreover, the method (torch.utils.data.DataLoader) for training- and testing- loader is implemented.

2.3 Model, Loss and Optimizer

To acheive high performance of a deep learning problem, we need to implement an appropriate model. Moreover, we know that when the network is too deep, the gadients of the loss function is going to be zero. Then, this results on the weights never updating its values; therefore, no learning is being performed. However, these obstacles are solved by implementing a deep residual learning model e.g. in our work we have implemented the resnet50. In this part, we will describe the residual learning model and its concerned topics.

2.3.1 Residual Learning Net

Now, we describe the main model for VGGFace2. According to the paper [18], the ResNet50 are actually modified such that it has the building block as a bottleneck design which is shown in the figure 4. In the programming part, we wrote the bottleneck framework as a class which is an input for the class of ResNet. In addition to the deep bottleneck architecture, actually they replace each 2-layer block in the 34-layer net with this 3-layer bottleneck block. As a result, this modified ResNet has 50 layers.

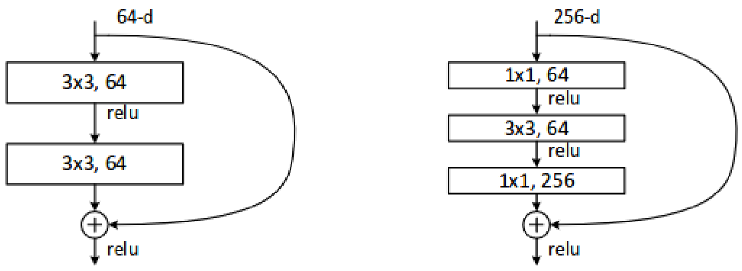

Figure 14: Basic block (left) and Bottleneck block (source: [19])

2.3.2 Weight

Since we have used a sample from VGGFace2 dataset, in order to obtain high performance we have implented the ResNet-50 with the pretrained weight provided by VGGFace2's author. In other words, the model ResNet-50 is previously trained on MS-Celeb-1M dataset, and then fined-tune on VGGFace2 dataset. For implementation, we have written a code of a model with the pretrained weight in resnet.py.

2.3.3 Loss Function

We now explain about the implemented loss function. As one of the most common tools for multi-class classification problems, the Cross-Entropy loss function is chosen to train and test our model on the VGGFace2 dataset. This criterion is provided by torch.nn as a class, torch.nn.CrossEntropyLoss() [20]. Moreover, this loss function combines nn.LogSoftmax() and nn.NLLLoss() in one single class which is convenient for users to implement. More mathematically, this criterion measures the performance of this classification model whose output is a probability value between 0 and 1. In addition, this criterion value increases as the predicted probability diverges from the actual label. The loss can be described as:

$$Loss(x, class) = -\log(\exp(x[class]) / \sum_j \exp(x[j]))$$

and the losses are averaged across observed data for each minibatch.

2.3.4 SGD Optimizer and its Initial Parameters

To assess the performance of the classification task, in addition to loss function an optimizer is needed for minimizing the previous loss function. In our work, a class in torch.optim, denoted as torch.optim.SGD, is implemented as the optimizer in demo.py. This optimizer can be considered as a stochastic approximation of gradient descent optimization. The table shows the initial setting for optimizer's parameters.

Optimizer	
Type	SGD
Learning rate	1e-1
momentum	0.9
Weight decay	0.0

Table 6: Initial parameters for the SGD optimizer

For updating the learning rate value of the optimizer every step-size epoch, we implement `torch.optim.lr_scheduler.StepLR`.

2.3.5 Precision

We have evaluated accuracy with top-N approach for N=1,5 in our work. The top-1 accuracy is the conventional accuracy. It can be described that the predicted class (the one with the highest probability) must be exactly the ground truth. For top-5 accuracy, one of the five highest probability classes must match the answer. These approaches are coded as a class AverageMeter() along with a function accuracy() in utils.py.

2.4 Experimental Results

In this section, the model ResNet50 is trained on the sample of VGGFace2 dataset in order to handle the face recognition. There are five main subtasks, that we have to consider and discuss, are detecting face in images, extracting feature for each test face-image, training ResNet50, testing the model and finally discussing about problems that we have encountered.

It should be noted that descriptions about how to run our program will be explicitly described in the appendix section of this report. This section concerns only the experimental results.

2.4.1 Face detection

Faces in the VGGFace2 are detected using the model published by [21]. The authors of VGGFace2 use the hyper-parameters recommended in that work to favor a good trade-off between precision and recall. For the face bounding box, it is extended by a factor of 0.3 to include the whole head. For VGGFace2 dataset, the authors not only provide the face images but also other relating files, e.g. bounding box files for each face image. Actually, we studied well enough about this dataset; therefore, we do not need to waste our time for doing this sub-problem.

2.4.2 Feature Extraction

It should be again noted that feature extraction acts as packaging information, reducing dimension of face images and also cleaning noise for face recognition system. In order to get a feature for each image, first go into the repository containing “demo.py” and then run the command “python demo.py extract”. There two testing datasets to be used in assessing the performance of our face recognition system, noted as the original VGGFace2 testing image and the segmented image. Therefore, we have to extract features from both testing datasets.

2.4.2.1 On the original

After running the command, a vector for each face image is saved in a folder of the corresponding identity. The size of this vector is 8,631 which are equal to the number of identities in the VGGFace2 training dataset. An example of an output feature-vector is [-0.5650346 0.6367815 1.7569996 ... -0.9601488 -1.574728 0.88301045]. This represents the feature array of the first image in our testing dataset whose image is Alejandro González Iñárritu. He is a Mexican film director.

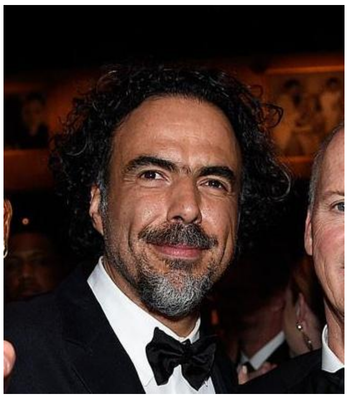

Figure 15: The first image in our testing dataset

2.4.2.2 On the segmented

We have a different feature array ([-1.2529944 -0.13535945 -0.9069282 ... 0.36901075 1.1133463 2.132247]) for the corresponding original image because the background of each image is removed by my partner’s model. The example of the segmented image corresponding to Alejandro González Iñárritu is shown as below.

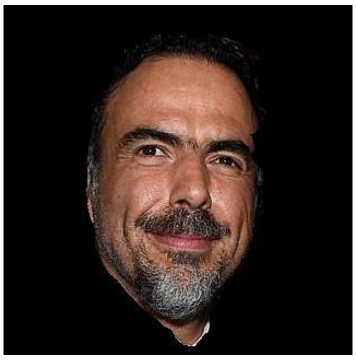

Figure 16: The segmented image of the first image in our testing set.

2.4.3 Training

In this section, training procedure for the ResNet model on the VGGFace2 dataset will be explained. Moreover, the implemented tricks during training are described. It is then followed by the training results.

2.4.3.1 Training Process

As we have learnt in lecture that the training procedure has to deal with optimizing the loss, and updating weights together with biases for the model. However, PyTorch already provides packages to handle these routine problems for training deep learning model. It should be again remarked that the Cross-Entropy loss acting as our criterion, and the SGD chosen to be the optimizer.

2.4.3.2 Implementation of Training Procedure

At this subsection, we have discussed about implementing the training scheme. The training procedure is coded as a class in the file of “trainer.py” because we realize that designing architecture of the code is very important (e.g. it is easy for us to edit the code and it is also convenient for user to understand). This file actually consists of both training and testing. However, the testing code will be explained later. Let us focus solely on the training part. The training class, denoted as Trainer(), consists of four main functions, i.e. init(), validate(), train_epoch() and train(). The method train() is actually a main function.

Before running the command, we can change the number of epochs in configurations by modifying at max_iteration. In our case, we trained on one hundred epochs which cost us around 21 hours. In other words, we have spent fifteen minutes for one epoch. This surprises us quite a lot because it is the first project in the field of the image analysis. The pseudocode for training the face recognition model is explicitly shown in the table below.

```
Pseudocode for training face recognizer
1. Set model.train()
2. for batch_idx, data in enumerate(train_loader):
    2.1) imgs, target, img_files, class_ids = data
    2.2) output = model(imgs)
    2.3) loss = criterion(output, target)
    2.4) optimizer.zero_grad()
    2.5) loss.backward()
    2.6) optimizer.step()
3. save model for later use
```

Table 7: Pseudocode for training model

2.4.3.3 Training Results

In this subpart, training outcomes will be explained step by step. We have applied the mean loss function and the top-N accuracy as our primary measures for the model’s performance. During training, the average loss for each epoch is stored as a list. Then, the loss array is saved in the selected path. Similarly, the top-1 and top-5 accuracy is also constructed as a list, and recorded in the same repository.

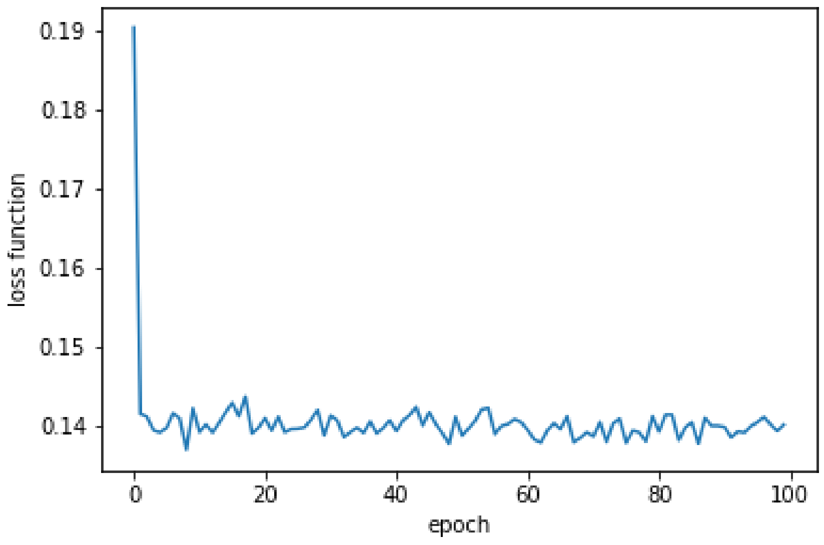

Figure 17: Training loss over 100 epochs costs 21 hours on NVIDIA GeForce RTX 2070 Laptop

First, we have initially set the learning rate (lr=1e-1). During training process, this learning rate is decayed every step_size epochs. As a result, the graph of the loss function in the figure 7 should follow the same pattern with the decayed learning rate. Furthurmore, the loss function, which is dependent on the number of iterations, decreases dramatically for the first six epochs. Then, the loss has fluctuated around 0.14 over the other 94 epochs even though the learning rate is reduced. Actually, it means we can stop the procedure early because we effectively gain no more improvments.

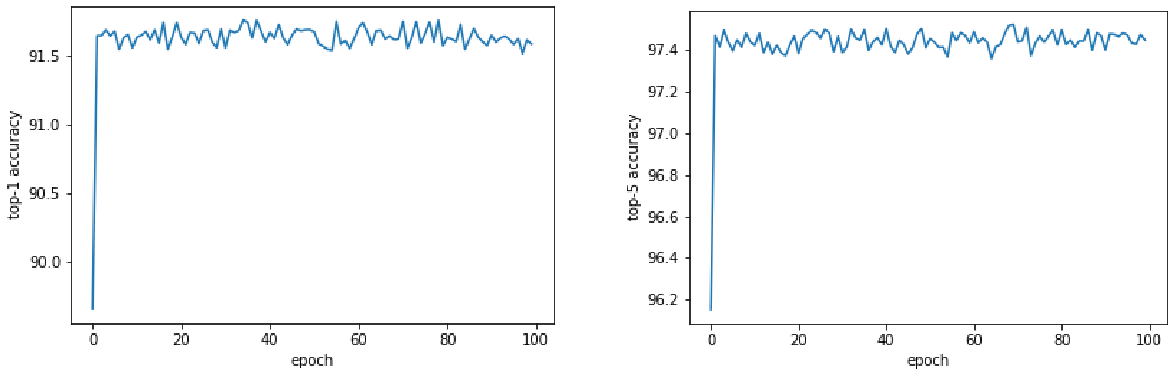

Figure 18: Top-1 accuracy (left) and Top-5 accuracy (right)

This figure shows the top-1 accuracy and top-5 accuracy are substantially increasing during the first five epochs. Then, the graph shows some form of a long plateau, around 91.5 percent for the top-1 and 97.4 for the latter. In other words, the characteristics of 100-(top-N) functions are very similar to the previous plot of loss function. Both figures can be interpreted that the ResNet performed truly well for this classification problem.

2.4.4 Testing

In this part, we will explain about the obtained results from testing the model. One of the most common measure for the recognition performance is top-N accuracy. There two test datasets for our model, a sample of VGGFace2 dataset and that sample with removed background. The top-1 and top-5 accuracy for the original sample are 90.8071 and 96.3530 percent respectively. Testing on the removed background dataset, we have got worse score of both measures, 70.9754 and 67.0911.

2.4.5 Difficulties

The challenges of doing this project consists of two main factors. One comes from the difficulties of our work and the other one is the time limit. Until now large-scale dataset has been required to feed on hungry deep learning models for both face recognition and human segmentation. VGGFace2 and CelebA are two of those big datasets and they are not provided in the PyTorch. Then, we have to study about loading data for PyTorch and do coding by ourselves. Moreover, normally effective algorithms are deep networks (U-Net for segmentation and ResNet50 for facial recognition in our case). Another disruptive factor to work as a team group under the time limit condition is that since we all have different tough subjects to study in various time; therefore, it is hard to meet in person to do project together.

2.5 Conclusion

In this work, we have implemented one of the most successful algorithms for face recognition (ResNet50). To recapitulate, applying the partner's segmentation, we have got top-1 accuracy of around 71 percent along with 67 percent for top-5 accuracy on the sampled testing dataset. This satisfies our determination for this project under the time limit of one month. This work overall is a starting point for future research projects in this chosen deep vision problem because we have learnt a lot about the current achievements in face recognition and also key factors that can significantly affect performance of our face recognition system.

2.6 Future work

Although we are restricted with the limited time, we have still managed to obtain a satisfactory final model for recognizing face on both the original sample VGGFace2 and the segmented version. In the future, we plan to do a research on any publicly available face dataset with the ResNet model. Furthermore, a group of deep learning models for example SeNet, FaceNet will be studied in terms of their architectures and benefits. The future project will also include the segmentation part. We will dive into various deep models and other face datasets which should improve the quality of segmented face images; therefore, our future face recognition system is expected to yield more satisfactory performance. Actually, we consider to about the adaptation of the existing face recognition algorithms to build a successful system.

2.7 Reference

[1] Borza, Diana and Ileni, Tudor and Darabant, Adrian. A Deep Learning Approach to Hair Segmentation and Color Extraction from Facial Images. In International Conference on Advanced Concepts for Intelligent Vision Systems, 2018

[2] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sébastien Ourselin, M. Jorge Cardoso. Generalised Dice overlap as a deep learning loss function for highly unbalanced segmentations. 2017.
[3] Cheng-Han Lee and Ziwei Liu and Lingyun Wu and Ping Luo. MaskGAN: Towards Diverse and Interactive Facial Image Manipulation. 2019
[4] Fausto Milletari, Nassir Navab, Seyed-Ahmad Ahmadi. V-Net: Fully Convolutional Neural Networks for Volumetric Medical Image Segmentation. 2016
[5] Gao Huang, Zhuang Liu, Laurens van der Maaten, Kilian Q. Weinberger. Densely Connected Convolutional Networks. 2016
[6] Liu, Ziwei and Luo, Ping and Wang, Xiaogang and Tang, Xiaoou. Deep Learning Face Attributes in the Wild. In Proceedings of International Conference on Computer Vision, 2015
[7] Lucas Fidon, Wenqi Li, Luis C. Garcia-Peraza-Herrera, Jinendra Ekanayake, Neil Kitchen, Sebastien Ourselin, Tom Vercauteren. Generalised Wasserstein Dice Score for Imbalanced Multi-class Segmentation using Holistic Convolutional Networks. 2017.
[8] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen. MobileNetV2: Inverted Residuals and Linear Bottlenecks. In The IEEE Conference on Computer Vision and Pattern Recognition, 2018.
[9] Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention, 2015
[10] Simon Jégou, Michal Drozdzal, David Vazquez, Adriana Romero, Yoshua Bengio. The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation. 2016
[11] Thuy Ng, Human-Segmentation-PyTorch, https://github.com/AntiAegis/Human-Segmentation-PyTorch
[12] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, Piotr Dollár. Microsoft COCO: Common Objects in Context. In Computer Vision and Pattern Recognition, 2014.
[13] Vijay Badrinarayanan, Alex Kendall, Roberto Cipolla. SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. 2015
[14] M. Grgic, and K. Delac, “Face recognition homepage” [online]
Available: http://www.face-rec.org/general-info. [Accessed 20 July 2019].
[15] WL. Chao, Face Recognition, [online]
Available: http://disp.ee.ntu.edu.tw. [Accessed 21 July 2019]
[16] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, A. Zisserman, “VGGFace2 webpage” [online]
Available: http://www.robots.ox.ac.uk/~vgg/data/vgg_face2/ [Accessed 15 July 2019]
[17] Q. Cao, L. Shen, W. Xie, O. M. Parkhi, A. Zisserman, “VGGFace2: A dataset for recognising face across pose and age”, International Conference on Automatic Face and Gesture Recognition, 2018.
[18] K. He, X. Zhang, Sh. Ren, J. Sun, “Deep Residual Learning for Image Recognition”, Microsoft Research, 2015.
[19] Pr. Jay, “Understanding and implementing ResNet Architecture” [online]
Available: https://medium.com/@14prakash [Accessed 1 July 2019]
[20] Torch Contributors, “torch.nn.CrossEntropyLoss” [online]
Available: https://pytorch.org/docs/stable/nn.html [Accessed 1 July 2019]

[21] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao. "Joint face detection and alignment using multitask cascaded convolutional networks". IEEE Signal Processing Letters, 23(10): 1499–1503, 2016.
[22] M. Hassaballah, S. Aly, "Face Recognition: Challenges, Achievements, and Future Directions". IET Computer Vision, 2015.

2.8 Appendix for Running Face Recognition Program

Before running our program of face recognition, please make sure the folder "vggface2" if it contanins the VGGFace2 dataset. To run the program, you can do step-by-step as explained.

1. Open an anaconda command line and then go to the repository containing Face_Recognition
2. Feature extractions, training model and testing with the default options can be done by running the followings respectively.

   2.1) python recog_face.py extract

   2.2) python recog_face.py train

   2.3) python recog_face.py test

   However, you can easily change options for each task. Just check the arguments inside the file recog_face.py, and then set new options for yourself.