Emotion Detection

Vashie Garan

March 2021

1 Introduction

In this project, Pytorch is used to train the model in order to generalize the model to predict emotion given an image or live feed video. Further in the chapter we would be discussing about the architecture of the machine learning model .

2 Data Preprocessing

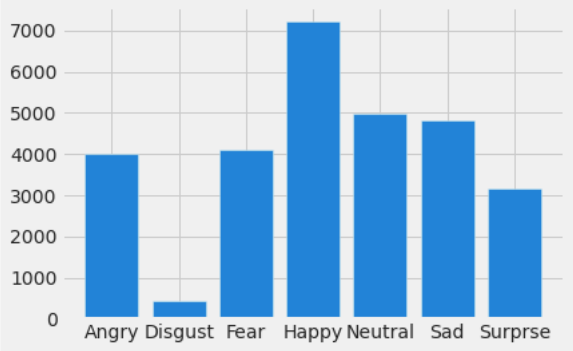

Figure 1: The graph shows the number of data.

```
In [91]: # Data transforms (gray scaling and data augmentation)

training = tt.Compose([tt.Grayscale(num_output_channels=1),
                       tt.RandomHorizontalFlip(),
                       tt.RandomRotation(40),
                       tt.ToTensor()])

validation = tt.Compose([tt.Grayscale(num_output_channels=1),
                       tt.ToTensor()])
```

Figure 2:

In Figure 2, the colour of images are changed into gray colours where the number of output channels is 1 since 1 represent black and white colour.

```
def get_default_device():
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        return len(self.dl)
```

Figure 3:

In Figure 3, The first function is used to detect if you have a 'Cuda' enabled GPU available. Second is used to transfer my model to GPU memory. The class is used to transfer my data(Data Loaders) to GPU memory.

3 Training the model

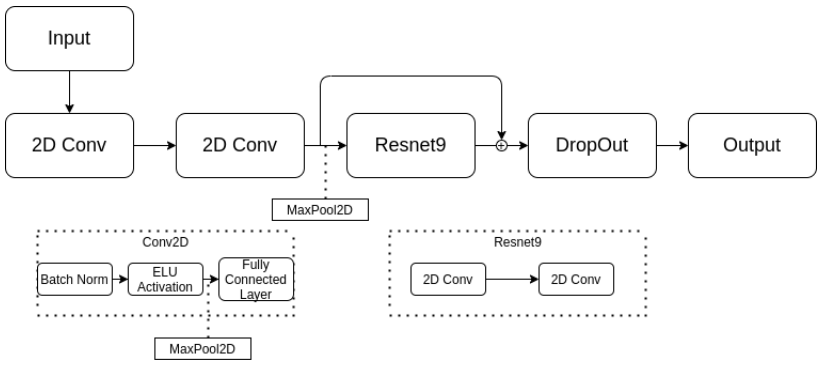

Figure 4: CNN Architecture

```
In [13]: def conv_block(in_channels, out_channels, pool=False):
             layers = [nn.Conv2d(in_channels, out_channels, kernel_size=3, padding=1),
                       nn.BatchNorm2d(out_channels),
                       nn.ELU(inplace=True)]
             if pool: layers.append(nn.MaxPool2d(2))
             return nn.Sequential(*layers)

         class ResNet(ImageClassificationBase):
             def __init__(self, in_channels, num_classes):
                 super().__init__()

                 self.conv1 = conv_block(in_channels, 128)
                 self.conv2 = conv_block(128, 128, pool=True)
                 self.res1 = nn.Sequential(conv_block(128, 128), conv_block(128, 128))
                 self.drop1 = nn.Dropout(0.5)

                 self.conv3 = conv_block(128, 256)
                 self.conv4 = conv_block(256, 256, pool=True)
                 self.res2 = nn.Sequential(conv_block(256, 256), conv_block(256, 256))
                 self.drop2 = nn.Dropout(0.5)

                 self.conv5 = conv_block(256, 512)
                 self.conv6 = conv_block(512, 512, pool=True)
                 self.res3 = nn.Sequential(conv_block(512, 512), conv_block(512, 512))
                 self.drop3 = nn.Dropout(0.5)

                 self.classifier = nn.Sequential(nn.MaxPool2d(6),
                                                 nn.Flatten(),
                                                 nn.Linear(512, num_classes))

             def forward(self, xb):
                 out = self.conv1(xb)
                 out = self.conv2(out)
                 out = self.res1(out) + out
                 out = self.drop1(out)

                 out = self.conv3(out)
                 out = self.conv4(out)
                 out = self.res2(out) + out
                 out = self.drop2(out)

                 out = self.conv5(out)
                 out = self.conv6(out)
                 out = self.res3(out) + out
                 out = self.drop3(out)

                 out = self.classifier(out)
                 return out
```

Figure 5:

ResNet9 One of the major things of my CNN model is the addition of the residual block, which adds the original input back to the output feature map obtained by passing the input through one or more convolutional layers.

- Conv2d - For all my convolutional layers I have used the same parameters except for the number of input channel and output channels. The kernel size is fixed at 3 and padding at 1.
- BatchNorm2d — I have used batch normalisation since at times the output from a convolution layer could be pretty large, which could affect our model negatively. Hence, Batch normalisation does just that, it normalisation's the values from the previous layer.
- ELU — Exponential Linear Unit is an activation function that tends to converge cost to zero faster and ELU has an extra alpha constant which should be a positive number. I used both ReLU and ELU, but found ELU to work best with my data.
- MaxPool2d — It is used to reduce the chances of the model from overfishing. What it does is, it takes an integer value(e.g. 4) and then takes

a block of 4 pixels, starting from one end and takes the highest value out of those 4 pixels and then replaces that single value with the whole block, thus, reducing the size of the tensor.

- Dropout — It is mainly used for better generalisation of the models, since it has a probability of dropping any channel from the tensor, which results in a more generalised model. The probability can be configured by providing a parameter.
- Flatten — It does what is say, it flattens the tensor from the previous layer into one-dimension. Since the output from convolutional layers and the max-pooling layers are all multi-dimensional, and we can't perform prediction from multi-dimensional tensor, hence, we use flatten
- Linear — The last layer of my model was a linear layer on which will decrease the size of my tensor to the size of the output labels and give me the probability of each class label being true for that image

4 Accuracy

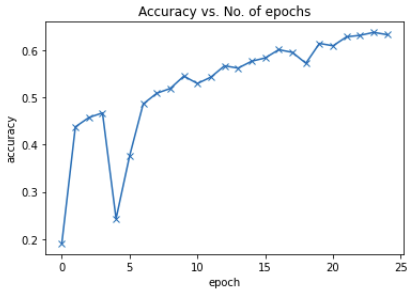

Figure 6:

This graph shows how my accuracy fluctuated when the learning rate was high, and then went up gradually as the learning rate became lower and lower

5 Result

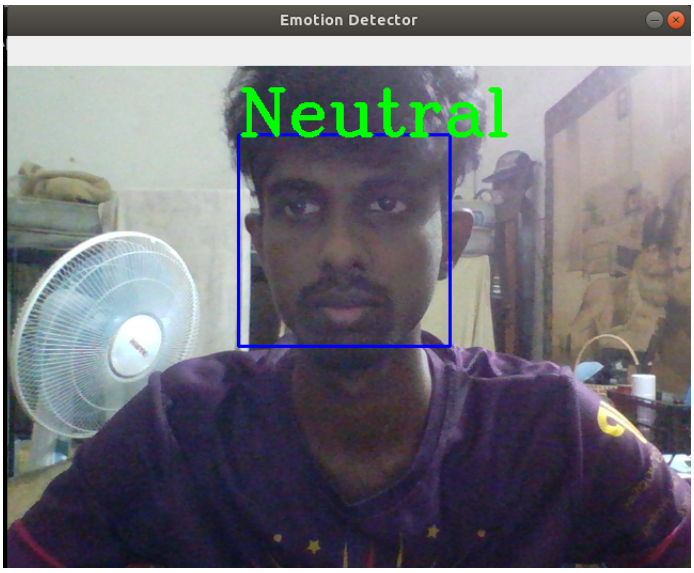

Figure 7:

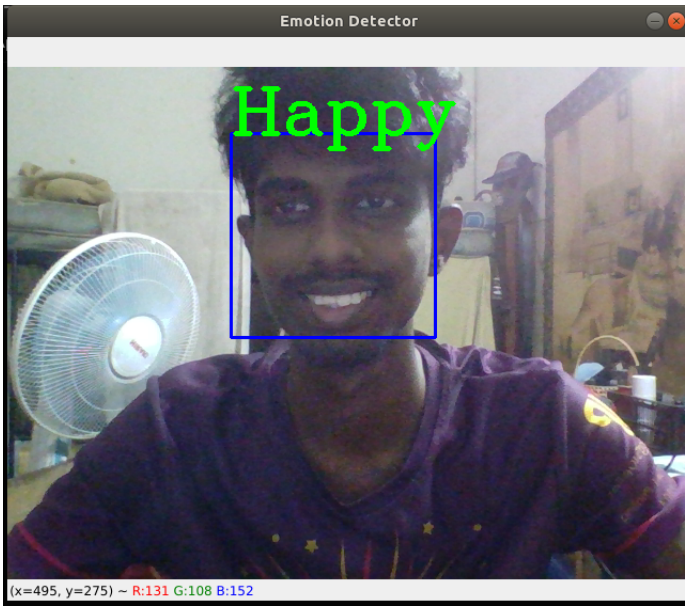

Figure 8:

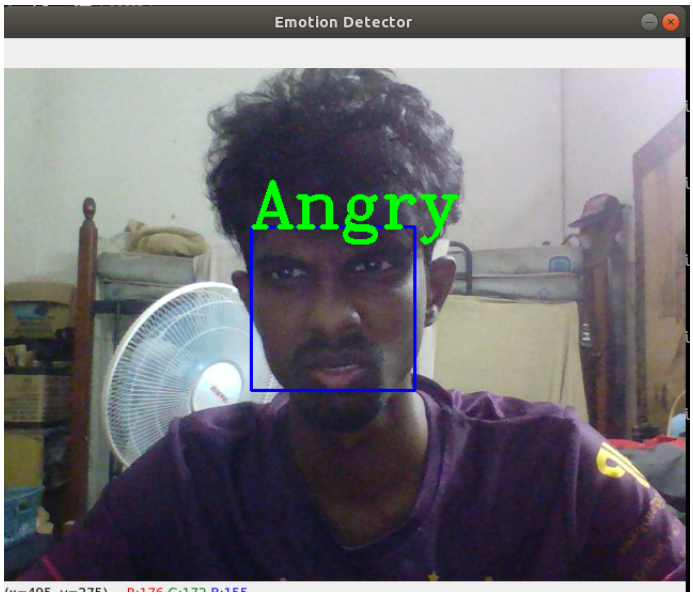

Figure 9:

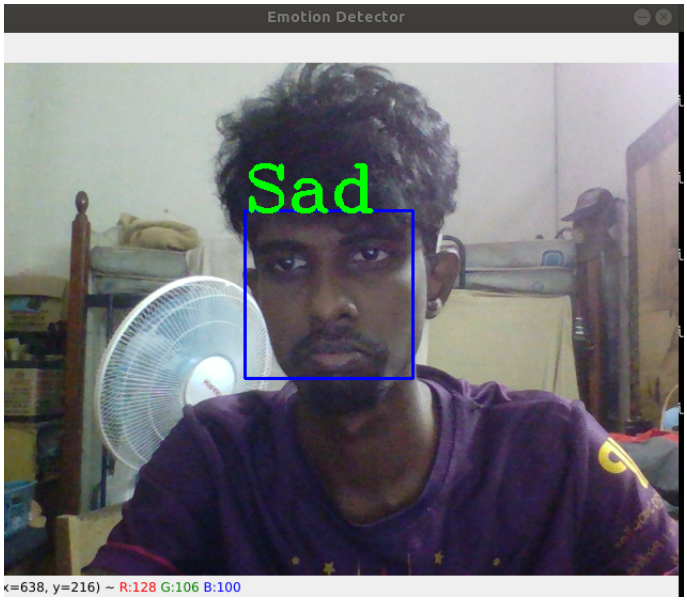

Figure 10:

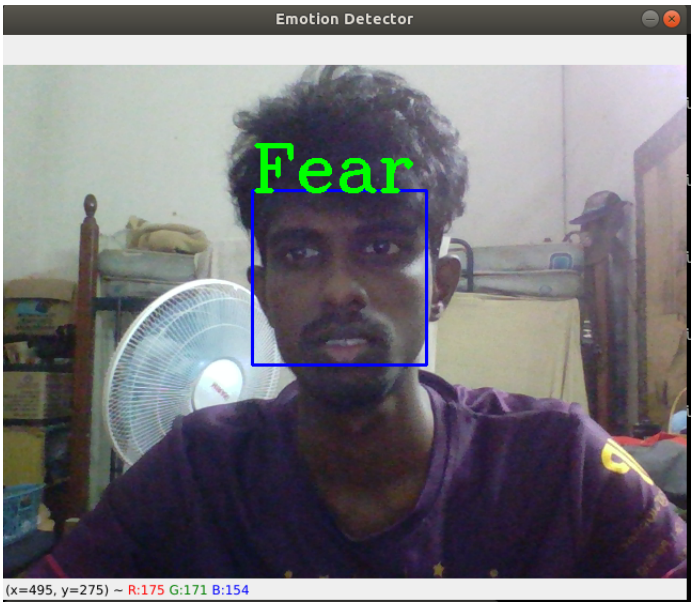

Figure 11: