Facial Expression Analysis

Team: S.T.A.R.T.

Yutong Chen 1005071869 yuton.chen@mail.utoronto.ca
Xiang Chen 1004704992 legendxiang.chen@mail.utoronto.ca
Keyi Zhang 1004705875 kya.zhang@mail.utoronto.ca

UofT CSC420

Abstract

In this project, we would like to build a python neural network that can recognize humans' emotions in pictures and cover up their faces with an emoji of that facial expression.

Computer vision has become increasingly important in a wide range of daily applications. We were interested in how a model trained by a neural network would perform on predicting human's facial expressions.

The first step was to find the appropriate dataset and transform data as needed for training purposes, since pictures could be too large if we did not downsize them. We needed a large dataset since humans observe a huge number of frames everyday. We had to let our program "see" (a.k.a. training) many pictures as well. Then, we had to find the faces in the images and resize the box containing the faces on the images to 48 by 48. Then we used CNN and ResNets to train our model. The batch size we chose was 120 and the number of epochs was 81, with an overall accuracy of 80% after training (in colab).

Conclusively, neural networks can work for determining facial expressions and provide accurate predictions with relatively high probability. We could use neural networks as a tool for machine learning in facial expression recognition.

Introduction

What is Computer Vision?

Computer vision is an interdisciplinary scientific field that deals with how computer scientists can train computers to recognize and have a full understanding of digital images or videos in order to complete some specific applications. From the perspective of artificial intelligence, it endeavors to understand and automate tasks that previously only humans can achieve.

What is our goal?

The problem we were trying to solve was mainly regarding analyzing people's facial expressions in pictures with only Python. To be exact, firstly, our Python algorithm should recognize human faces in a picture. Then, based on the current state of the face, we could analyze the organs such as the nose, mouth, eyebrows, and other muscles to determine if this face reveals any anxiety, anger, happiness, sadness, and other emotions.

When the details of the face were successfully gathered, we had to find the emoji that best matched the face shown in the picture.

What is our approach?

Our approach was divided into two main parts. The first one was to recognize the face in an image. The second was to recognize the facial expression of the face found in the previous step.

Step 1:

In the first step, we need to locate the head position. We have done pretty much research from the internet. We first tried Haar-Cascade Face detection OpenCV using Python. OpenCV provides a training method (see Cascade Classifier Training) “calledhaarcascade_frontalface_default.xml”, which is a pre-trained model, that can be imported to our Python program using the cv::CascadeClassifier::load method. However, we have found that the traditional method, Haar-Cascade Face detection, actually does not do a satisfactory job. Luckily, we have seen a paper from Cornell University, which is about the construction for MTCNN (Multi-task Cascaded Convolutional Networks). This model will consist of three inner nets and have a sequential running process. In such architecture, we can precisely locate five face landmarks and get the correct face. We have also investigated other methods such as HOG, but we think it has under performance compared with MTCNN.

Step 2:

In this step, we used the third dataset that we introduced below as our input dataset, loaded the data into our program, built the model utilizing the database, finally trained our model using CNN and ResNet.

Step 3:

In the final step, we would provide an image containing one or more faces, then we detected the faces as described in step 1. Next, we should feed the boundary containing the face into our trained model from step 2 and get the prediction results. Lastly, we put corresponding emoji on top of the face found.

Methodology

Explanation

As we were training a neural network, we had to find a large dataset, allowing our trained algorithm to be accurate. But why does our algorithm need a large dataset in the first place? A 2-year-old human child only observes 12 hours daily, equating to 4000 hours of observation in total. That is about 14,000,000 seconds, which can be approximated as 860,000,000 frames.

As a matter of fact, any 2-year-old child has gained such a large volume of visual data, which was extremely diversified. The child should be really good at recognizing objects compared to a neural network. In order to achieve the goal of recognizing facial expressions using deep learning, we had no choice but to feed our neutral network an extremely balanced, diversified, and enormous dataset for training (*Why Do Neural Networks Need so Many Training Examples to Perform?*, 2019). Thus our model can learn for itself and gives accurate predictions on a new image that has never been seen before.

Datasets (Three Collected One Selected)

There are 3 datasets that we have tried to use to train our model. The first one did not yield accurate results while the second one worked perfectly.

First Dataset

The first dataset, which did not work very well was called *FER2013 (Sharma, 2020),* available at fer2013, uploaded by Rohit Verma. The total csv file size is 287.13 MB. It is a modest-sized dataset, which should work fine for our challenges. The specifications of this dataset are as follows. It contains 34034 faces and 80% of the data was for training the other was for testing. The labels are classified in this order, 0: Angry, 1: Disgust, 2: Fear, 3: Happy, 4: Sad, 5: Surprise, 6: Neutral. The training model using this dataset has a final accuracy around 70%.

Caveats

Nonetheless, this dataset did not work well for us since the model trained using those pictures would not do a good job at predicting emotions of asians in the pictures. That is probably because the dataset contained few pictures containing asian individuals.

Second Dataset

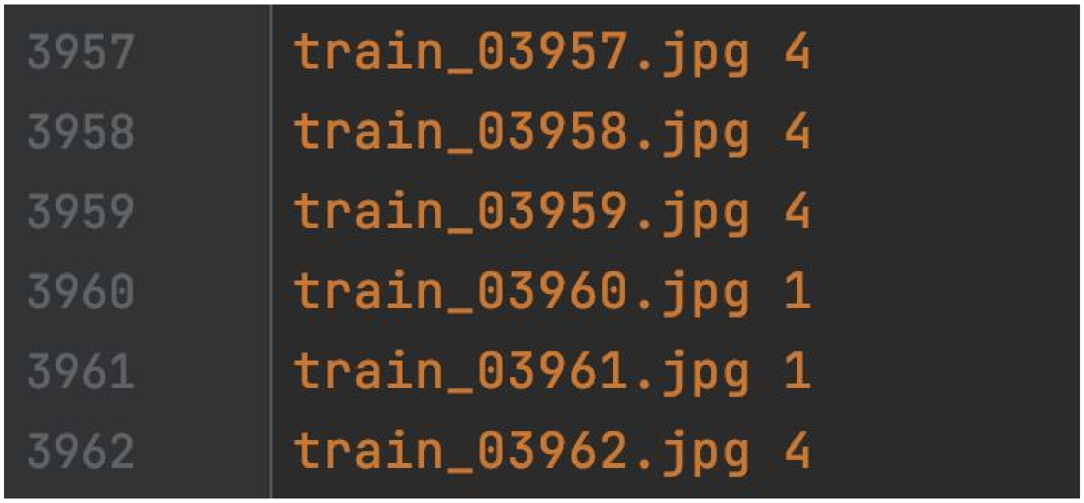
The second dataset contained 9,000 aligned images, size of 100 by 100 from the RAF-dataset. We chose 2000 of them to form a validation testing set, while the other 7000 images were training dataset. We manually transformed the selected images and it's label using the functions we implemented in FaceRec.py to raf_db.npz file. The labels are stored using one-hot encoder. The final labels are classified in this order, 0: Surprise, 1: Fear, 2: Disgust, 3: Happy, 4: Sad, 5: Angry, 6: Neutral. This model trained by ResNet has accuracy around 73%.

Third Dataset

Since the 7000 images may not be large enough to form the final dataset, we combine the dataset toronto_faces.npz from one assignment of CSC311 which has 3374 training inputs and 419 validation inputs. The labels for the dataset toronto_faces is the same as the first dataset so we convert it before we trained the model. In total, our training set contains 10374 training inputs and 2419 validation inputs. This model trained by ResNet has accuracy around 80% and we use it as our final model. The final labels are classified the same as in the second dataset.

Preparation: Transformation of Our Second Dataset

The aligned images of size 100 * 100 were stored inside a file with clearly annotated names, we also have a txt file that contains the names and labels of the face images. Some examples of labels are shown below.



```
train_03957.jpg 4
train_03958.jpg 4
train_03959.jpg 4
train_03960.jpg 1
train_03961.jpg 1
train_03962.jpg 4
```

However, we also had to extract the face of each 100 by 100 images using the MTCNN detector, once the faces were detected, we needed to resize the image to 48 by 48 which could be fed to our model for training and validation testing. The function in convert_data.py for detecting and resizing images is shown below.

```
def FaceRec(path):
    img = pyplot.imread(path)
    detector = MTCNN()
    faces = detector.detect_faces(img)
    output = resize_faces(path, faces)
    return output
```

The function in convert_data.py for resizing a face is shown below.

```
def resize_faces(path, result_list):
    data = pyplot.imread(path)
    h, w = data.shape[0], data.shape[1]
    output = np.empty((1, 48*48))
    flag = 0
    for i in range(len(result_list)):
        x1, y1, width, height = result_list[i]['box']
        x2, y2 = x1 + width, y1 + height
        x1 = max(0, x1)
        x2 = min(x2, w)
        y1 = max(0, y1)
        y2 = min(y2, h)
        if (width < w/2 or height < h/2):
            flag = 1
            break
        img = data[y1:y2, x1:x2]
        res = cv2.resize(img, dsize = (48, 48),
interpolation=cv2.INTER_CUBIC)
        gray = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
        output = gray.reshape((1, 48*48))
        break

    # For some cases(e.g. wearing glasses) the MTCNN can not detect faces,
    # or it detect something that's really small, we simply resize the
image
    if (len(result_list) == 0 or flag ==1):
        res = cv2.resize(data, dsize = (48, 48),
interpolation=cv2.INTER_CUBIC)
        gray = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
        output = gray.reshape((1, 48*48))

    return output
```

Preparation: Transformation of Our Dataset (Strength data)

In order to reduce overfitting, and generalize our data, we had to do multiple transformations on the data we had. This process is vital as it refines our data leading to better accuracy of our model when it comes to prediction.

The code for generating the transformation is shown below.

```
# using the built in function from torchvision
train_trfm = transforms.Compose(
    [
        # convert to PIL image
        transforms.ToPILImage(),
        # convert the image to grayscale
        transforms.Grayscale(num_output_channels=1),
        # randomly crop the image and add padding of 4 on the image
        transforms.RandomCrop(48, padding=4, padding_mode='reflect'),
        # randomly horizontally flip the image, probability = 0.5
        transforms.RandomHorizontalFlip(),
        # convert to tensor so we can utilize GPU
        transforms.ToTensor(),
        # normalize the pixels in the image
        transforms.Normalize((0.5), (0.5), inplace=True)
    ])
```

(Sharma, 2020)

Preparation: Read in our training & testing data

```
    npzfile = np.load("./read_images/raf_db.npz")
    npzfile1 = np.load("./read_images/toronto_face.npz")

    train_images = npzfile["inputs_train"]
    train_labels = np.argmax(npzfile["target_train"], axis=1)
    test_images = npzfile["inputs_valid"]
    test_labels = np.argmax(npzfile["target_valid"], axis=1)

    train_images1 = npzfile1["inputs_train"]
    train_labels1 = npzfile1["target_train"]
    test_images1 = npzfile1["inputs_valid"]
    test_labels1 = npzfile1["target_valid"]

    # Convert the labels so that it matches the label class in our model
    for i in range(len(train_labels1)):
        if train_labels1[i] == 0:
            train_labels1[i] = 5
        elif train_labels1[i] == 1:
            train_labels1[i] = 2
        elif train_labels1[i] == 2:
            train_labels1[i] = 1
        elif train_labels1[i] == 3:
            train_labels1[i] = 3
        elif train_labels1[i] == 4:
            train_labels1[i] = 4
        elif train_labels1[i] == 5:
            train_labels1[i] = 0
        elif train_labels1[i] == 6:
            train_labels1[i] = 6
        else:
            print("wrong train label",train_labels1[i])

    for i in range(len(test_labels1)):
        if test_labels1[i] == 0:
            test_labels1[i] = 5
        elif test_labels1[i] == 1:
            test_labels1[i] = 2
        elif test_labels1[i] == 2:
            test_labels1[i] = 1
        elif test_labels1[i] == 3:
            test_labels1[i] = 3
        elif test_labels1[i] == 4:
            test_labels1[i] = 4
        elif test_labels1[i] == 5:
            test_labels1[i] = 0
        elif test_labels1[i] == 6:
            test_labels1[i] = 6
        else:
            print("wrong test label",test_labels1[i])

    train_images = np.concatenate((train_images, train_images1))
    train_labels = np.concatenate((train_labels, train_labels1))
    test_images = np.concatenate((test_images, test_images1))
    test_labels = np.concatenate((test_labels, test_labels1))
```

Batch

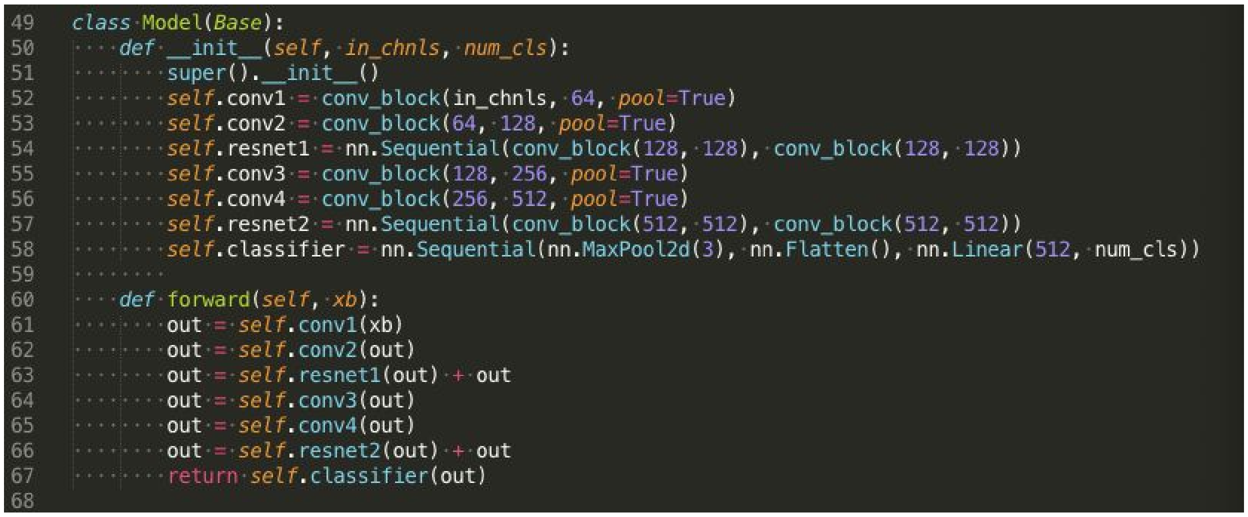
Let us see an example image of our training data in the batches of size 120.

Preparation: Building our model to be trained

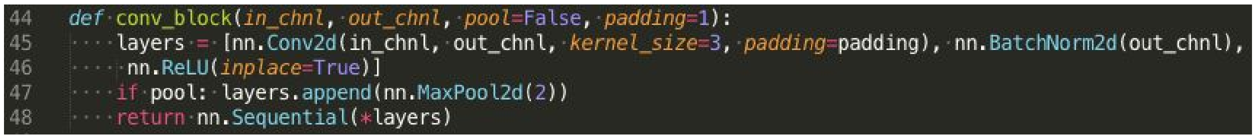
We chose to use 6 layers for our neural network, with 4 being convolution neural networks and the other 2 being ResNet using Residual Networks. Below is the architecture of ResNet9 that we used.

conv1.0
conv1.1
conv2.0
conv2.1
resnet1.0.0
resnet1.0.1
resnet1.1.0
resnet1.1.1
conv3.0
conv3.1
conv4.0
conv4.1
resnet2.0.0
resnet2.0.1
resnet2.1.0
resnet2.1.1
classifier.2

Our network should improve accuracy over time. The last layer would be connected to 7 nodes, each representing a class (e.g. a specific facial expression), ranging from -1 to 1, indicating the probability of this class. The code for defining the model is shown below.



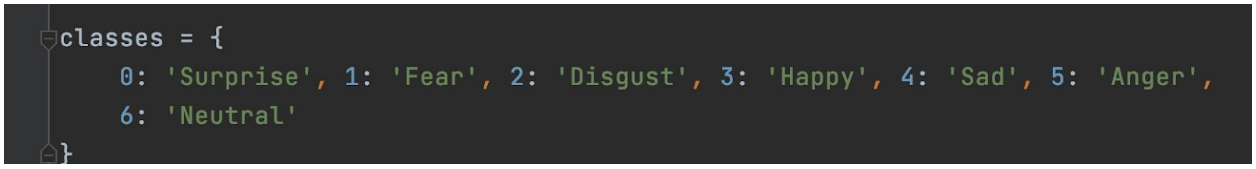
```python
class Model(Base):
    def __init__(self, in_chnls, num_cls):
        super().__init__()
        self.conv1 = conv_block(in_chnls, 64, pool=True)
        self.conv2 = conv_block(64, 128, pool=True)
        self.resnet1 = nn.Sequential(conv_block(128, 128), conv_block(128, 128))
        self.conv3 = conv_block(128, 256, pool=True)
        self.conv4 = conv_block(256, 512, pool=True)
        self.resnet2 = nn.Sequential(conv_block(512, 512), conv_block(512, 512))
        self.classifier = nn.Sequential(nn.MaxPool2d(3), nn.Flatten(), nn.Linear(512, num_cls))
        
    def forward(self, xb):
        out = self.conv1(xb)
        out = self.conv2(out)
        out = self.resnet1(out) + out
        out = self.conv3(out)
        out = self.conv4(out)
        out = self.resnet2(out) + out
        return self.classifier(out)

```



```python
def conv_block(in_chnl, out_chnl, pool=False, padding=1):
    layers = [nn.Conv2d(in_chnl, out_chnl, kernel_size=3, padding=padding), nn.BatchNorm2d(out_chnl),
     nn.ReLU(inplace=True)]
    if pool: layers.append(nn.MaxPool2d(2))
    return nn.Sequential(*layers)
```

(Sharma, 2020)

In our model, each number represents an emotion.



```python
classes = {
    0: 'Surprise', 1: 'Fear', 2: 'Disgust', 3: 'Happy', 4: 'Sad', 5: 'Anger',
    6: 'Neutral'
}
```

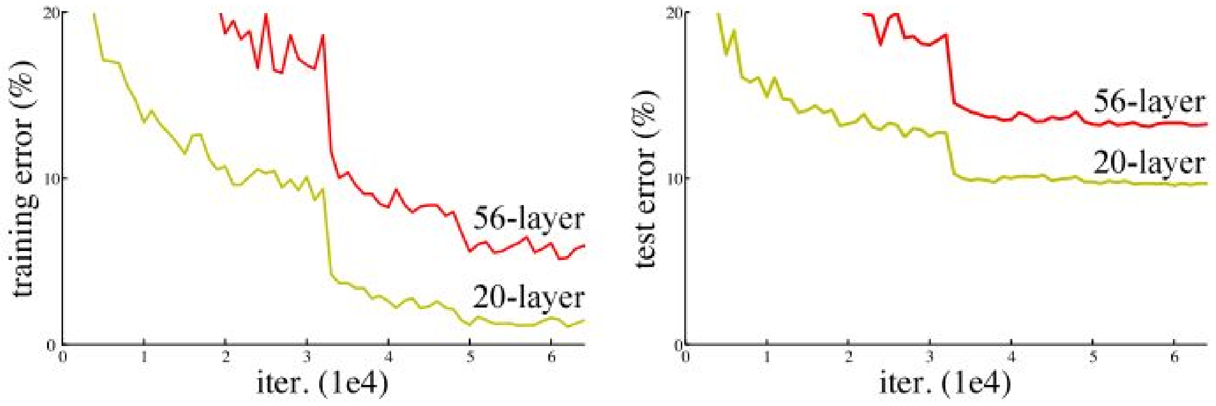
We also needed a base classifier which would operate and serve as a device to store information like validation data loss and accuracy for each epoch.

The reason for us to choose convolutional neural networks over standard deep-learning algorithms comes as follows. First thing we would like to point out is that convolutional networks are more efficient, reducing the number of parameters and layers needed. They really utilize the concept of features and locality of digital images (Mishra, 2019). By contrast, in traditional neural networks, each neural does not take their neighbors into consideration. That means a traditional neural network would require more layers in order to be adequately trained. Convolutional Neural Networks find the underlying patterns hidden in the images, consider the images as a whole (Chatterjee, 2020). Convolutional Neural Networks can also detect latent features that we do not observe initially. Furthermore, one bonus of using Convolutional Neural Networks is we could take advantage of torch and it can be accelerated by GPUs. That was because convolutional operations involve a huge number of matrix multiplications, at which GPUs are extremely efficient. Specifically, training our model on Google Colab using the GPU provided, it only took 20 mins compared with training with our CPU, which takes many hours. Moreover, CNNs are effective at reducing the number of pixels at each layer without losing much information about the images. That process is called dimensionality reduction. CNN takes advantage of the hierarchical structure in data and can assemble more complex patterns using smaller and simpler patterns.

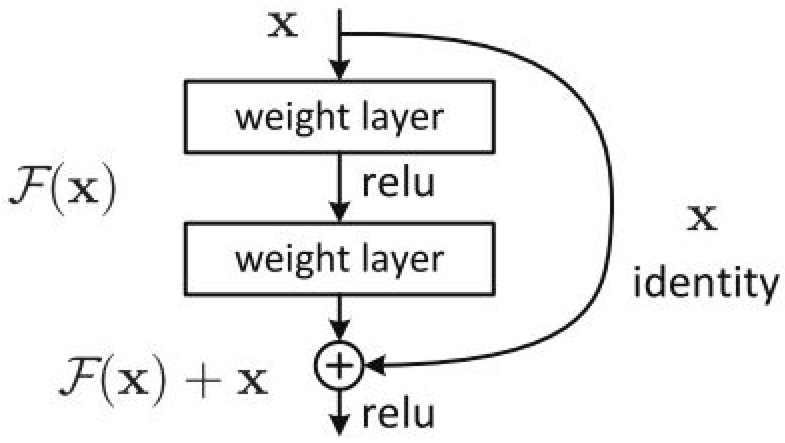
In a nutshell, CNN models work like a funnel, with each layer might reduce the number of pixels, and finally yields the probability distribution of classification.

We also implemented ResNet into our CNN model. Some benefits are brought when using ResNet 9. We could implement the entire neural network with only one layer, in accordance with the universal approximation theorem, if we had as much capacity as we wanted.

However, this imagined layer would be enormous and would be undermined by the overfitting of data. To overcome this issue, we had to add more layers. But there is always a limit on how many layers we can add before we gain worse performance when more layers are added. This issuer is caused by the vanishing gradient problem. As we back-propagate to previous layers, huge numbers of multiplication would lead the gradient to be extremely small, resulting in neural networks with many layers to have downgrading performance as the number of layers increases. Below is a picture from Vincent Fung, a machine learning engineer, illustrating this effect. The citation is in the reference section (Fung, 2018).



To solve this issue, use something called a residual block, it looks like this.

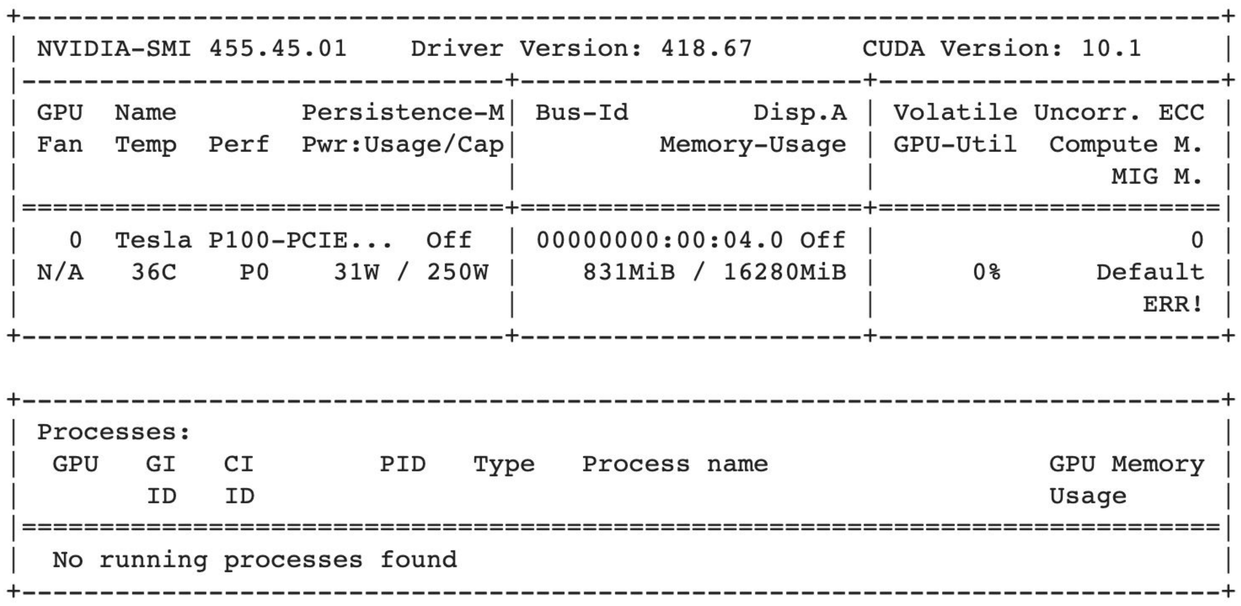

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

The method is simple, we can add the result from the previous layer to the next layer, allowing the algorithm to perform just as well. Thus deeper models would not produce a training error. However, this approach showed that the network with 1202 layers had worse performance than its counterpart. However, for our project, since we had only fewer than 10 layers, our ResNet should work fine (Fung, 2018).

Utilizing GPUs

We could not finish our project without GPU's on Google Collab since none of us had access to a Nvidia GPU. It would take a couple hundred hours to train our model with the CPUs on our old MacBooks. As we mentioned earlier, GPUs are very efficient at training CNNs. Here are some reasons for that. First, GPUs are good at matrix multiplication. That is because CPUs are very quick at calculations but the amount of information CPUs can process is way less than that of GPUs. The bandwidth of data transfer between RAM for GPU is about 10 times as fast as CPUs'. Also, GPUs are good at simple calculations since they tend to have hundreds of cores where CPUs usually have around 10 cores (Shaikh, 2019).

We chose to use the GPU on Google Colab, which was a Tesla P100 GPU.

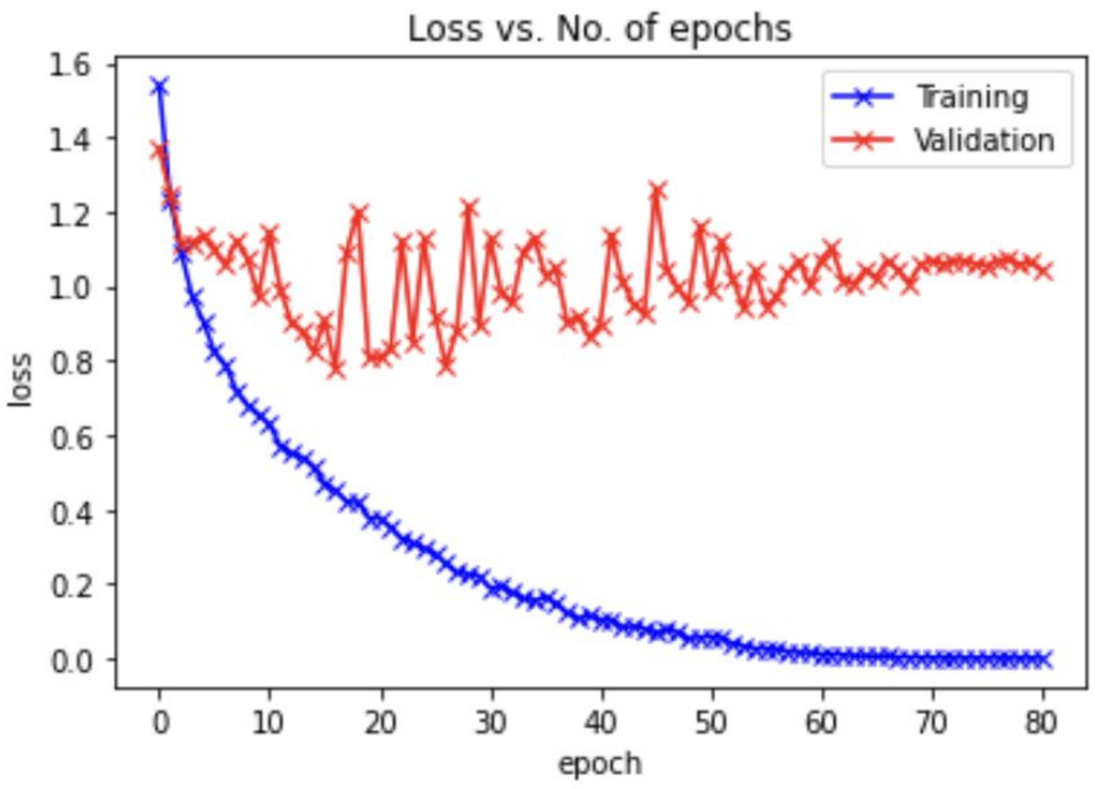


```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 455.45.01    Driver Version: 418.67       CUDA Version: 10.1     |
|-------------------------------+----------------------+----------------------+
| GPU  Name        Persistence-M| Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf  Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                               |                      |               MIG M. |
|===============================+======================+======================|
|   0  Tesla P100-PCIE...  Off  | 00000000:00:04.0 Off |                    0 |
| N/A   36C    P0    31W / 250W |    831MiB / 16280MiB |      0%      Default |
|                               |                      |                 ERR! |
+-------------------------------+----------------------+----------------------+

+-----------------------------------------------------------------------------+
| Processes:                                                                  |
|  GPU   GI   CI        PID   Type   Process name                  GPU Memory |
|        ID   ID                                                   Usage      |
|=============================================================================|
|  No running processes found                                                 |
+-----------------------------------------------------------------------------+
```

It gave us the opportunity to use cuda, which is a parallel computing platform and application programming interface model created by Nvidia. It provides a wide variety of deep learning libraries and GPU-accelerated libraries we could use. Thus we chose to use the Tesla P100 GPU on Google Colab.

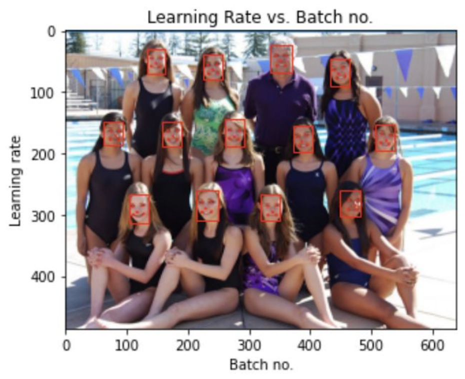
Training Our Model

The training and validation loss are plotted as functions of epochs below. The batch size we chose was 120. The number of epochs we chose was 80. Since the loss stabilized after the number of epochs reached around 80.

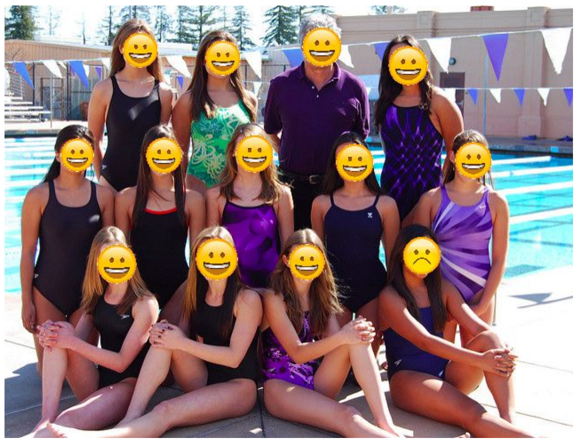


Results

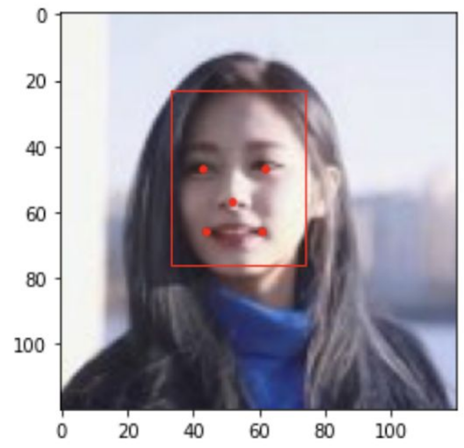
After training our model, we tested a picture containing 13 pictures, with 12 faces being happy, 1 being unhappy. We used our trained model, file 9.pth**,** to predict their emotions, we got basically 13/13. The probability for the emotion was also pretty high for our prediction of each face. The results are shown below.



```
Happy
Happy 0.9955671429634094
Happy
Happy 0.9122448563575745
Happy
Happy 0.997506320476532
Happy
Happy 0.991341769695282
Happy
Happy 0.7710886597633362
Happy
Happy 0.8805151581764221
Happy
Happy 0.9130634665489197
Happy
Happy 0.9971062541007996
Happy
Happy 0.99741530418396
Sad
Sad 0.41097912192344666
Happy
Happy 0.9988172650337219
Happy
Happy 0.8933581709861755
Happy
Happy 0.9997572302818298
```

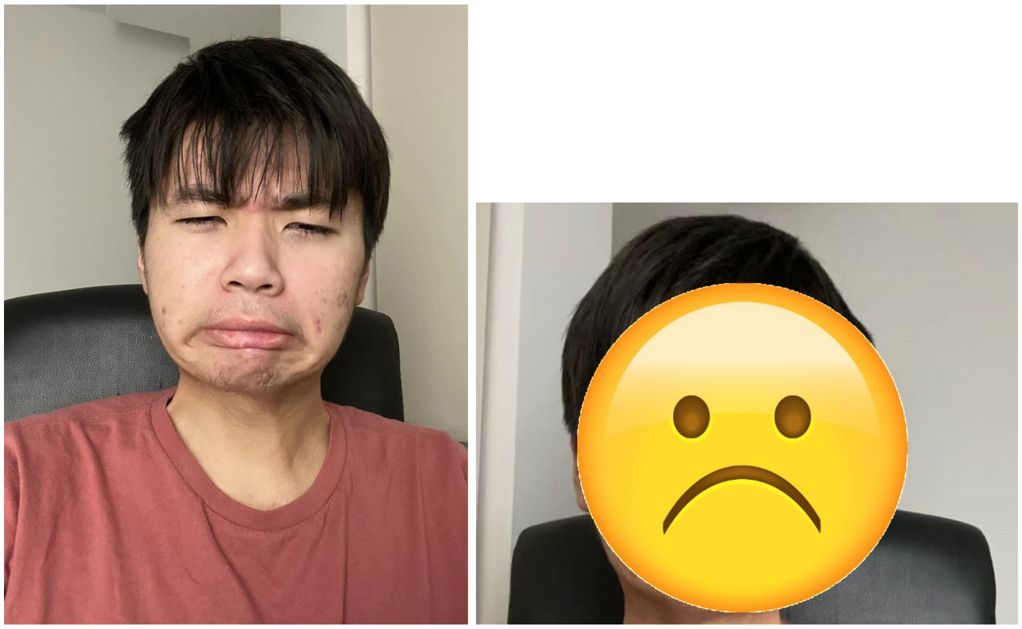


Another example we tried was a picture of a happy girl. Our model successfully predicted her mention which was happy, with a relatively high probability of roughluy 89%.

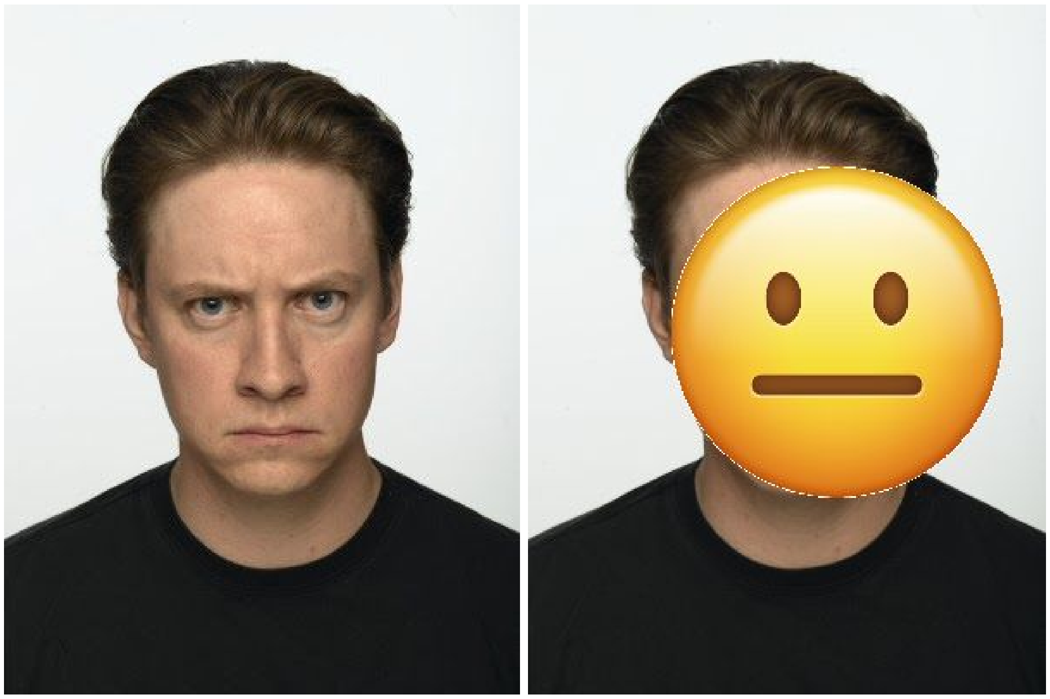


```
Happy
Happy 0.8940987586975098
```

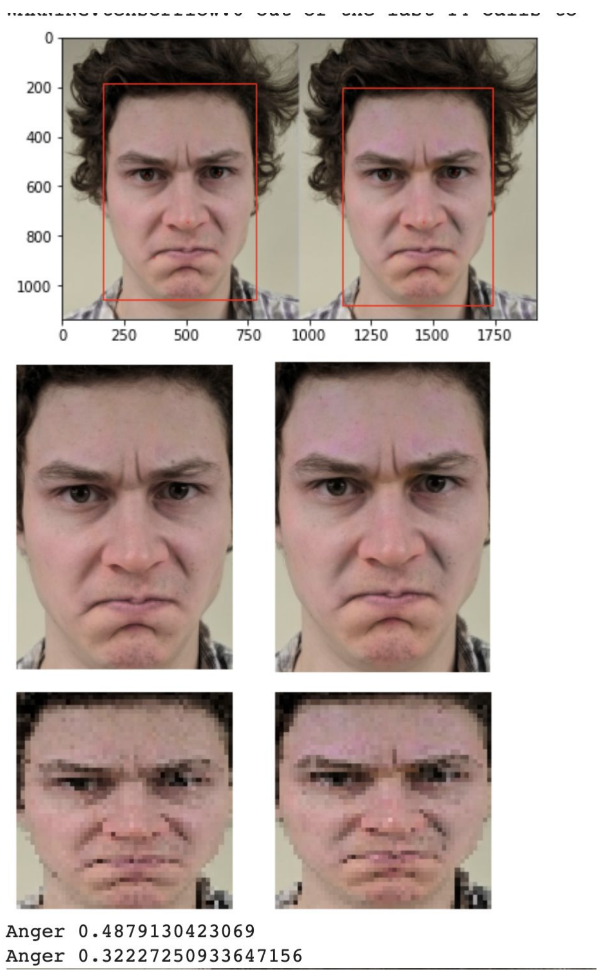
Below is a sad face of Xiang Chen with a prediction of sad 0.8076507449150085.



Below is a face that is kind of ambiguous between neural and anger. Our model predicts it as neutral with probability 0.5126513838768005 which is not very high as we expected.



Below is an example of anger.

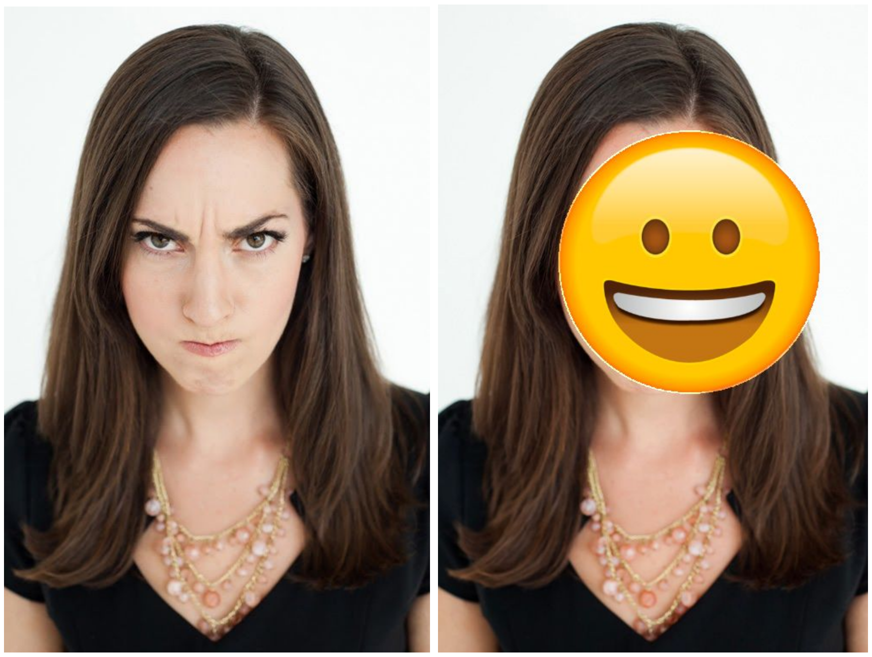


```
Anger 0.4879130423069
Anger 0.32227250933647156
```

Failures

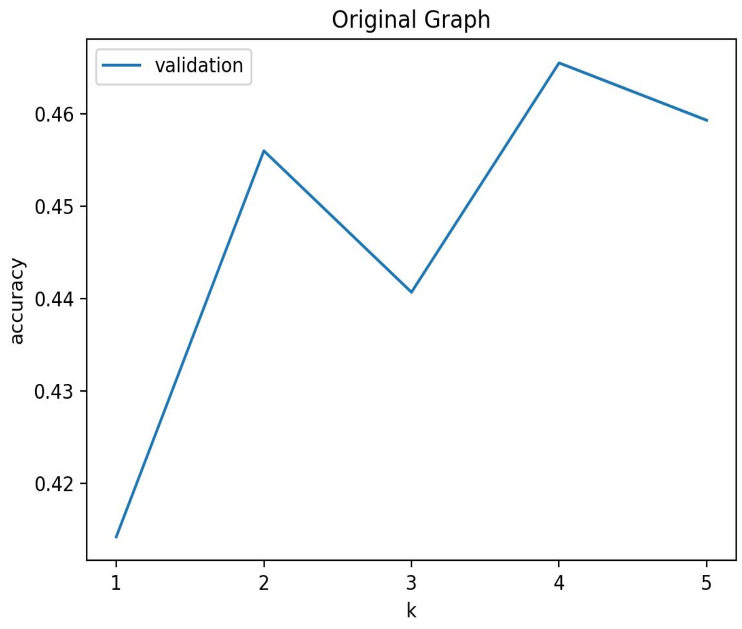
A very obvious problem about our model is that it can not recognize angry facial expressions very well and it may classify some faces with teeth showing as happy. From the successful results of anger above we see that even the successful result does not have a relatively high probability of anger which means that the prediction is not that “confident”. This is probably because our training input does not have a large number of angry faces and thus our model did not learn very well with that feature.

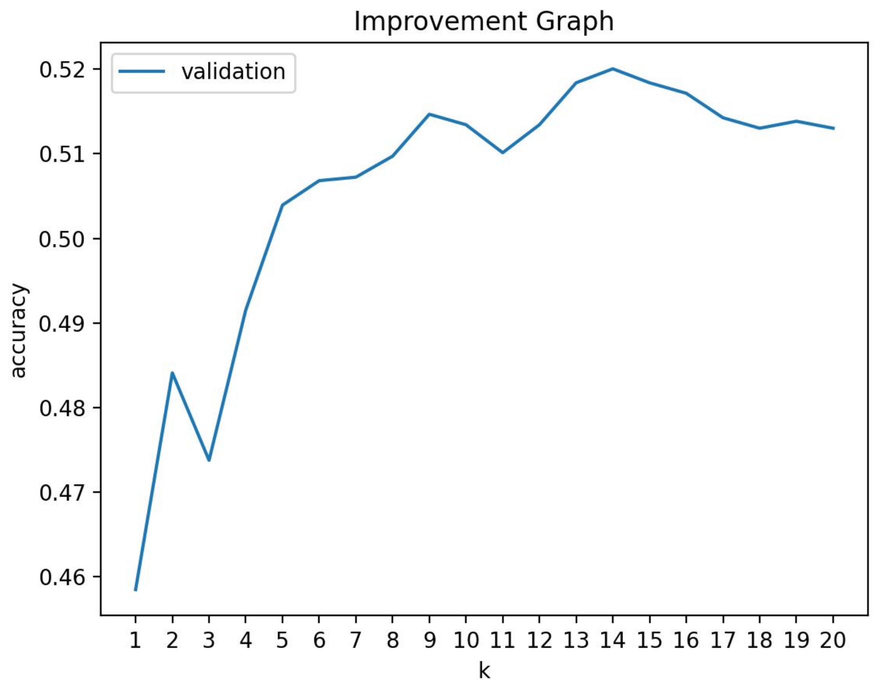
One failure example is shown above. This facial expression is definitely not happy but it has a significantly high prediction of ‘happy’ of 0.976340115070343.



KNN Implementation

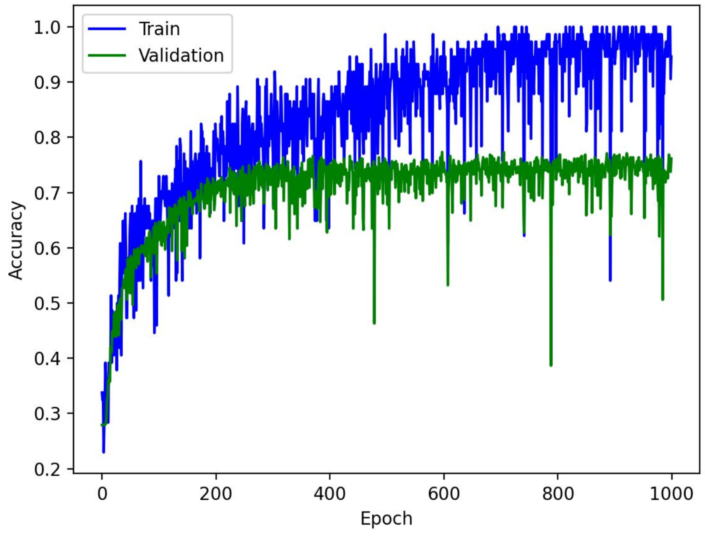
We also train one K-nearest neighbours model. In our implementation, we first use Euclidean Metric to find the nearest neighbours. We find it takes extremely long time for predicting and we believe it is because we have a designed matrix with extremely high dimensions(48 * 48), and also the accuracy of the validation set is really low. We think when we have a hyper dimension designed matrix, the Euclidean Distance is roughly the same and really hard to compute. After we investigate the model of K-nearest neighbours, we decide to use cosine metric to calculate the angular distance. We find it actually performs far better than Euclidean Metric for aspects of both runtime and accuracy. Here are two graphs for the relationship between k value and the validation accuracy. We only test k={1,5} for Euclidean Metric and k={1,20} for cosine metric.

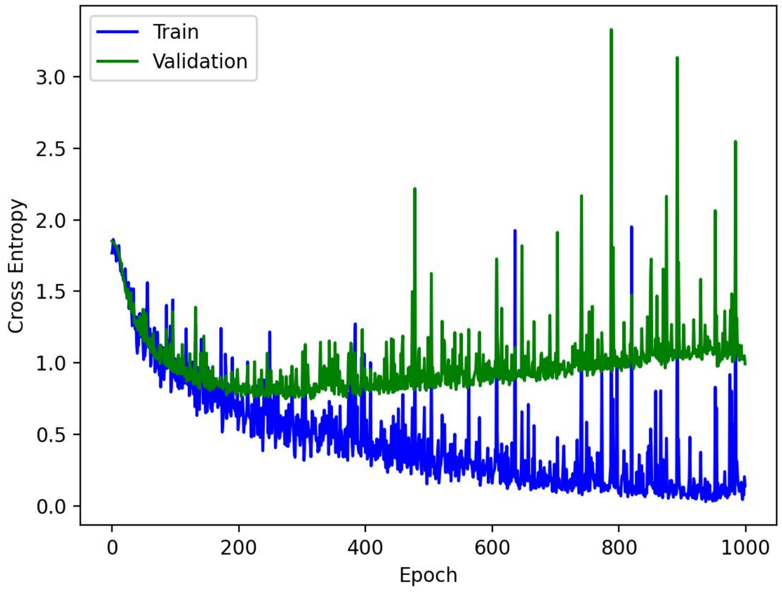


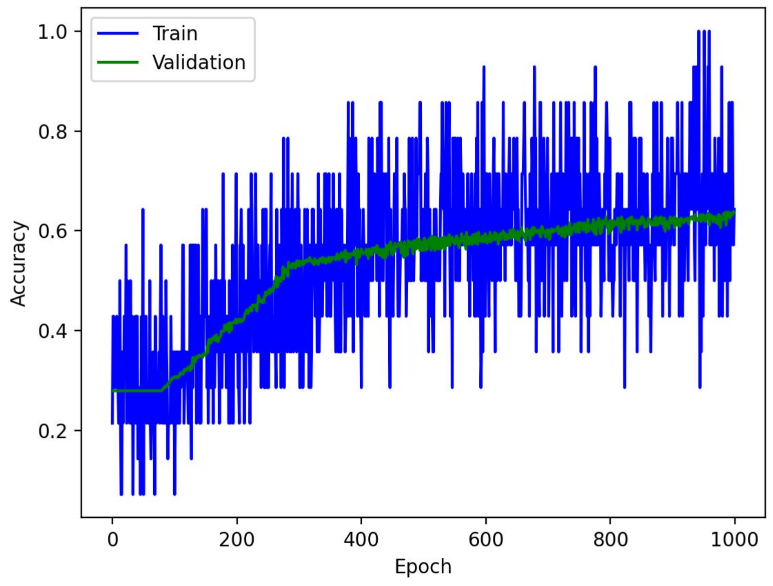


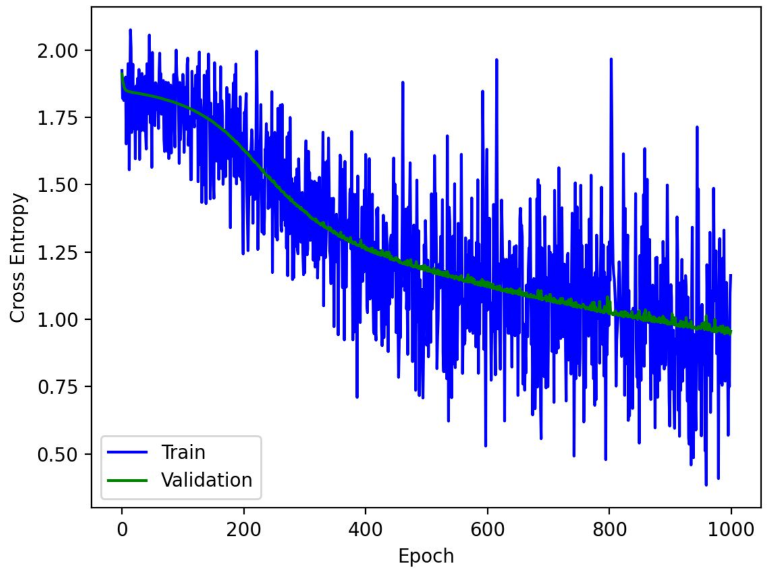
Naive NN Implementation

We also have the implementation of a naive neural net with backward propagation written by us. In this neural net, we only use partial dataset for training since we know if we have too much data, due to the simplicity of the net, it can not represent all the features. We tune the hyper parameters including the number of hidden layers, learning rate, min batch size and train it with Stochastic gradient descent approach. After several tests, the accuracy is still pretty low. The reason for this situation may be caused by the problem of overfit which is solvable due to the fact that our model is too naive, it can not catch some latent features of images. Here are some graphs for iteration times corresponded with cross entropy(loss) and accuracy.


Train
Validation
Accuracy
Epoch


Train
Validation
Cross Entropy
Epoch


Train
Validation
Accuracy
Epoch
1.0
0.8
0.6
0.4
0.2
0
200
400
600
800
1000


Train
Validation
Cross Entropy
Epoch
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0
200
400
600
800
1000

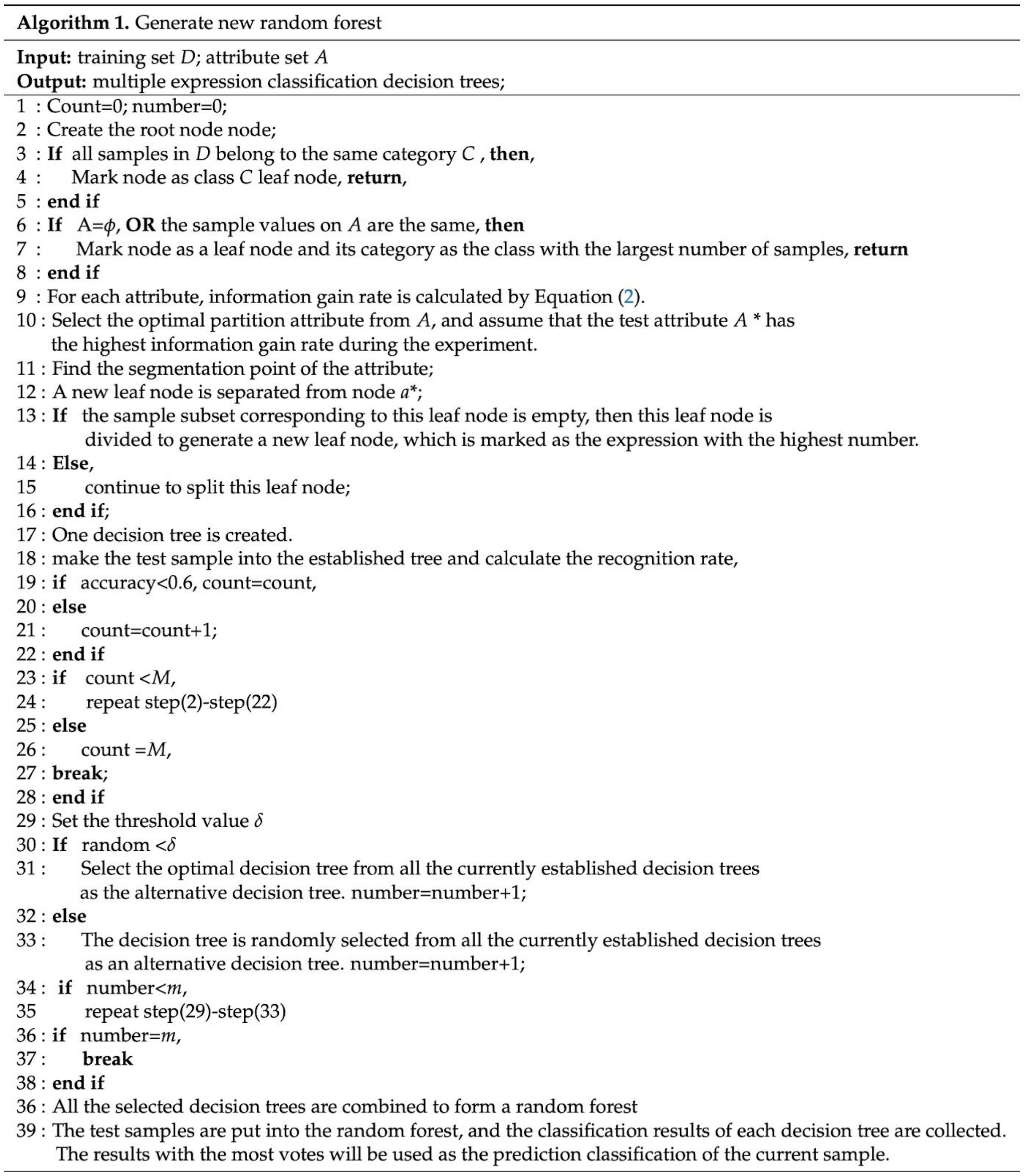
How is our algorithm using ResNet and CNN compared to algorithms using Random Forest and CNN?

We read a journal article regarding algorithms of facial expression recognition using Random Forest and CNN.

According to *(Wang, Li, Song, & Rong, 2019)*, they emphasized the importance of facial expression. Specifically, 50 percent of total emotional expression is facial expression, 40 percent is voice and about only 2 percent is language. They also stated that there are a total six basic emotions: happiness, surprise, sadness, fear, anger and disgust, which are common everywhere in the world.

For the choice of decision tree, they chose C4.5 classifier, which has been extensively used in the area of image recognition. Thus, they figured C4.5 would be appropriate. Their implementation of Random Forest is shown below.

Algorithm



Algorithm 1. Generate new random forest

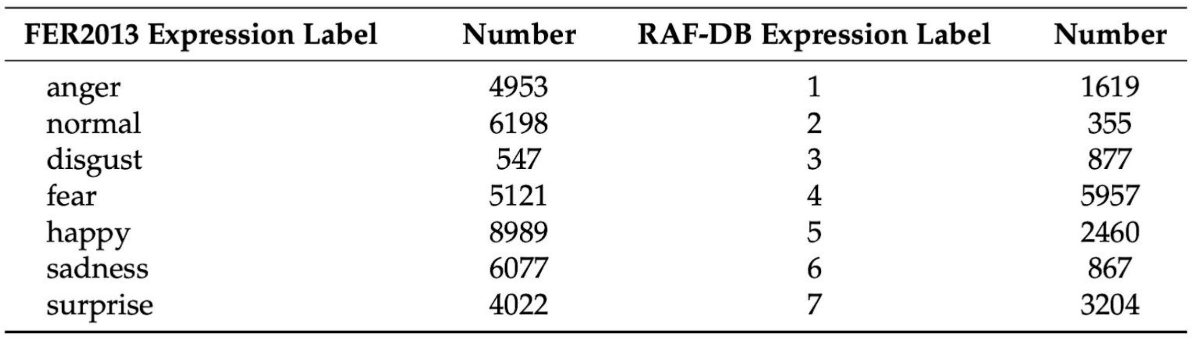
Input: training set D; attribute set A
Output: multiple expression classification decision trees;

```
1  : Count=0; number=0;
2  : Create the root node node;
3  : If  all samples in $D$ belong to the same category $C$ , then,
4  :     Mark node as class $C$ leaf node, return,
5  : end if
6  : If   A=$\phi$, OR the sample values on $A$ are the same, then
7  :     Mark node as a leaf node and its category as the class with the largest number of samples, return
8  : end if
9  : For each attribute, information gain rate is calculated by Equation (2).
10 : Select the optimal partition attribute from $A$, and assume that the test attribute $A$ * has
     the highest information gain rate during the experiment.
11 : Find the segmentation point of the attribute;
12 : A new leaf node is separated from node $a$*;
13 : If   the sample subset corresponding to this leaf node is empty, then this leaf node is
          divided to generate a new leaf node, which is marked as the expression with the highest number.
14 : Else,
15        continue to split this leaf node;
16 : end if;
17 : One decision tree is created.
18 : make the test sample into the established tree and calculate the recognition rate,
19 : if   accuracy<0.6, count=count,
20 : else
21 :      count=count+1;
22 : end if
23 : if    count <$M$,
24 :       repeat step(2)-step(22)
25 : else
26 :      count =$M$,
27 : break;
28 : end if
29 : Set the threshold value $\delta$
30 : If   random <$\delta$
31 :      Select the optimal decision tree from all the currently established decision trees
          as the alternative decision tree. number=number+1;
32 : else
33 :      The decision tree is randomly selected from all the currently established decision trees
          as an alternative decision tree. number=number+1;
34 :  if   number<$m$,
35        repeat step(29)-step(33)
36 : if  number=$m$,
37 :      break
38 : end if
36 : All the selected decision trees are combined to form a random forest
39 : The test samples are put into the random forest, and the classification results of each decision tree are collected.
     The results with the most votes will be used as the prediction classification of the current sample.
```

(Wang, Li, Song, & Rong, 2019)

Dataset

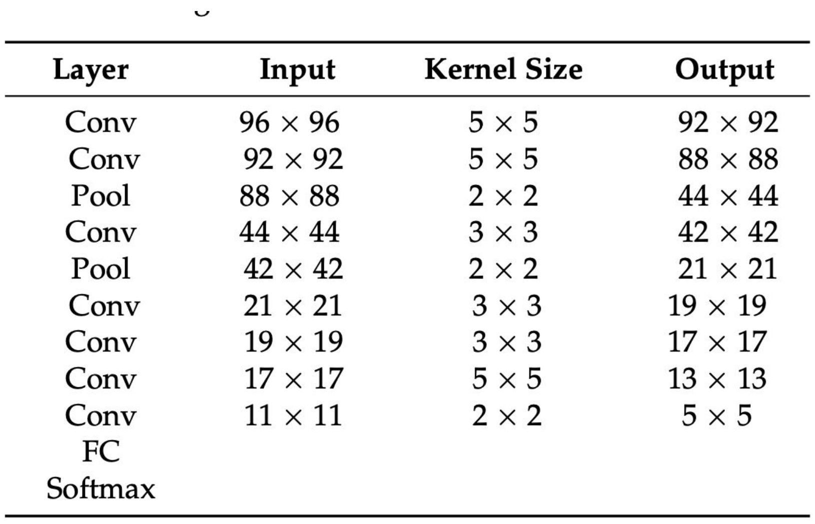
The dataset they chose was FER-2013, consisting of 35,887 different images, where 28,709 of them are for training. The other images make up two test sets. 3589 images are public test images and 3589 images are private test images. Public dataset is used to find and refine the CNN model, and the private test is used to determine accuracy of the CNN model. The dataset is shown below.



FER2013 Expression Label	Number	RAF-DB Expression Label	Number
anger	4953	1	1619
normal	6198	2	355
disgust	547	3	877
fear	5121	4	5957
happy	8989	5	2460
sadness	6077	6	867
surprise	4022	7	3204

(Wang, Li, Song, & Rong, 2019)

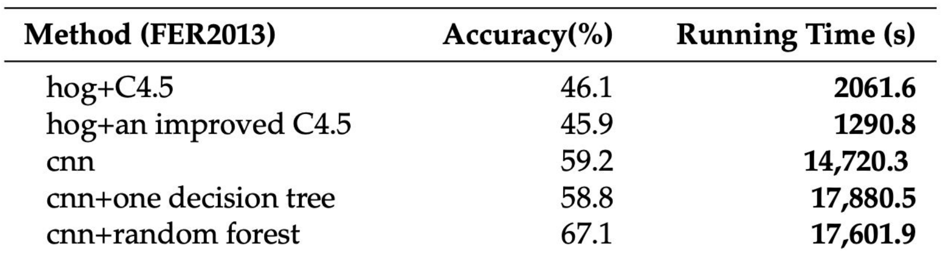
They apply a random forest into the last pool layer and a neural network shown below.



Layer	Input	Kernel Size	Output
Conv	96 × 96	5 × 5	92 × 92
Conv	92 × 92	5 × 5	88 × 88
Pool	88 × 88	2 × 2	44 × 44
Conv	44 × 44	3 × 3	42 × 42
Pool	42 × 42	2 × 2	21 × 21
Conv	21 × 21	3 × 3	19 × 19
Conv	19 × 19	3 × 3	17 × 17
Conv	17 × 17	5 × 5	13 × 13
Conv	11 × 11	2 × 2	5 × 5
FC			
Softmax			

(Wang, Li, Song, & Rong, 2019)

Results



Method (FER2013)	Accuracy(%)	Running Time (s)
hog+C4.5	46.1	**2061.6**
hog+an improved C4.5	45.9	**1290.8**
cnn	59.2	**14,720.3**
cnn+one decision tree	58.8	**17,880.5**
cnn+random forest	67.1	**17,601.9**

(Wang, Li, Song, & Rong, 2019)

Thoughts on their algorithm

Their results showed that they had an accuracy of 67.1% with running time 17,601.9s. Our implementation of facial expression recognition yields an accuracy rate of above 80%. Thus, we think our algorithm using ResNet is a better choice.

Conclusion

In conclusion, our dataset choice was decent and sufficed our goal of training our model and predicting people's emotion in pictures. We have built, trained a neural network using CNN and ResNet and they have satisfied performance, yielding accurate results. We have witnessed that complicated neural networks could help us solve real-world problems if it is properly constructed with appropriate hyper parameters such as learning rates, number of epochs and batch sizes. Tweaking those would significantly change our resulting model to varying degrees. We can say neural networks can be used in computer vision or image processing related programs or might help humans make predictions. One application of our neural network could be that we can make a program that automatically covers people's faces with appropriate emojis, or, we could use our model to make a program that can teach kids to recognize humans' emotions and learn English words of emotions if we replaced the emojis with correct English words.

Individual Contributions

All of our members have a clear understanding of our project. We collaborate through Zoom and share information immediately for each part. Here comes the detailed contribution for each group member.

Dataset Processing (Seek & Resize data): Yutong Chen, Keyi Zhang

MTCNN: Xiang Chen, Keyi Zhang

Train Model: Xiang Chen, Yutong Chen

Independent research: Xiang Chen, Keyi Zhang, Yutong Chen

Resnet 9: Yutong Chen, Keyi Zhang, Xiang Chen

Naive model: Xiang Chen, Keyi Zhang

Mask: Keyi Zhang, Yutong Chen

Data report: Keyi Zhang, Yutong Chen

Final report: Yutong Chen, Keyi Zhang, Xiang Chen

References

Facial expression using Random Forest:

Wang, Y., Li, Y., Song, Y., & Rong, X. (2019). Facial Expression Recognition Based on Random Forest and Convolutional Neural Network. *Information, 10*(12), 375. doi:10.3390/info10120375

OpenCV tutorial on cascade classifier:

doxygen. (n.d.). *Cascade Classifier*. OpenCV. Retrieved December 19, 2020, from https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html

Why do we need huge dataset:

Why do neural networks need so many training examples to perform? (2019, February 24). Cross Validated. https://stats.stackexchange.com/questions/394118/why-do-neural-networks-need-so-many-training-examples-to-perform

Why are CNN so good:

Mishra, P. (2019, December 17). *Why are Convolutional Neural Networks good for image classification?* Medium.

https://medium.com/datadriveninvestor/why-are-convolutional-neural-networks-good-for-image-classification-146ec6e865e8

Chatterjee, S. (2020, May 8). *Why Convolutional Neural Networks works better than Vanilla Neural Network??* Medium. https://medium.com/@somuchatterjee54/why-convolutional-neural-networks-works-better-than-vanilla-neural-network-1acba20761b2

Overview of ResNet:

Fung, V. (2018, June 21). *An Overview of ResNet and its Variants - Towards Data Science*. Medium. https://towardsdatascience.com/an-overview-of-resnet-and-its-variants-5281e2f56035

Real-World Affect Database

Li, Shan and Deng, Weihong and Du, JunPing(2017).*Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild*. Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on pages 2584--2593

http://www.whdeng.cn/raf/model1.html

Why are GPSs important for deep learning:

Shaikh, F. (2019, June 24). *Why are GPUs necessary for training Deep Learning models?* Analytics Vidhya. https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/

Blog of facial expression recognition using PyTorch:

Sharma, T. (2020, October 24). *Facial expression recognition using PyTorch - Jovian*. Medium. https://medium.com/jovianml/facial-expression-recognition-using-pytorch-b7326ab36157