Facial Emotion Recognition using EfficientNet

1 Definition

1.1 Project Overview

With the pandemic, the default mode of communication has been via video conference and remote meeting. This mode of communication has allowed for it to engage wider folk but has it been able to engage deeper?

Facial expressions are both a natural and direct means for human beings to convey their emotions and intentions. It is used to express non-verbal communication to one another whether deliberate or unintentionally. According to experts, these nonverbal signals make up a huge part of essential communication[1]. From our facial expressions, the things we *don't* say can still convey volumes of information[2]. Understanding and interpreting emotions among social interactions has became an important skillsets

Researchers are particularly interested in developing techniques to interpret, code facial expressions and extract these features in order to have a better prediction. This has widespread applications and implications for fields in retail, surveys and even negotiation. Even more so in this pandemic era, where meetings, presentations and interviews are done online and information of intention can be gathered by buyers and sellers alike.

1.2 Problem Statement

With the remarkable success of machine learning and particularly deep learning, the different types of architectures of this technique are exploited to achieve a better performance. The problem is a computer vision supervised learning classifier problem based upon multiple classes. From an image, one must predict the correct classification of the emotion display (of which there are originally 7). The emotion state display in this image are: (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). We will have to build a model to correctly assess the emotion display by the faces expression in the picture

This project has twin purpose

1. It is to make a study automatic facial emotion recognition FER via deep learning
2. To fulfill the requirements of Udacity Machine Learning Nanodegree

1.3 Metrics

The metrics used in this study will be the accuracy or how accurate is the prediction of the facial emotion which is the statistic used in the competition

$$Overall\ Accuracy\ =\ \frac{Sum\ of\ True\ Positive\ in\ each\ class}{Total\ No\ of\ cases}$$

This is similar to how scikit learn library measure accuracy.

However since this is an imbalance set, precision, recall and F-1 score of each class should be calculated where

$$Precision\ =\ \frac{TP}{TP+FP},\quad Recall\ =\ \frac{TP}{TP+FNs},\ F1\ =\ 2\frac{Precision*Recall}{Precision+Recall}$$

Where TP, FP and FN are from each class

2 Analysis

2.1 Data Exploration and Visualization

The dataset, FER2013 is downloaded from Kaggle, based on a competition: "**Challenges in Representation Learning: Facial Expression Recognition Challenge". [4].** The data can be directly downloaded from Kaggle using its API, provided the authentication json file is being downloaded to the local drive.

2.1.1 Glimpse Into FER2013 Database

Using pandas.read_csv, we have opened up and observed that the database has 35887 items and is represented by 3 columns:

- emotion is the class label of data, y
- pixels is the data or x
- and usage is which stage the data is supposed to be used for

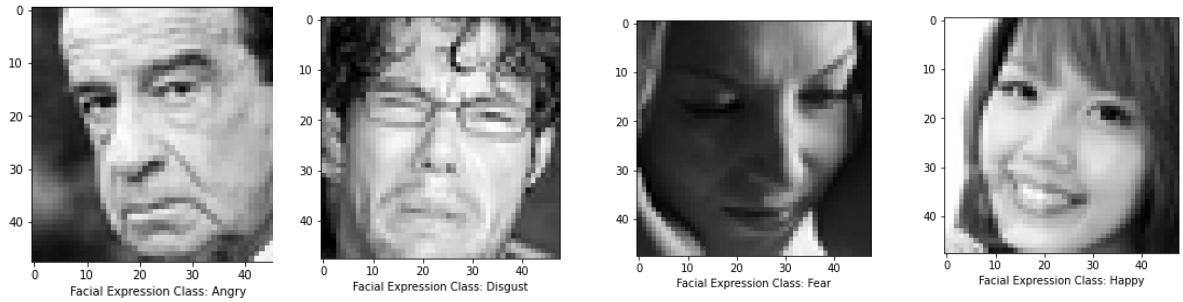
	emotion	pixels	Usage
0	0	70 80 82 72 58 58 60 63 54 58 60 48 89 115 121...	Training
1	0	151 150 147 155 148 133 111 140 170 174 182 15...	Training
2	2	231 212 156 164 174 138 161 173 182 200 106 38...	Training
3	4	24 32 36 30 32 23 19 20 30 41 21 22 32 34 21 1...	Training
4	6	4 0 0 0 0 0 0 0 0 0 0 0 0 3 15 23 28 48 50 58 84...	Training

Figure 1 FER2013 Database in Tabular Format

The break down of the data would be further discussed in Section 2.1.3

2.1.2 Conversion of Pixel Into Image

We then proceed to convert the pixels into images (based on the information given that the image supplied is gray-scale) and see for ourselves whether the label corresponds with the image. From the images itself, we can see that some of the images classification is questionable. For example, in the images below, “Disgust” might equally likely be interpreted as “Sad” by another labeller. The same could be said for “Surprise” can be interpreted to be “Happy”. Realistically speaking, a person can go through 1-2 emotion state at the same point of time (for example, we could be both “Surprise” and “Happy” when presented with unexpected gift or situation or we could be both “Angry” and “Sad” or maybe even fearful when we are confronted with unfortunate incident)



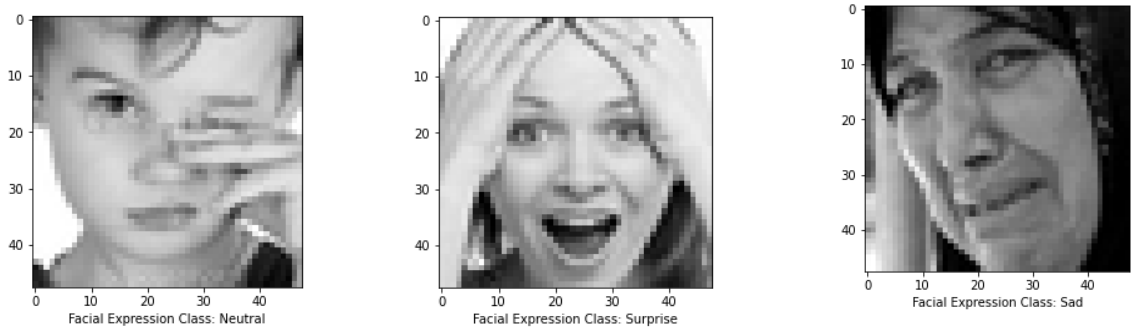


Figure 2. Representative Image and Their Corresponding Emotion State (starting from top left in clockwise fashion: Angry, Disgust, Fear, Happy, Sad, Surprise, Neutral)

There is quite a number of irregularities within the test set. Such as an empty picture, cartoon drawn faces or even faces with watermark. An initial attempt to use a face detector (Facenet MTCNN) to detect faces and automate elimination of faces prove to be a failure as less than 1% of the picture actually makes it for the selection(perhaps due to its low resolution, compared with modern network detector). However for interest sake to mimics actual conditions in actual competition, I did not choose to eliminate them

2.1.3 Data Distribution

I then dive into the data to find out the proportion of each class and the distribution of their usage. From the brief look at the dataset, we can observe that there is no over dominating class with a large number of images in its dataset, though smile is almost twice the number of the next biggest class. The emotion state of disgust is very little. This might mean it might not form a representative proportion in the dataset and class imbalance is a serious problem we would have to consider for this data set. We would need to either augment it or perhaps to think of ways to discard some of the images in other sets

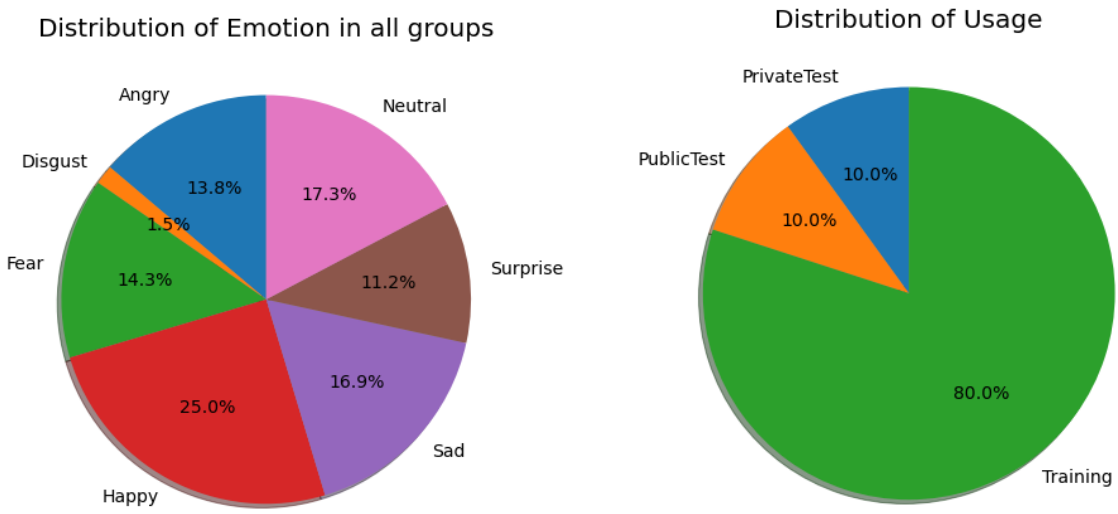
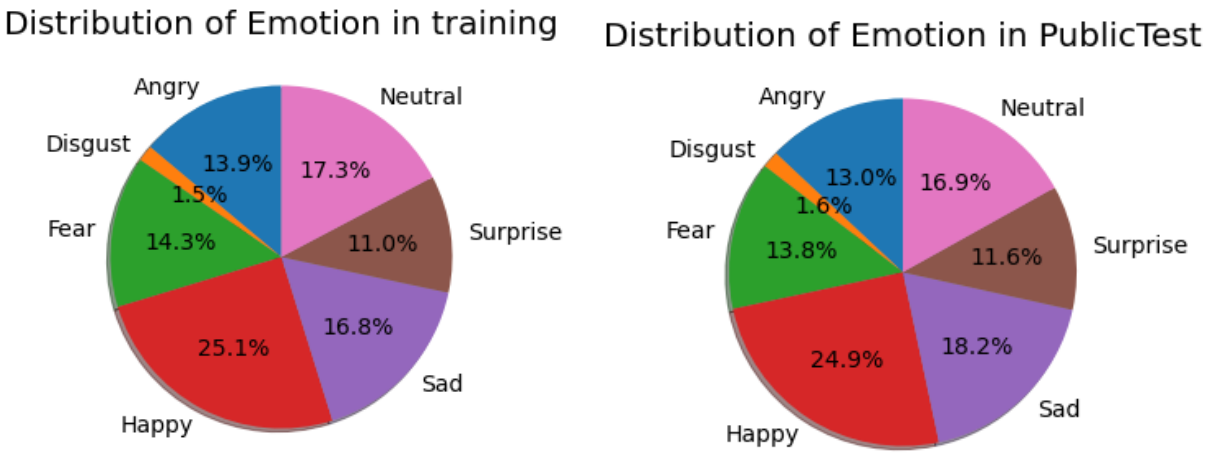


Figure 3. Distribution of Various Emotion Class with Distribution of Their Usage

Let see the distribution in the Training, Validation and Training set. They do not deviate too much from the main distribution statistic



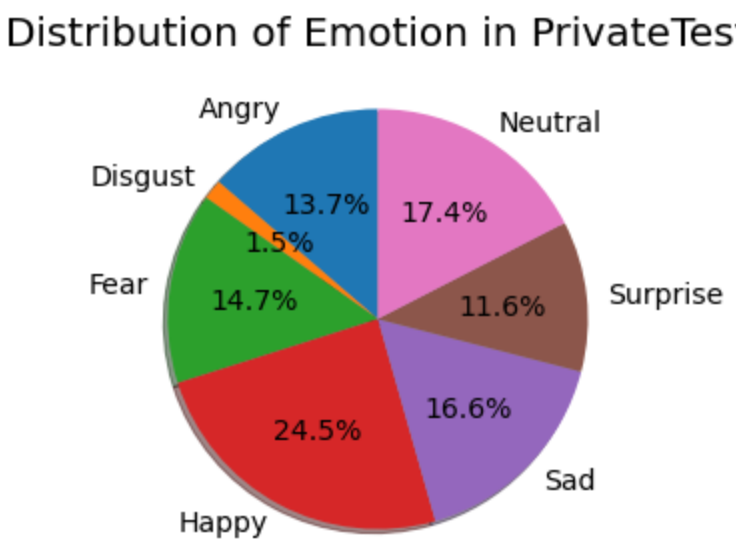


Figure 4. Distribution of Various Emotion Class According to Their Usage

2.2 Algorithms and Techniques

2.2.1 EfficientNet

EfficientNet does not boast of novel architecture design like Resnet which introduces a short-cut by-pass connection that skipped layers and alleviate problems of vanishing gradient. Or that of Inception, which likewise introduces to us the concept of having different convolution filter size in order to capture both global and local information. Nor like MobileNet which uses uses depthwise separable convolutions to reduce learning parameter and enhance training speed yet retain respectable performance

Efficient is concerned with the efficient uses of computational resource yet maintaining same superior performance. It is based along the 3 common strategy of improving results in Deep Learning, namely to increase depth (increase filter layers), width (capture broad information) and resolution (inputs have more information). Simply put, EfficientNet offered a smaller model and also cheaper computational cost

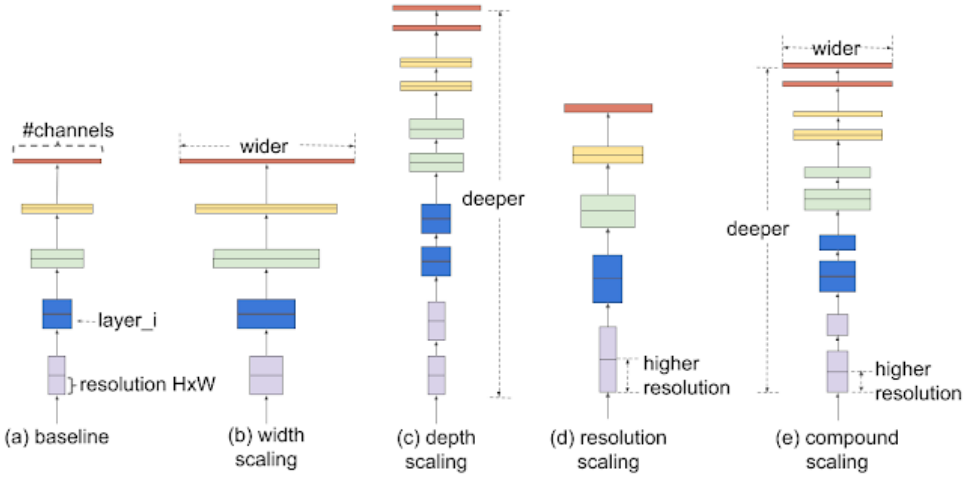


Figure 5. Common Approaches to Improving Deep Learning Network

EfficientNet make use of a technique known as Compound Scaling which is used to scale the 3 factors according to the algorithms below. ϕ is a user-defined, global scaling factor (integer) that controls how many resources are available whereas **α**, **β**, and **γ** determine how to assign these resources to network depth, width, and resolution respectively. FLOPS (floating point operations per second), which is determined by the depth,width and resolution. By making FLOPS a constrain, the parameters (depth, width, and resolution) can be determined by grid search and from this parameters by increasing ϕ, other order of EfficientNet is developed.

$$\begin{aligned} \text{depth: } & d = \alpha^\phi \\ \text{width: } & w = \beta^\phi \\ \text{resolution: } & r = \gamma^\phi \\ \text{s.t. } & \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \end{aligned}$$

Figure 6. Ideas Governing EfficientNet

The base network, EfficientNet0 is of this structure and subsequently EfficientNet1-7 are developed based on the above ideas

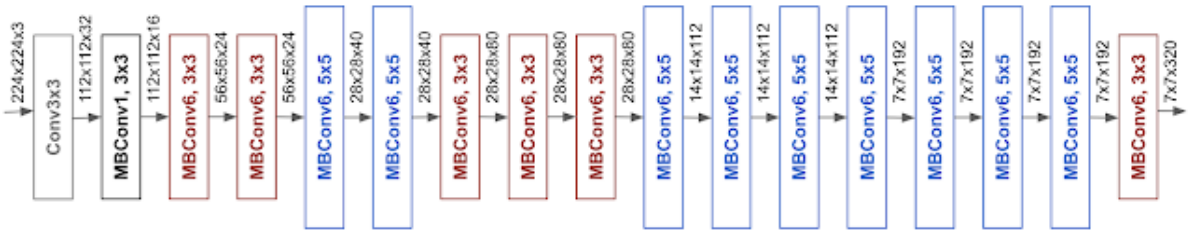


Figure 7. EfficientNet0 Layers Layout

2.3 Benchmark

As stated in the proposal, this is a rather tough data set when it was first introduced, human accuracy on a subset of the dataset was 65±5% [3]. And the final winner of the competition has achieved a result of 71%. Even after so many years, on the Kaggle board, the score board barely progressed by one percent. For a novice we aimed to have a baseline goal of 60% and performance goal of around than 65%.

3 Methodology

3.1 Data Preprocessing

Once we obtain the data through Kaggle API, we separate the images in the database to Training, Validation and Test Set. With training set to be used for training and validation set would be used for validation of training. Finally Test set would be used for testing

That is not really much to do in preparation of data for this data set, with the exception of creating the Dataset Class, FERDataset together with the transform we are going to use. The transform that we are going to is as follows

Dataset	Transformation Adopted
Train	transforms.ToPILImage(), transforms.RandomCrop(48, padding=2, padding_mode='reflect'), transforms.RandomAffine(degrees=(-15,15), translate=(0.1,0.1)), transforms.RandomHorizontalFlip(), transforms.Grayscale(num_output_channels=1), transforms.RandomAutocontrast(), transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,), inplace=True)

Validate	transforms.ToPILImage(), transforms.Grayscale(num_output_channels=1), transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))
Test	transforms.ToPILImage(), transforms.Grayscale(num_output_channels=1), transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))

3.2 Implementation

As we are training locally on a laptop, efficient use of resources is necessary especially we employed deep learning to solve our problem

3.2.1 Environment

The environment that I am using is a laptop, running on Ubuntu 20.04, 32 GB RAM and equipped with a GTX 1650, 4GB GPU. This is somewhat quite modest in modern day computation power. This somewhat influences my choice of using EfficientNet as a network of choice, being able to achieve the most out of what one has.

Setting up of the Environment in Jupyter Notebook.

- First create the environment in conda
- Activate that environment
- conda install pytorch torchvision torchaudio cudatoolkit=11.1 -c pytorch
  Where cudatoolkit version is dependent on your gpu and test if you can access cuda with
  ```
  Import torch
  torch,cuda.is_available()    ## the execution should return true
  ```
- Install any libraries that the ipython notebook needs. Or you have other chances in the Jupyter Notebook itself
- Install the ipython: conda install -c anaconda ipykernel
- Then link the environment with ipython: python -m ipykernel install --user --name=yourenvname
- Finally activate Jupyter notebook and see if the environment is available within the kernel

3.2.2 Actual Implementation

For the network implementation, we downloaded the efficientnet_Pytorch implementation from https://github.com/lukemelas/EfficientNet-PyTorch. We have amended the input channel number of the input layer and also the final classifier class.

```
from efficientnet_pytorch import EfficientNet

class Conv2dStaticSamePadding(nn.Conv2d):
    """2D Convolutions like TensorFlow's 'SAME' mode, with the given input image size.
       The padding mudule is calculated in construction function, then used in forward.
    """

    # With the same calculation as Conv2dDynamicSamePadding

    def __init__(self, in_channels, out_channels, kernel_size, stride=1, image_size=None, **kwargs):
        super().__init__(in_channels, out_channels, kernel_size, stride, **kwargs)
        self.stride = self.stride if len(self.stride) == 2 else [self.stride[0]] * 2

        # Calculate padding based on image size and save it
        assert image_size is not None
        ih, iw = (image_size, image_size) if isinstance(image_size, int) else image_size
        kh, kw = self.weight.size()[-2:]
        sh, sw = self.stride
        oh, ow = math.ceil(ih / sh), math.ceil(iw / sw)
        pad_h = max((oh - 1) * self.stride[0] + (kh - 1) * self.dilation[0] + 1 - ih, 0)
        pad_w = max((ow - 1) * self.stride[1] + (kw - 1) * self.dilation[1] + 1 - iw, 0)
        if pad_h > 0 or pad_w > 0:
            self.static_padding = nn.ZeroPad2d((pad_w // 2, pad_w - pad_w // 2,
                                                pad_h // 2, pad_h - pad_h // 2))
        else:
            self.static_padding = nn.Identity()

    def forward(self, x):
        x = self.static_padding(x)
        x = F.conv2d(x, self.weight, self.bias, self.stride, self.padding, self.dilation, self.groups)
        return x


class EmotionDetection(EmotionBase):
    def __init__(self,n_channels, n_classes):

        super(EmotionDetection, self).__init__()

        self.effnet                 =  EfficientNet.from_pretrained('efficientnet-b0', num_classes=7)
        self.effnet._conv_stem      = Conv2dStaticSamePadding(1,32,kernel_size=(3,3), stride=(2,2),
                                                              image_size=(48),bias=False)

    def forward(self, input):
        x = self.effnet(input)
        return x

model = EmotionDetection(1,7)

print(model)
```

The EmotionDetection class is based on EmotionBase which helps to handle the training and validation cycle calculation

```python
## class to support loading of data and loss calculation for training and validation steps
class EmotionBase(nn.Module):

    def __init__(self):
        super(EmotionBase, self).__init__()
        self.val_acc = 0.0

    # this takes is batch from training dl
    def training_step(self, batch):
        images, labels = batch
        images, labels = images.to(device), labels.to(device) # send to the appropriate device to train
        out            = self(images).to(device)         # calls the training model and generates predictions
        loss           = F.cross_entropy(out, labels)    # calculates loss compare to real labels using cross entropy
        return loss

    # this takes in batch from validation dl
    def validation_step(self, batch):
        images, labels = batch
        out            = self(images)
        loss           = F.cross_entropy(out, labels)     # calculate the loss using cross_entropy
        acc            = accuracy(out, labels)            # calls the accuracy function to measure the accuracy
        return {'val_loss': loss.detach(), 'val_acc': acc}

    def validation_epoch_end(self, outputs):
        batch_losses = [x['val_loss'] for x in outputs]
        epoch_loss = torch.stack(batch_losses).mean()    # finds out the mean loss of the epoch batch

        batch_accs = [x['val_acc'] for x in outputs]
        epoch_acc = torch.stack(batch_accs).mean()       # finds out the mean acc of the epoch batch

        return {'val_loss': epoch_loss.item(), 'val_acc': epoch_acc.item()}

    def epoch_end(self, epoch, result):
        print("Epoch [{}], last_lr: {:.5f}, train_loss: {:.4f}, val_loss: {:.4f}, val_acc: {:.4f}".format(
            epoch, result['lrs'][-1], result['train_loss'], result['val_loss'], result['val_acc']))

        result = result['val_acc']                       # save the best model
        if result > self.val_acc:
            self.best_desc  = "{:.4f}".format((result)) + '.pth'
            self.best_model = model.state_dict()
            self.val_acc = result
```

We have tried to retrain the whole network due to that it was said that ImageNet result was mainly RGB and would not generalise well for grayscale images. The setting of network training was as below:

max_lr = 0.001
grad_clip = 0.1
weight_decay = 1e-4
max_epoch = 30-50

We plotted out the learning rate history, training and validation loss and found that most of the time, learning would stop around epoch no 35-38 and difference between training loss and validation loss would diverge round epoch round 30-35

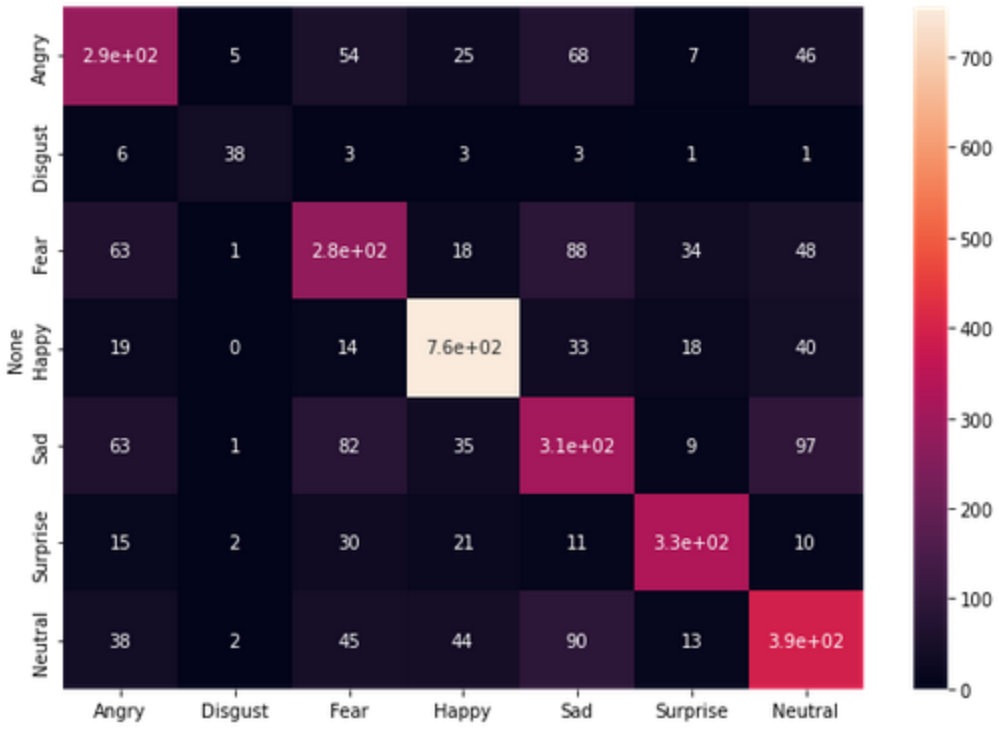
```python
@torch.no_grad()    # this is for stopping the model from keeping track ofold parameters
def evaluate(model, val_loader):
    # This function will evaluate the model and give back the val acc and loss
    model.eval()
    outputs = [model.validation_step(batch) for batch in val_loader]
    return model.validation_epoch_end(outputs)

# getting the current learning rate
def get_lr(optimizer):
    for param_group in optimizer.param_groups:
        return param_group['lr']

# this fit function follows the intuition of 1cycle lr
def fit(epochs, max_lr, model, train_loader=train_dl, val_loader=val_dl,
        weight_decay=0, grad_clip=None, opt_func=torch.optim.Adam):
    torch.cuda.empty_cache()
    history = []    #keep track of the evaluation results

    # setting upcustom optimizer including weight decay
    optimizer = opt_func(model.parameters(), max_lr, weight_decay=weight_decay)
    # setting up 1cycle lr scheduler
    sched = torch.optim.lr_scheduler.OneCycleLR(optimizer, max_lr, epochs=epochs, steps_per_epoch=len(train_loader))

    for epoch in range(epochs):
        # training
        model.train()
        train_losses = []                          ## used to collect all historical loss
        lrs = []                                   ## used to collect all learning rate
        for batch in train_loader:
            loss = model.training_step(batch)
            train_losses.append(loss)
            loss.backward()

            # gradient clipping
            if grad_clip:
                nn.utils.clip_grad_value_(model.parameters(), grad_clip)

            optimizer.step()
            optimizer.zero_grad()

            # record the lr
            lrs.append(get_lr(optimizer))
            sched.step()

        #validation
        result = evaluate(model, val_loader)
        result['train_loss'] = torch.stack(train_losses).mean().item()
        result['lrs'] = lrs
        model.epoch_end(epoch, result)
        history.append(result)
    return history
```

4 Experiments and Results

4.1 Model Evaluation and Validation

We then loaded our best trained result for evaluation with the test set. And this is our result

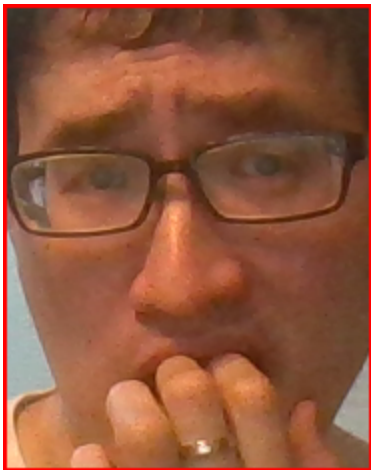
```
The overall accurate percentage is 0.6639732516021176
The per-class precision value is :
[0.58367347 0.7755102  0.54761905 0.83795782 0.51166667 0.799511
 0.61949686]
The per-class recall value is :
[0.58248473 0.69090909 0.52272727 0.8589306  0.51683502 0.78605769
 0.62939297]
The per-class fscore value is :
[0.58307849 0.73076923 0.53488372 0.84831461 0.51423786 0.79272727
 0.62440571]
```



We have also deployed our trained model to capture live images in webcam and tested with our model. Some of the image and their label are presented here

Happy

Neutral



Fear

5 Conclusion

5.1 Reflection

Much of the challenges in doing this project lies in learning pyTorch framework and scaling it to work and fit to your environment (hardware, cpu, python environment). As a Keras user, we are spoon fed with high level APIs which most of the time help us to do the heavy lifting and handle the backend for the project but in pyTorch, sometimes we have to walk the extra mile to get what we want. Here are some of the difficulties that we have encounter

- One of the tasks is to handle the transfer of model, datas and outputs from the CPU to GPU. One would have to figure out and remember where the data/models etc is residing at this point of the time.
- And also to scale your training model to fit into your GPU like reducing batch size, image size when training so as do not run into situation of out of memory before and while training
- Another difficulty lies in adjusting the right EfficientNet to work. For example EfficientNet1 is first used, as thought that its input sizing is 240 (a multiple of 48) should be a right match for the image but the increase in computation time was rather too much and the benefits seem little

For the Network Model itself, some of the challenges lies in preventing it from being overfitted. Early on in the project, I am quite happy to see that validation accuracy was increasing with increasing epoch but failed to take notice that validation loss was also increasing at the same time. It is on later I then realise the model may have been overtrained and the training could stop at earlier time

5.2 Improvement

5.2.1 Databases

Some of the improvement lies in getting better data (aka more correctly labelled) such as CK+ dataset (extended Cohn-Kanade). The dataset has been tested by many and found to yield excellent results (some counting over 90% accuracy). However, it must be remembered these high accuracy is due to the database using pictures from a small group of people (around 120). This make the emotion detection rather consistent (emotion display among small group of people are quite similar) but could not generalise as well as FER2013 database

Other things we can improve general facial emotion detection accuracy is that we should take newer images. The database was created almost 10 years ago and hence the lower resolution in the image, 48x 48. If we merely expand the image size to fit modern networks, we could likely create information that is of little to no value to the training. Sometimes micro facial features which could only be seen with images with better resolution could help in gaining the extra percentages in accuracy. This is akin to the approaches mentioned in EfficientNet

Another way is we can come up with top 2 prediction metrics model and rank them. This in turn would help to eliminate ambiguity in the label (as discussed in Section 2.1.2) or we can relabel the image and change this into a multilabel instead of a multiclass problem.

5.2.1 Method

Other ways could be that to try some other kinds of network, like for example MTCNN and maybe to frame the problem as a biinary

References

1. Tipper CM, Signorini G, Grafton ST. Body language in the brain: constructing meaning from expressive movement. *Front Hum Neurosci*. 2015;9:450. doi:10.3389/fnhum.2015.00450
2. Foley GN, Gentile JP. Nonverbal communication in psychotherapy. *Psychiatry (Edgmont)*. 2010;7(6):38-44.
3. "Challenges in Representation Learning: A report on three machine learning contests." I Goodfellow, D Erhan, PL Carrier, A Courville, M Mirza, B Hamner, W Cukierski, Y Tang, DH Lee, Y Zhou, C Ramaiah, F Feng, R Li, X Wang, D Athanasakis, J Shawe-Taylor, M Milakov, J Park, R Ionescu, M Popescu, C Grozea, J Bergstra, J Xie, L Romaszko, B Xu, Z Chuang, and Y. Bengio. arXiv 2013