Introduction

This project was an attempt to improve upon the Remote Emotion Detector [1], a Machine Learning project developed during the Summer of 2022. The aim of the project was to classify the emotion (from seven possible emotions) of a person shown on a webcam located on a remote computer. A diagram detailing the functionality of this system is shown below in figure 2.

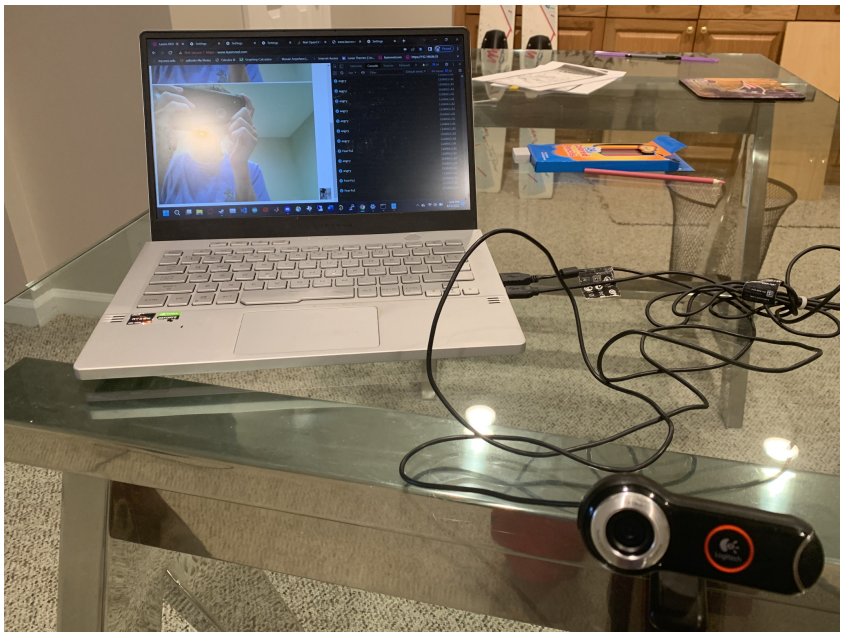

Figure 1 Remote Emotion Classifier in action

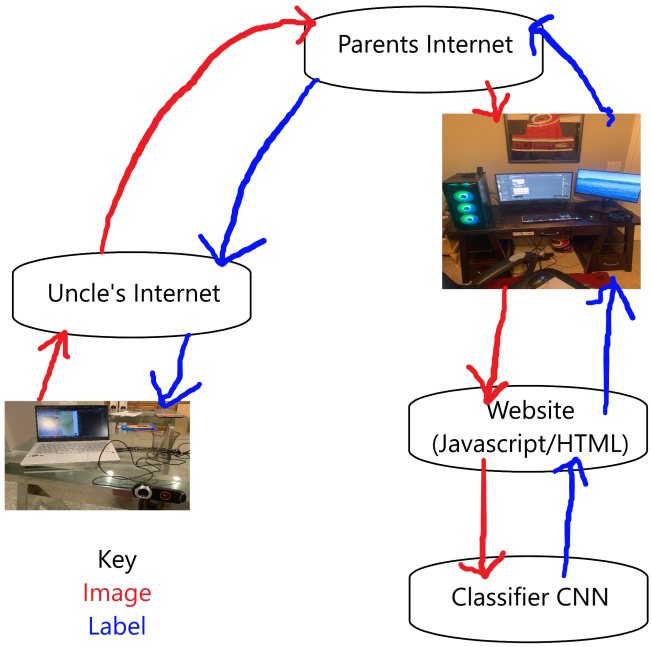

Figure 2 Remote Emotion Detector functional diagram

The major problem that faced the original version of this project was that it could not reliably classify more complex emotions, such as disgust or anger. This new and improved version of this project seeks to rectify this problem.

Dataset

The major problem that the previous model used for this project had was its very small 48x48 training images*, which not only prevented training the model beyond ~64 % validation accuracy, but also substantially crippled the model's ability to make accurate inferences when run on a live webcam.

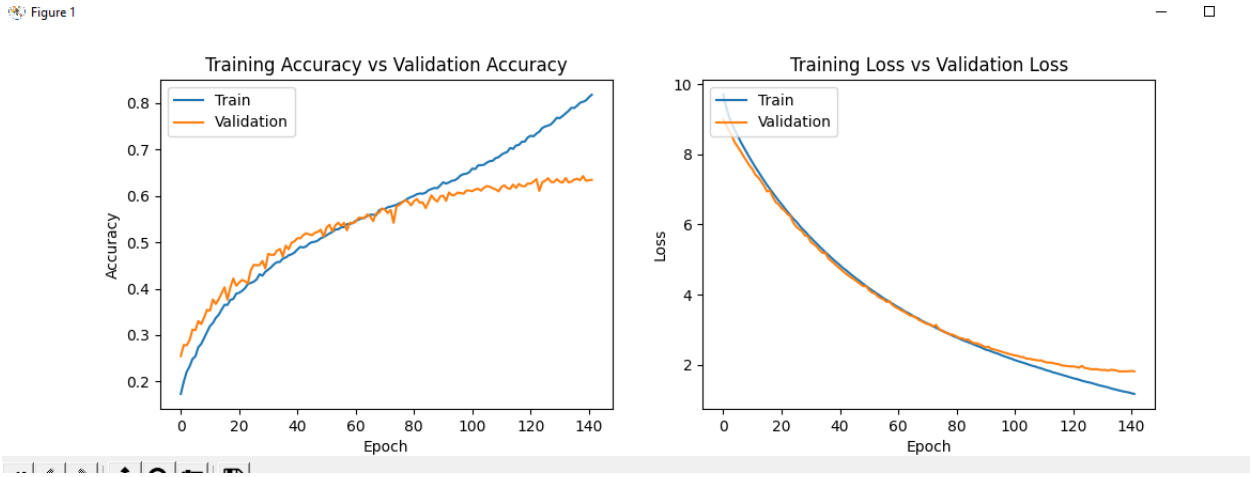

Figure 3 Training and validation loss/accuracy graph from training of V1 detector's model
*note: during the presentation it was claimed that the original images were 64x64, this was in error.

The obvious solution to this problem would be to train a new model with a dataset with larger images. Initially, the Caltech 10k faces dataset [2] was considered, as it contained over 7000 images with an average resolution of 304x312. However two issues arose that ultimately prevented its use for the model:

1. Some images in the dataset contained more than one face, which the original detector did not have to account for.
2. The dataset was not labeled for emotional classification, but was instead labeled for feature identification (eyes, nose and mouth)

ReadMe.txt

```
The ground truth contains the following information:
image-name Leye-x Leye-y Reye-x Reye-y nose-x nose-y mouth-x mouth-y
```

Figure 4 Caltech 10k faces dataset ground truth

Ultimately it was decided that the best dataset to use for this project would be the Extended Cohn-Kanade dataset [3], a copy of which was found in the CS229 repository on Github [4]. This dataset came with several upsides:

- Its image size was 640x490, which was changed to 640x480 for the final Pytorch model, but even still this was a massive upgrade in data that could be used as features.

- It came with all 7 emotions the previous model featured (in alphabetical order: anger, disgust, fear, happiness, neutral, sadness, surprise) plus one extra, contempt (which was ignored to keep the model as similar to the original as possible). Each emotion had a folder which contained the respective sample images for their label.
- A reasonably high diversity of its samples, with variations in gender, race, background and lighting conditions, which would all help create a more generalized model.
- After trimming out some samples, 848 samples were used for training in total, which after taking into consideration image size gave us 260,505,600 data points (1 pixel = 1 data point).

Figure 5 visual comparison of 'anger' category samples between previous dataset and Cohn-Kanade dataset

One major downside of the Cohn-Kanade dataset is the timestamps located on the images, which could potentially create false features for the model to focus on. Despite this, it still appeared to be the best dataset freely available online. To assist the training process, the pixel data was normalized from a range of 0 to 255 to a range of 0 to 1, which was the only augmentation performed on the dataset. The dataset was split 70 % for training data and 30 % for validation data, with a test set being absent as live testing of the model on a webcam would substitute for it. Another issue that was not accounted for during the training process was that the dataset was heavily imbalanced, with neutral samples making up 556 of 848 samples (65.56 %). This should have been caught beforehand, however oversight from other project members was nonexistent, thus this error was allowed to go unchecked.

The first major hurdle of the project that had to be overcome was the issue of loading the data itself, as while the data stored in the .png files was only a little 100 mb in size, this was quickly enlarged when loaded into Python. To this end, custom dataloaders were used for both models.

Model Architecture

Two models would be trained for use in this project, one using Keras and the other using Pytorch.

Keras model

The Keras model was based off of the preexisting model architecture used for the original remote emotion detector. It was the first neural network I ever trained, and it clearly shows, being more of a hack job than being built of any reliable model architecture. I believe the model summary speaks for itself.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 640, 490, 32)      320

 conv2d_1 (Conv2D)           (None, 640, 490, 64)      18496

 batch_normalization (BatchN  (None, 640, 490, 64)     256
 ormalization)

 max_pooling2d (MaxPooling2D  (None, 320, 245, 64)     0
 )

 dropout (Dropout)           (None, 320, 245, 64)      0

 conv2d_2 (Conv2D)           (None, 320, 245, 128)     204928

 batch_normalization_1 (Batc  (None, 320, 245, 128)    512
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 160, 122, 128)    0
 2D)

 dropout_1 (Dropout)         (None, 160, 122, 128)     0

 conv2d_3 (Conv2D)           (None, 160, 122, 512)     590336

 batch_normalization_2 (Batc  (None, 160, 122, 512)    2048
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 80, 61, 512)      0
 2D)
```

```
dropout_2 (Dropout)          (None, 80, 61, 512)       0

conv2d_4 (Conv2D)            (None, 80, 61, 512)       2359808

batch_normalization_3 (Batc  (None, 80, 61, 512)       2048
hNormalization)

max_pooling2d_3 (MaxPooling  (None, 40, 30, 512)       0
2D)

dropout_3 (Dropout)          (None, 40, 30, 512)       0

flatten (Flatten)            (None, 614400)            0

dense (Dense)                (None, 256)               157286656

batch_normalization_4 (Batc  (None, 256)               1024
hNormalization)

dropout_4 (Dropout)          (None, 256)               0

dense_1 (Dense)              (None, 512)               131584

batch_normalization_5 (Batc  (None, 512)               2048
hNormalization)

dropout_5 (Dropout)          (None, 512)               0

dense_2 (Dense)              (None, 1)                 513

=================================================================
```

```
Total params: 160,600,577
Trainable params: 160,596,609
Non-trainable params: 3,968
```

Figure 6 Keras-based CNN Architecture

Pytorch Model

The pytorch model was based off of ResNet18, which was found to have the lowest loss and highest validation accuracy of all the model architectures in Homework 3.

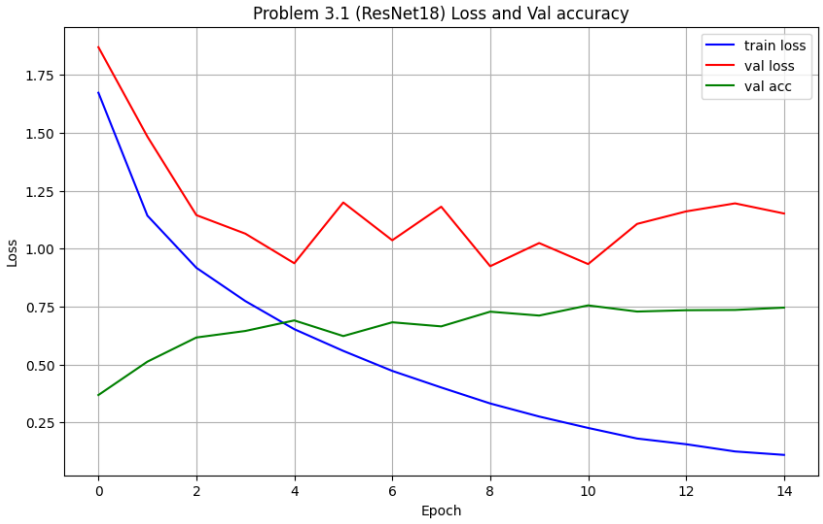

Figure 7 ResNet18 training data from homework 3

Compared to the Keras-based model, this model was much more efficient, using only about 1/16 of the Keras model's parameters.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1         [-1, 64, 320, 240]           3,200
       BatchNorm2d-2         [-1, 64, 320, 240]             128
              ReLU-3         [-1, 64, 320, 240]               0
         MaxPool2d-4         [-1, 64, 160, 120]               0
            Conv2d-5         [-1, 64, 160, 120]          36,928
       BatchNorm2d-6         [-1, 64, 160, 120]             128
            Conv2d-7         [-1, 64, 160, 120]          36,928
       BatchNorm2d-8         [-1, 64, 160, 120]             128
          Residual-9         [-1, 64, 160, 120]               0
           Conv2d-10         [-1, 64, 160, 120]          36,928
      BatchNorm2d-11         [-1, 64, 160, 120]             128
           Conv2d-12         [-1, 64, 160, 120]          36,928
      BatchNorm2d-13         [-1, 64, 160, 120]             128
         Residual-14         [-1, 64, 160, 120]               0
           Conv2d-15         [-1, 128, 80, 60]           73,856
      BatchNorm2d-16         [-1, 128, 80, 60]             256
           Conv2d-17         [-1, 128, 80, 60]         147,584
      BatchNorm2d-18         [-1, 128, 80, 60]             256
           Conv2d-19         [-1, 128, 80, 60]           8,320
         Residual-20         [-1, 128, 80, 60]               0
           Conv2d-21         [-1, 128, 80, 60]         147,584
      BatchNorm2d-22         [-1, 128, 80, 60]             256
           Conv2d-23         [-1, 128, 80, 60]         147,584
      BatchNorm2d-24         [-1, 128, 80, 60]             256
         Residual-25         [-1, 128, 80, 60]               0
           Conv2d-26         [-1, 256, 40, 30]         295,168
      BatchNorm2d-27         [-1, 256, 40, 30]             512
           Conv2d-28         [-1, 256, 40, 30]         590,080
      BatchNorm2d-29         [-1, 256, 40, 30]             512
           Conv2d-30         [-1, 256, 40, 30]          33,024
         Residual-31         [-1, 256, 40, 30]               0
           Conv2d-32         [-1, 256, 40, 30]         590,080
      BatchNorm2d-33         [-1, 256, 40, 30]             512
           Conv2d-34         [-1, 256, 40, 30]         590,080
      BatchNorm2d-35         [-1, 256, 40, 30]             512
         Residual-36         [-1, 256, 40, 30]               0
           Conv2d-37         [-1, 512, 20, 15]       1,180,160
      BatchNorm2d-38         [-1, 512, 20, 15]           1,024
           Conv2d-39         [-1, 512, 20, 15]       2,359,808
      BatchNorm2d-40         [-1, 512, 20, 15]           1,024
           Conv2d-41         [-1, 512, 20, 15]         131,584
         Residual-42         [-1, 512, 20, 15]               0
           Conv2d-43         [-1, 512, 20, 15]       2,359,808
      BatchNorm2d-44         [-1, 512, 20, 15]           1,024
           Conv2d-45         [-1, 512, 20, 15]       2,359,808
      BatchNorm2d-46         [-1, 512, 20, 15]           1,024
         Residual-47         [-1, 512, 20, 15]               0
AdaptiveAvgPool2d-48           [-1, 512, 1, 1]               0
          Flatten-49                 [-1, 512]               0
           Linear-50                   [-1, 7]           3,591
================================================================
Total params: 11,176,839
Trainable params: 11,176,839
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 1.17
Forward/backward pass size (MB): 305.87
Params size (MB): 42.64
Estimated Total Size (MB): 349.68
----------------------------------------------------------------
```

Figure 8 ResNet18 Model Architecture

Training

The training was conducted on my personal gaming computer with the following specs:

Spec	Value
CPU	11th Gen Intel i7-11700 @ 2.5 GHz
GPU	Nvidia GeForce RTX 3060 Ti with 8 GB RAM
RAM	16 GB

Table 1 Training computer hardware specs

Keras model

To the shock of absolutely no one, the Keras model with its absolutely garbage architecture did not learn a single bit of useful insight from the data. It began overfitting at epoch 9 and early stopped at epoch 19, ending up with worse than random guess accuracy at the end of training.

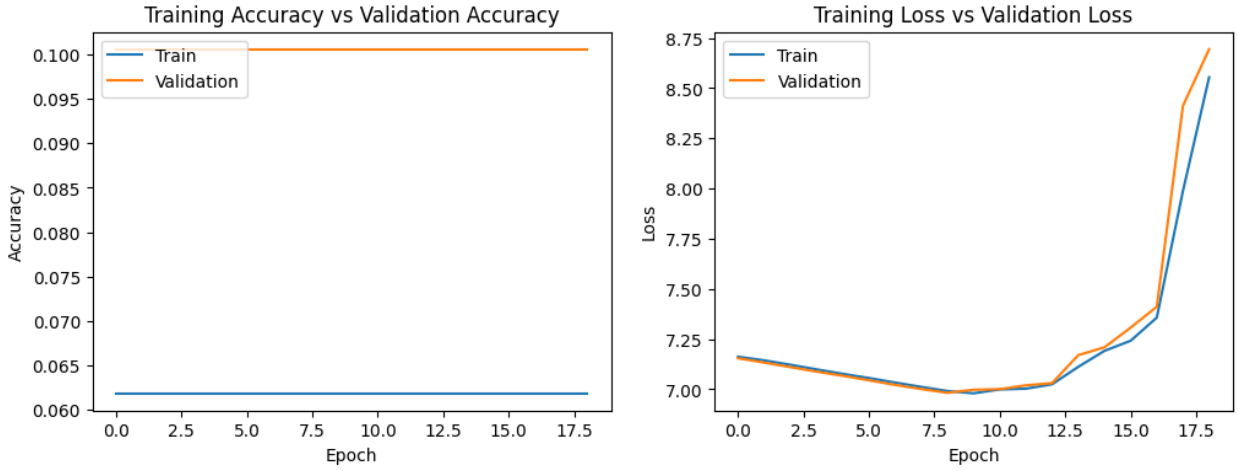

Figure 9 Keras model attempting to train and failing miserably

The only useful bit of insight learned from training this particular model was the realization of how taxing the training was going to be on GPU memory. This model was only able to train after the following measures were taken:

- Reducing batch size from 64 to 8
- Moving some parts of the model off the gpu
- Closing some open tabs, especially the one with the Project Zomboid map [5]

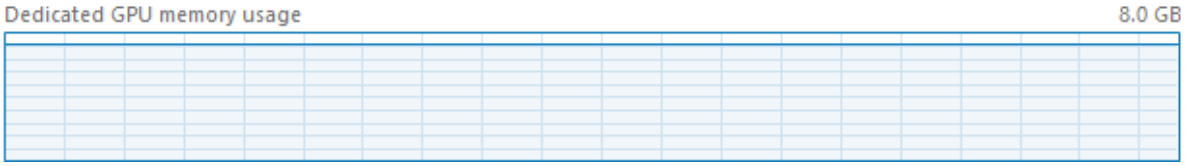

Figure 10 Keras Model GPU Memory usage

Pytorch model

The Pytorch model was not only able to train, it was also able to achieve a 91 % peak validation accuracy on Epoch. However, with the realization that the data was heavily imbalanced, this figure seems much less impressive than it otherwise would be. Even still, the fact that my hardware was able to train this model at all is impressive.

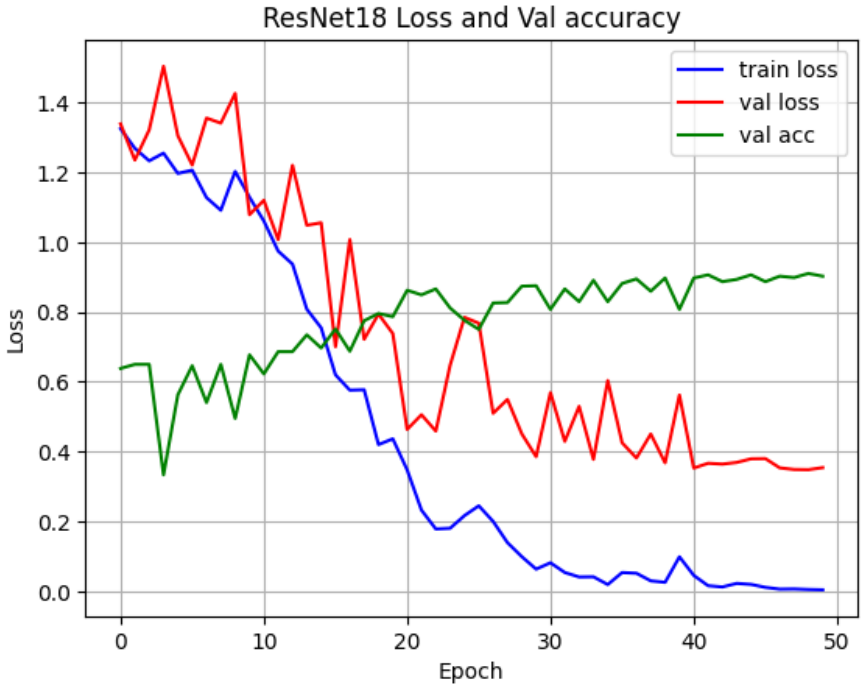

Figure 11 Pytorch model losses and validation accuracy

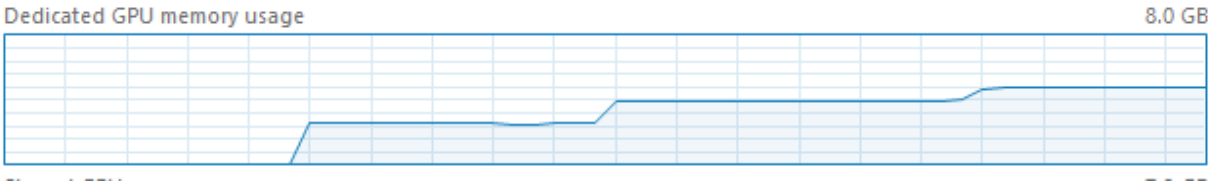

Figure 12 Pytorch model GPU Memory usage

Unfortunately, I was not able to extrapolate classification reports and confusion matrices for the entire training and validation sets due to further issues with GPU memory after loading a previous model and running inferences on it. I have, however, managed to compile a classification report and confusion matrix for 48 random samples of the validation set using the highest accuracy model.

	precision	recall	f1-score	support
anger	1.00	1.00	1.00	1
disgust	1.00	1.00	1.00	2
fear	1.00	1.00	1.00	2
happiness	1.00	1.00	1.00	7
neutral	1.00	1.00	1.00	30
sadness	1.00	1.00	1.00	2
surprise	1.00	1.00	1.00	4
accuracy			1.00	48
macro avg	1.00	1.00	1.00	48
weighted avg	1.00	1.00	1.00	48

Table 2 Classification report of 48 random samples from validation set

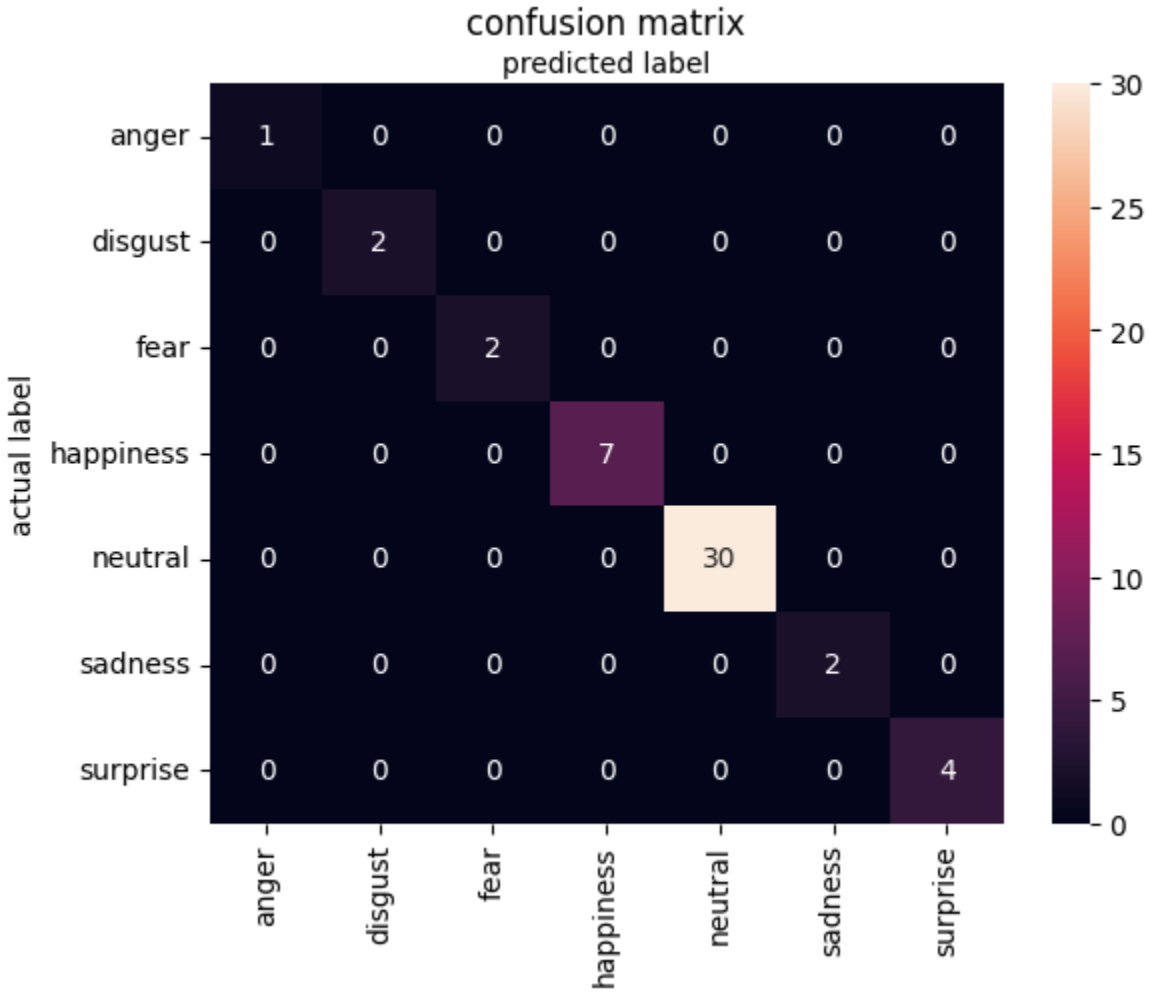

Table 3 Confusion matrix of 48 random samples from validation set

Real-time implementation

In lieu of a test dataset, the Pytorch model was tested using my logitech webcam by using the OpenCV library. These webcam images were preprocessed to the same format (640x480, grayscale, normalized) used by the model, with all results, baring one correctly labeled as anger, returning as sadness. This is particularly odd as the dataset was heavily skewed toward neutral samples, which I would have expected to be the majority of returns from the model. Refer to above confusion matrix for order of scoring categories.

Figure 13 attempt at portraying happiness

```
[ 4.2759376   0.22264926 -0.7248371  -4.86818    -1.6287544   5.477323
 -1.8054811 ]
predicted label: sadness
```

Figure 14 Model prediction scores and labels of figure 13

Figure 15 attempt at portraying sadness

```
[ 2.318191   1.5910375 -1.3776926 -3.8943267 -0.871484   3.8717034
 -0.6926903]
predicted label: sadness
```

Figure 16 Model prediction scores and labels of figure 15

Figure 17 attempt at portraying anger

```
[ 5.8157897  -0.46454358  0.03104487 -4.4797034  -2.279106    5.529242
 -3.0647838 ]
predicted label: anger
```

Figure 18 Model prediction scores and labels of figure 17

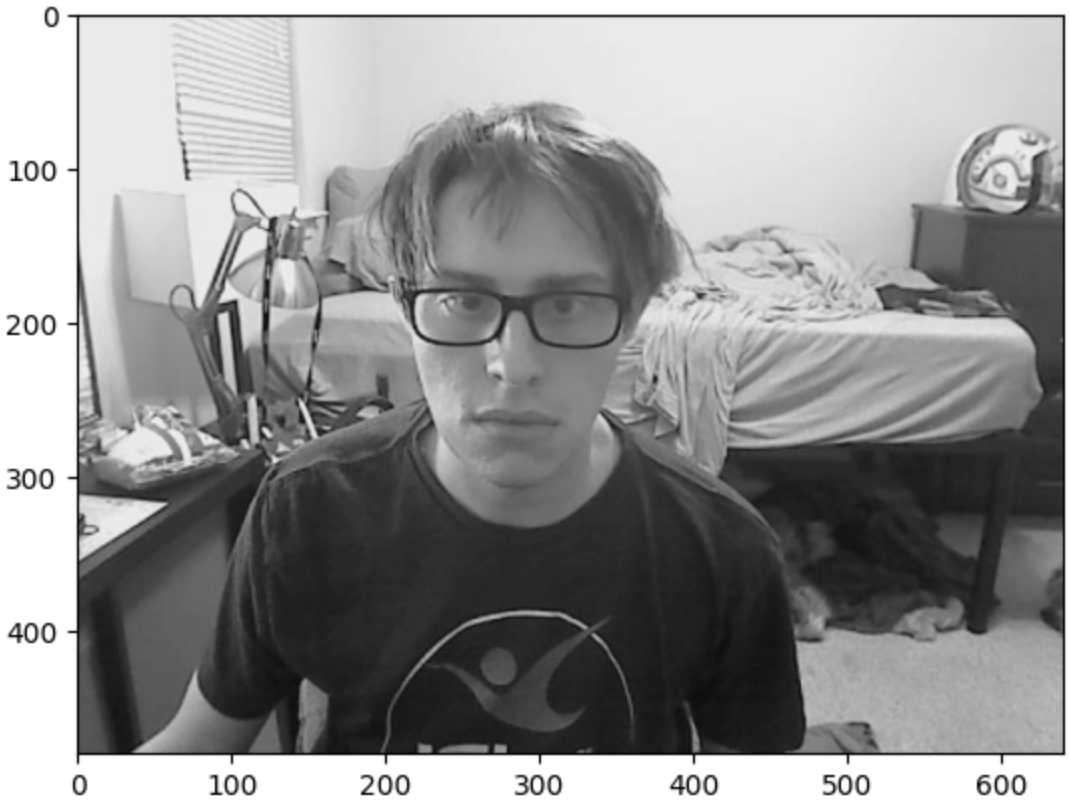

Figure 19 attempt at portraying neutral

```
[ 3.1010463   0.4604013  -0.54174304 -3.7138517  -0.53447896  3.81029
 -1.6490252 ]
predicted label: sadness
```

Figure 20 Model prediction scores and labels of figure 19

Conclusions

Even though this model did not ultimately work as intended, there have been some important takeaways from this experience. Firstly, before I trained this model I did not believe there was such a thing as having too much data when it came to machine learning, and training this model has completely refuted that by pushing my hardware to its very upper limits (hopefully no long-term damage was done, I still love using it to play games). Second, quality of data is just as important as the quantity of it, particularly with this dataset being heavily imbalanced it likely made the model effectiveness much worse. Lastly, gaining more experience with pytorch in general will almost certainly be useful in the future.

Sources

[1] Therrien, L. (2022, August 10). *Lazer's remote emotion detector*. GitHub. https://github.com/lucastherrien/lazer-s_remote_emotion_detector

[2] "Caltech 10K Web Faces." *Perona Lab - Caltech 10k Web Faces*, www.vision.caltech.edu/datasets/caltech_10k_webfaces/. Accessed 9 May 2023.

[3] P. Lucey, J. F. Cohn, T. Kanade, J. Saragih, Z. Ambadar and I. Matthews, "The Extended Cohn-Kanade Dataset (CK+): A complete dataset for action unit and emotion-specified expression," 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Workshops, San Francisco, CA, USA, 2010, pp. 94-101, doi: 10.1109/CVPRW.2010.5543262.

[4] Spenceryee. (n.d.). Spenceryee/CS229. GitHub.

https://github.com/spenceryee/CS229/tree/master

[5] Schneider, B. (n.d.). Project Zomboid Map Project.

https://map.projectzomboid.com/