National Institute of Technology Karnataka, Surathkal

Audio Data Augmentation using GANs

Submitted by:

Aditya S Gourishetty (171EE103)

Dwijesh Athrey K (171EE116)

M Nikhil Bharath (171EE223)

Supervisor: Dr. Debashisha Jena

Department of Electrical and Electronics Engineering

Contents

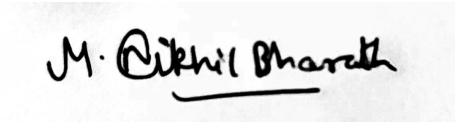
1 Declaration

We hereby declare that the project entitled "Audio Data Augmentation using GANs" was carried out by us during the VIII Semester of the academic year 2020-2021. We declare that this is our original work and has been completed successfully according to the direction of our guide Dr.Debashisha Jena and as per the specifications of NITK Surathkal.

Place: NITK,Surathkal
Date: 22-04-2021

Aditya S Gourishetty (171EE103)

Dwijesh Athrey K (171EE116)



M Nikhil Bharath (171EE223)

2 Acknowledgement

Firstly we would like to thank Dr. Debashisha Jena, Dept. of Electrical and Electronics for his prolonged support which was crucial for driving this project to its completion. We are also immensely grateful to Dr.Debashisha Jena for giving his time for providing his profound insights on our initial and intermediate results and accordingly suggesting changes to our methods to get better results without which our project could not have arrived at this stage.

We are also thankful to Electrical and Electronics Department, National Institute of Technology Karnataka for providing us the platform to learn and acquire the relevant skills and for giving the opportunity to be a part of such projects.

Lastly, we thank our families and friends for providing us with constant support throughout the course of the project.

3 Abstract

Modern neural network architectures have been growing bigger and more complex by the day to suit and solve the various problems that the real world poses. It is not uncommon for the number of parameters in these state of the art networks to be hundreds of millions or even run into billions. These extremely large networks require a lot of data to work with and be trained on, and original data may not be enough for them to be sufficiently well trained, especially if there is imbalance in the number of data samples in each class the dataset has. A common solution to solve this problem is to augment the data by techniques such as rotation, cropping and flipping in case of images. However with the development of a new class of neural networks known as the Generative Adversarial Networks (GANs), a new path has been explored for data augmentation. GANs can be used to generate fake images resembling the real dataset, and these fake images can be added to the original dataset to increase the number of data samples and reduce imbalance in the dataset. In this project, we take up the problem of speech emotion recognition, and aim to improve the solutions to this classification problem by augmenting the data using GANs.

4 Introduction

Data Augmentation refers to the process of increasing diversity in data without collecting any more real data to improve the accuracy of the training model. This is done by artificially creating new data samples from the available data through various techniques. Some of the commonly used transformational techniques applied to the images include geometric transformations such as Cropping, Rotation, Scaling, Flipping, Translation, and color space transformations such as noise injection, color casting and Varying brightness . In multi-class classification, data augmentation is mainly used to curb the data imbalance problem that arises due to the distribution of data across the classes being skewed. It also greatly helps in reducing overfitting which is in turn a common problem in large neural networks.

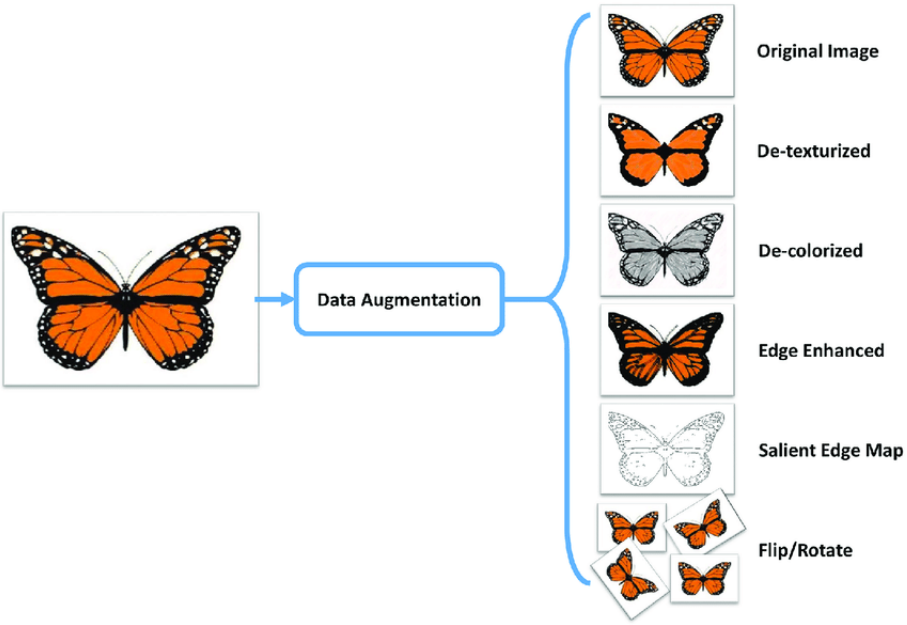


Figure 1: Common techniques of data augmentation

This project revolves around applying data augmention using GAN's which is a deep learning based technique, on a speech emotion classification dataset. Speech Emotion Recognition (SER) is the process of attempting to recognize human emotive states from speech. We know that the fundamental emotions from a voice sample can be captured through its underlying tone and pitch. This is the parameter that SER algorithms use in order to detect the emotion. Speech emotion recognition is a challenging task simply because emotions are subjective. Humans can effectively perform this task as a natural part of speech communication, the ability to conduct it automatically through various machine learning methodologies is still an ongoing subject of research.

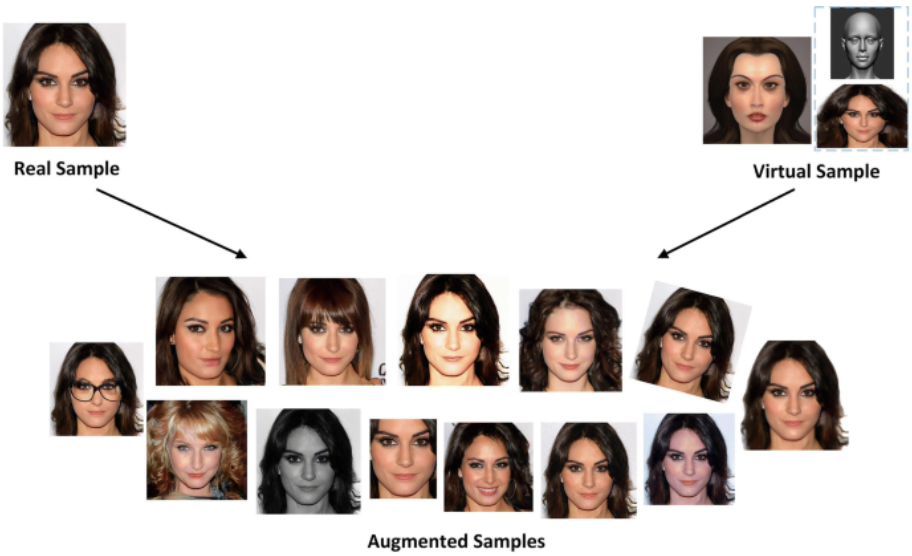


Figure 2: GAN based data augmentation

4.1 Generative Adversarial Networks

Generative Adversarial Networks (GANs), which were first developed by Ian Goodfellow in 2014, are one of the most inventive deep learning based data augmentation techniques that are in use. GANs are architectural networks that use two neural networks, competing against each other (thus the "adversarial") in order to generate new, artificial instances of data from the real data. The objective is to generate data which look authentic to human observers i.e. as real as the original data. They are extensively used in image generation, video generation and voice generation. The network algorithm consists of two parts:

1. Generator network: The generator model generates new images by taking in a fixed-length random noise vector as input. The generated fake images then go in as input to the discriminator network.

2. Discriminator network: The discriminator model takes the generated image as input and classifies the image as real or fake depending on how far it has learnt from the training.

4.1.1 Discriminator training

The training process is separate for both the networks. Given a random noise vector to an untrained generator with random weight initialization, the generator then generates a set of images which are very random and hence fake images. These fake images labelled '0' are then combined with a set of real images ('1') taken from the training dataset. This combined dataset is given as inputs to the discriminator and the network is trained just like a normal binary classifier wherein the weights are updated to get better at discriminating real samples from fake samples. The cost function used here is the Binary Cross Entropy loss function (BCE Loss) which is predominantly used for binary classification. It computes how poorly our model has performed by comparing the actual output with the predicted output. The output of this loss function is a value between 0 and 1 which is indicated through a single node. We then apply sigmoid function to this output in order to classify it into either of the two classes. Depending on the difference between the actual output and the predicted output, the loss function penalizes the network model accordingly.

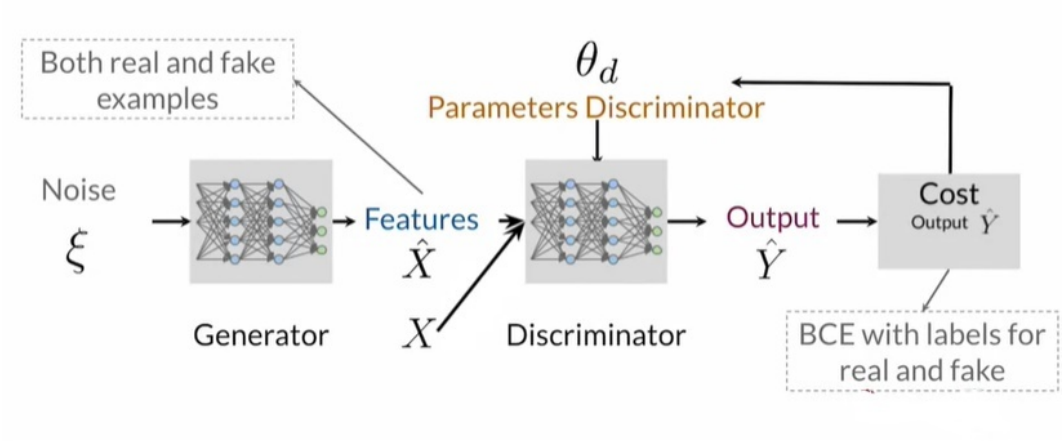


Figure 3: Discriminator Training

4.1.2 Generator training

Now, the untrained generator is again made to generate images through random weight initialization (as before) with random noise vectors given as input. The generated images then are given as inputs to the discriminator which then classifies these images as real or fake depending on how much it has learnt from previous trainings. Based on the predicted outputs and the real outputs, BCE loss function then calculates the error and the generator weights are updated accordingly. One of the core aspects of generator training is the transposed convolution that upsamples the input feature map to an output feature map as per the requirement. Usually this is done by halving the depth while doubling the height and width of the previous layer using a stride of 2.

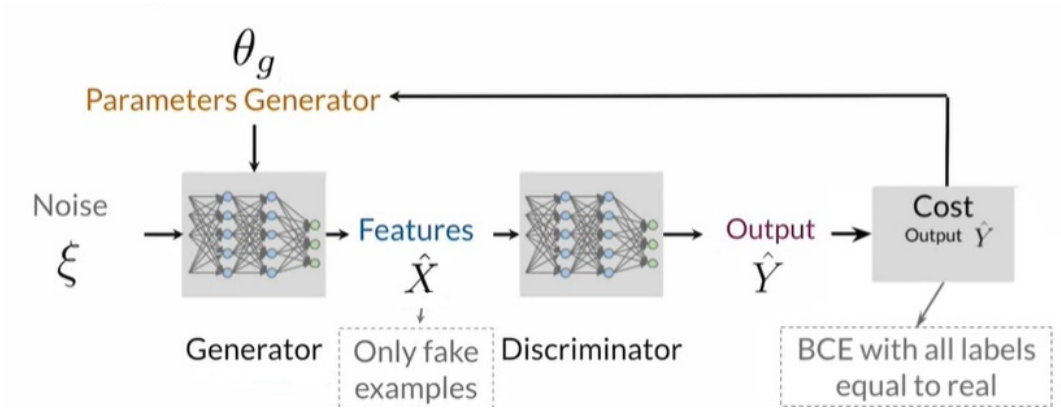
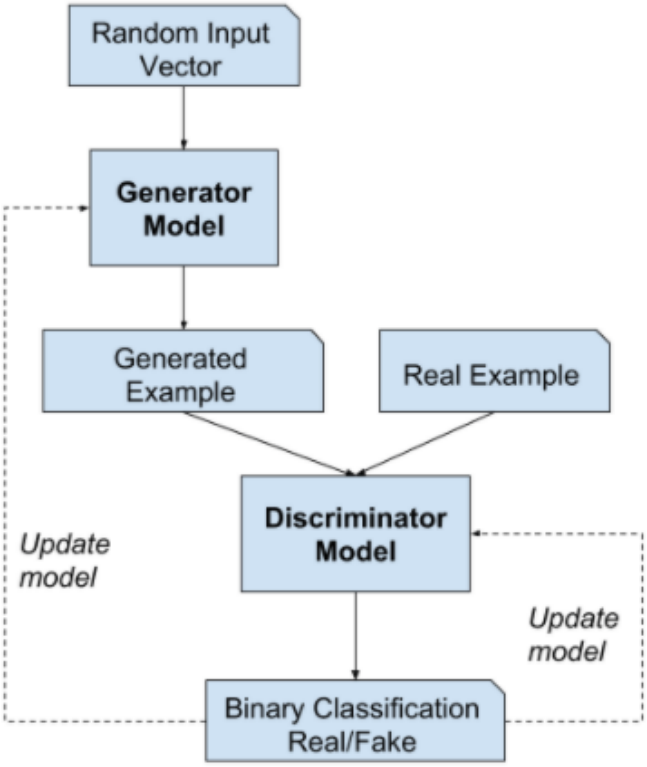


Figure 4: Generator Training

4.1.3 Flowchart

Both the discriminator and generator are trained together in an alternate fashion. However, based on situations, the ratio can be changed. Importantly, the error and accuracy in both the networks have to be along a similar trajectory. Anything otherwise can cause abnormalities in generating new images. The training is said to be complete when error or accuracy of both the generator and discriminator reaches a particular threshold. Once the training is done, given a random noise vector as input, the generator generates images that look as real as the original ones.



4.2 Classification Task

4.2.1 Transfer learning

Transfer learning in CNN refers to the process using an already trained network as a starting point for training a new network. This pre-trained network contains weights that are stored and are subsequently updated in the new training. Some of the major applications where they are extensively utilized are deep neural networks such as computer vision and natural language processing as these usually involve high computation and are time consuming. In other words, instead of training afresh with randomly initialized weights this transfer learning uses pre-trained weights of a different network hence acting as an optimization technique to speed up the learning process in the new network.

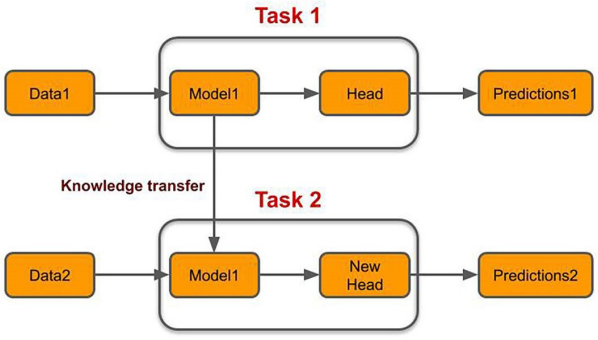


Figure 5: Transfer learning

4.2.2 ILSVRC

ILSRVC or ImageNet large scale visual recognition challenge is a computer vision competition held every year based on a publicly available dataset called ImageNet with the goal being to

promote and develop computer vision architectures. This dataset contains a large collection of human annotated photographs developed over the years by various academics. Precisely, it contains about 22 million images approximately distributed roughly across 22,000 categories. ILSVRC uses a subset of this dataset for developing its algorithms. The objectives of this challenge mainly include image classification, single-object localization and object detection. The competition has paved the way for some of the most important and ground-breaking algorithms that are in use today. VGG16 and ResNet50 are two of the many algorithms that were developed in the year 2014 and 2015 respectively. Below is a graph pertaining to classification error of the winning algorithms over the years.

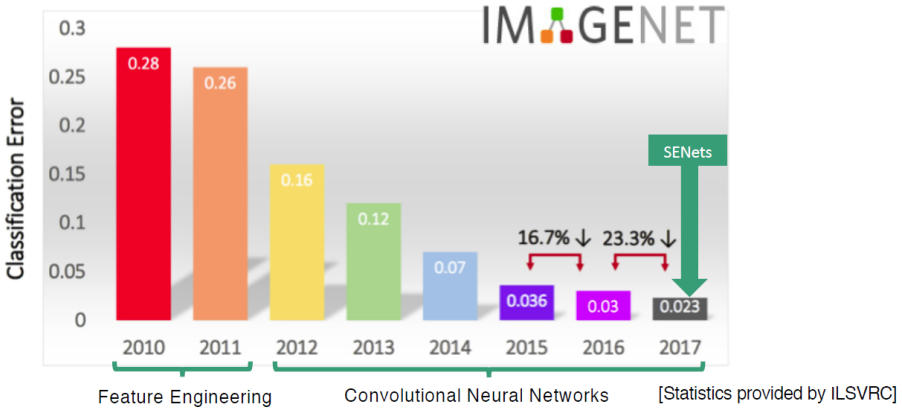


Figure 6: Classification results of winning algorithms in ILSVRC over the years

4.2.3 VGG16 Network Model

VGG16 is a state-of-the-art deep convolution neural network architecture trained over ILSVRC. It was adjudged as the winning algorithm in the year 2014 [10] and is considered to be one of the best CNN architecture developed in recent times. VGGNet is a deep neural network consisting of 138 million parameters, which can be a bit challenging to handle.

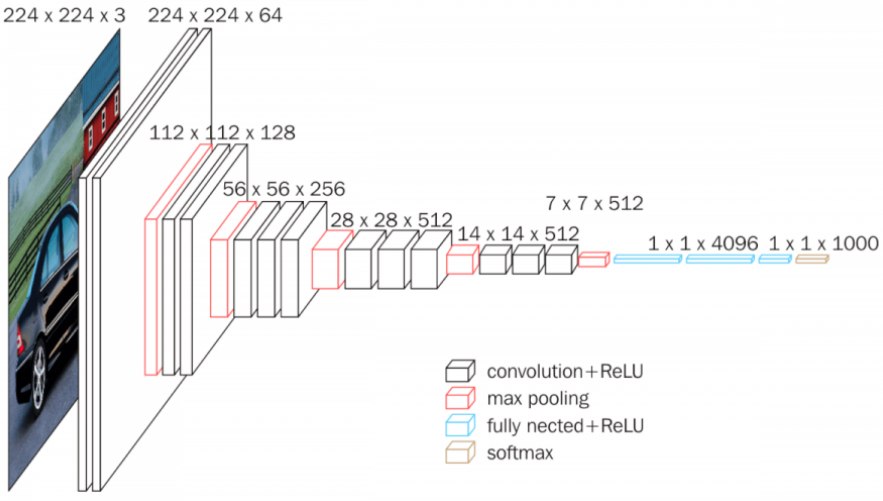


Figure 7: VGG16 Architecture diagram

The input to this network is 228x228 size RGB image i.e. (224,224,3). The first two layers are the convolutional layers that contain 64 channels with 3*3 filter size and same padding. It is then followed by a max pool layer of stride (2, 2), two convolution layers with filter size (3, 3) which is in turn followed by a max pooling layer of stride (2, 2) which is the same as the previous layer. Following it are 2 convolution layers of filter size (3, 3) and 256 filters. Moving forward, there are 2 sets of three convolution layers and a max pool layer with each having 512

filters of (3, 3) size with the same padding. Same padding (padding of 1-pixel) is done after each convolution layer to stop the spatial feature of the image.

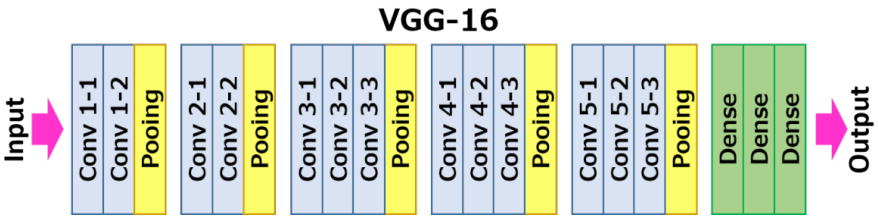


Figure 8: VGG16 architecture map

After the stack of convolution and max-pooling layer, we get (7, 7, 512) feature map as shown in figure 8. We then flatten this output to make it a (1, 25088) feature vector which is then followed by three fully connected layers with the first two having 4096 channels eac and the third performing 1000-way classification, thus containing 1000 channels. The final layer is the soft-max layer. All the networks have the same configuration for their fully connected layers. The activation function used in all the hidden layers is ReLU as it is more computationally efficient compared to others hence resulting in faster learning.

4.2.4 ResNet50 Network Model

Residual networks (ResNet) is another convolutional neural network which allows extremely deep networks to train without the problem of vanishing gradient. This too was developed at ILSVRC using the ImageNet dataset [11]. It was declared as the winner at ILSVRC 2015. Here 50 refers to the number of layers the network has which includes 48 convolutional layers, 1 MaxPool layer and 1 Average Pool layer.

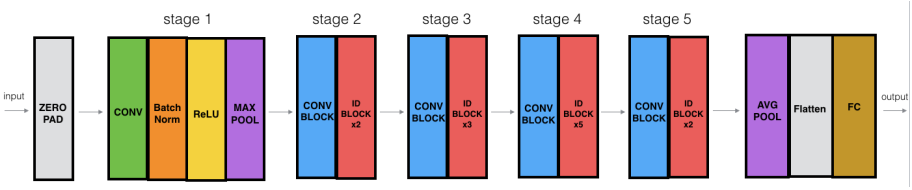


Figure 9: ResNet50 architecture map

The network architecture of ResNet50 has 4 convolutional stages which forms its core as shown in the diagram below. The input given to the network is of the size (224 x 224 x 3). Initially a convolution using 7*7 kernel followed by max-pooling using 3*3 kernel size is done. Following it is the Stage 1 of the network which contains 3 residual blocks containing 3 layers each. The size of kernels used to perform the convolution operation in all 3 layers of the block of stage 1 are 64, 64 and 128 respectively. In the figure below, the curved arrows represent the identity connection while the dashed arrows represent the convolution operation in the residual block is performed with stride 2. This results in the size of input getting reduced to half in the height and width, but the channel width getting doubled. This halving of height and width with doubling of channel width repeats as we progress to subsequent stages. In the end, the network finally has an average pooling layer followed by a fully connected layer of 1000 neurons.

4.3 Spectrograms

A spectrogram, in simple words, is a visual representation of an analog signal, in this case an audio signal. It plots the strength of the signal, generally in terms of amplitude, of different frequencies at any point in time. Usually, the amplitude is represented by the colour, the

brighter the colour, the larger the amplitude. The time represented on the X-axis while the frequency is on the Y-axis. Sometimes, a 3D representation is used, with the Z-axis representing the amplitude.

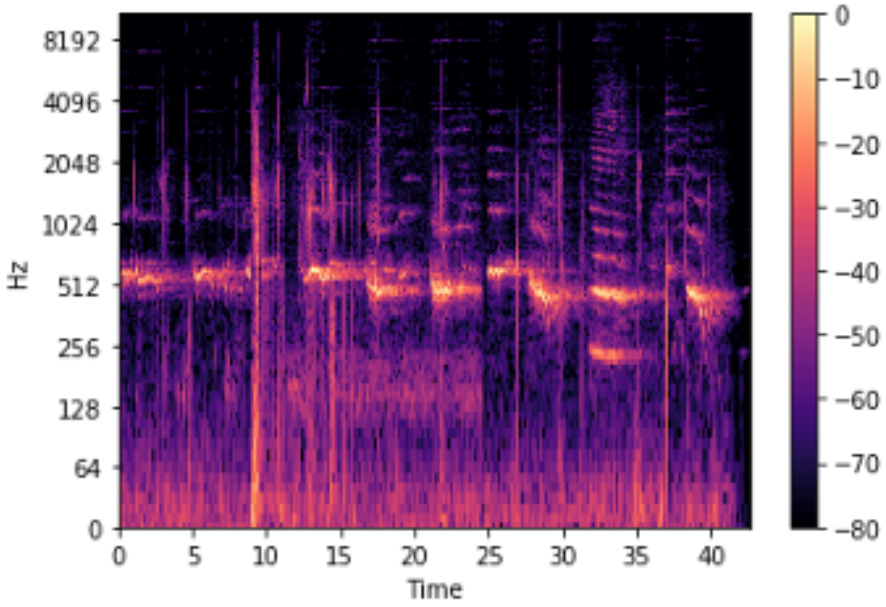


Figure 10: Example of a Spectrogram

4.3.1 Generation of spectrograms

Audio signals can be visualised in multiple methods: one of which is time-domain representation. Here the amplitude of the signal is plotted against time, as shown in the example below:

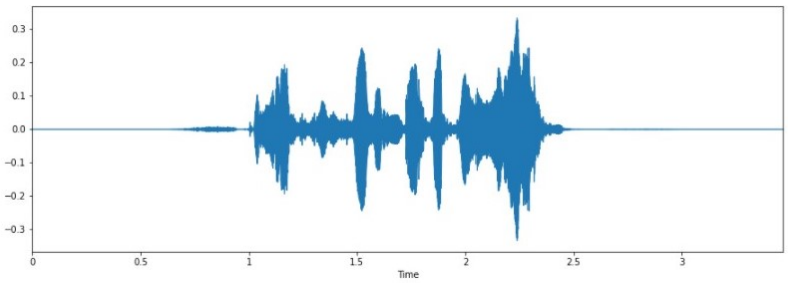


Figure 11: Time-domain Representation of Audio Signal

This representation does not convey information about the frequencies of the components of the audio signal directly. To do that effectively, the Fourier transform can be used. The Fourier transform plots the decomposition of the signal into its component frequencies, but here the temporal aspect is ignored. To get the best of both, a technique known as Short Time Fourier Transform (STFT) is used.

To compute the STFT, the audio signal is divided into windows of smaller length, where the Fourier transform is applied for each section. This is then plotted on a graph to visually represent it. Generally the sounds that humans can hear lie in a small range of frequencies and amplitudes, so instead of a linear scale, a logarithmic scale is used to obtain the spectrogram.

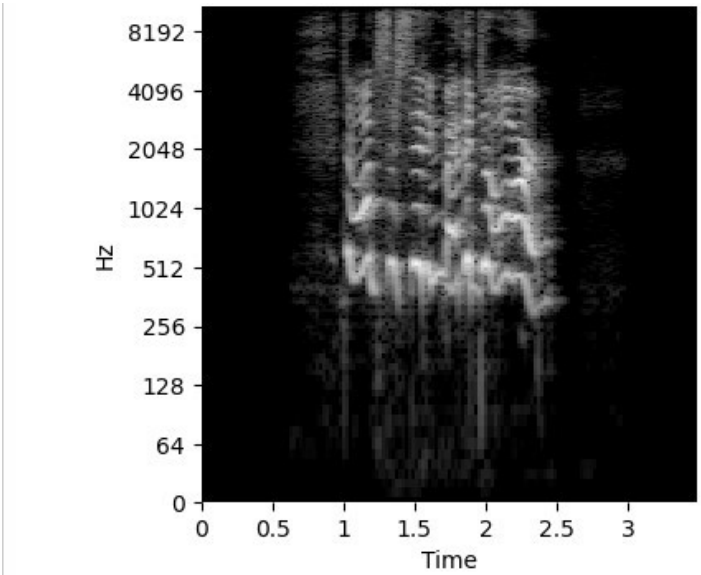


Figure 12: Greyscale Spectrogram of Audio Signal

5 Data preprocessing

5.1 Dataset

The data for the problem of speech emotion recognition has been collected from various sources. The following datasets have been used in our project:

1. The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): This database originally contains 7356 files, which are recordings of 24 professional actors, which includes 12 males and females each, speaking or singing two statements that are lexically matched, in a North American accent. The dataset contains audio and video files of the speeches and songs, but the part used in this project is the audio-only section of the database of speeches, containing 1440 files belonging to 8 classes of emotions: neutral, calm, happy, sad, angry, fearful, disgust, and surprised. For the purpose of this project, the neutral and calm classes have been treated as the same.

2. Toronto Emotional Speech Set (TESS): This dataset consists of a total of 2800 audio files, spoken by two female actors, saying the phrase "say the word _____", the blank being 200 different words, in 7 emotions, namely: neutral, anger, fear, disgust, happiness, sadness, and surprise.

3. Surrey Audio-Visual Expressed Emotion (SAVEE): This dataset has 480 audio files, which are recordings of British English sentences spoken by 4 male actors in 7 different emotions: angry, surprise, disgust, fear, happy, sad and neutral.

4. Crowd-sourced Emotional Multimodal Actors Dataset (CREMA-D): This dataset, as is made clear by the name, is a large dataset of recordings sourced from the public, containing 7442 audio files, recorded by 91 participants of different ethnic backgrounds, speaking in various accents. The dataset is categorized into 6 emotions: happy, sad, fear, disgust, anger and neutral.

For this project, the four above-mentioned datasets have been combined into a single master dataset, so that a considerably large dataset can be obtained and used for our project. The emotion categories of each file can be decoded from the filename, with the decoding process being followed as given in the source of the datasets.

After combining the datasets, an exploratory analysis is performed to understand the dataset. It is seen here that there is a skew in the categories that the audio files belong to, with some emotions such as surprise being underrepresented as compared to the others. This leads us to the conclusion that data augmentation for these categories can help in balancing the dataset and might improve the results of the models that are used for classification.

```
female_angry       1105
female_sad         1101
female_happy       1101
female_fear        1098
female_disgust     1098
female_neutral     1059
male_neutral        839
male_fear           827
male_disgust        827
male_happy          827
male_angry          827
male_sad            827
female_surprise     496
male_surprise       156
```

Figure 13: Number of examples in each category

5.2 Preprocessing

The files in the master dataset are in the .wav format, which is not very conducive for training neural networks on. To make it easier to train the neural network on the dataset, we have converted the audio files into images by using the concept of spectrograms. As explained earlier, spectrograms are visual representations of audio data, created by taking short time fourier transforms of the audio data, and then plotting the frequency values as they vary with time, with the amplitude of the wave being represented by the intensity of colour at that particular point. The frequency is generally on the Y-axis, and time is on the X-axis. The images in the master dataset are converted into spectrograms following the process described in the earlier section. The relevant code snippets are as below:

```
fname = RAV + 'Actor_14/03-01-06-02-02-02-14.wav'
data, sampling_rate = librosa.load(fname)
plt.figure(figsize=(15, 5))
librosa.display.waveplot(data, sr=sampling_rate)


ipd.Audio(fname)
```

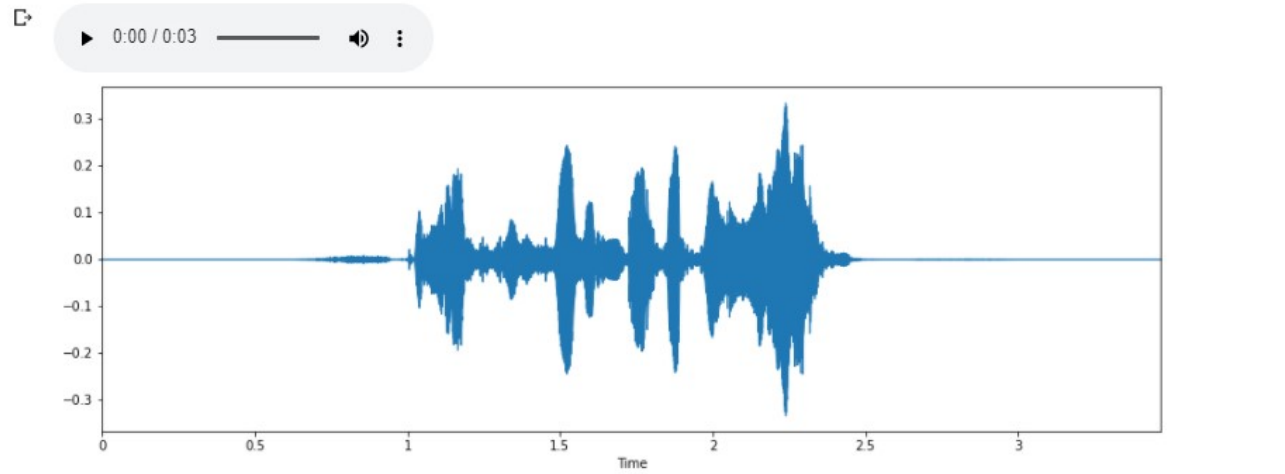


Figure 14: Plotting Time-domain Representation of Audio Signal

```
X = librosa.stft(data)
Xdb = librosa.amplitude_to_db(abs(X))
fig = plt.figure(figsize=(3, 3), dpi=100, frameon=False)
ax = fig.add_axes([0, 0, 1, 1], frameon=False)
#ax.axis('off')
librosa.display.specshow(Xdb, sr=sampling_rate, cmap='gray', x_axis='time', y_axis='log')
```

```
<matplotlib.collections.QuadMesh at 0x7f2e76065350>
```

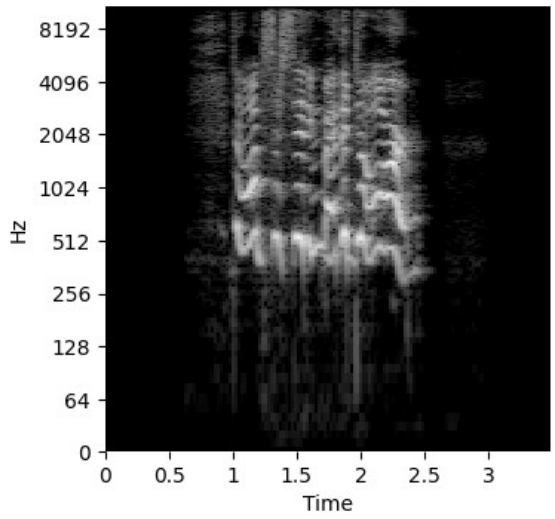


Figure 15: Plotting Greyscale Spectrogram of Audio Signal

This spectrogram plot can be used to identify the type of emotion that is being conveyed in the audio data. The spectrograms are basically images that need to be classified into their categories by the chosen machine learning model.

For this project, spectrograms were generated for all the audio data samples from the master dataset, and then stored in folders as per the emotion category it belongs to. This helps when the dataset has to be loaded as input into the models.

6 Prediction Models and their results:

6.1 GAN

Generator

For the generation of data for augmentation, for each of the classes of the present data with comparatively less data which was causing the imbalance, a Deep Convolutional Generative Adversarial Network (DCGAN) was trained. The architecture for all of them was the same with the difference in resulting weights while training with the different data. Since the training images we were working with were greyscale black and white spectrograms, the number of channels in the images generated by DCGAN was set to 1 and the image resolution was set to 224x224. The size of the input noise vector or the z-latent vector was adequately set to 5.

```
class Generator(nn.Module):
    def __init__(self, ngpu):
        super(Generator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            # Block1
            nn.ConvTranspose2d( nz, ngf * 8, 4, 1, 0, bias=False),
            nn.BatchNorm2d(ngf * 8),
            nn.ReLU(True),
            # Block2
            nn.ConvTranspose2d(ngf * 8, ngf * 4, 4, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 4),
            nn.ReLU(True),
            # Block3
            nn.ConvTranspose2d( ngf * 4, ngf * 2, 6, 2, 1, bias=False),
            nn.BatchNorm2d(ngf * 2),
            nn.ReLU(True),
            # Block4
            nn.ConvTranspose2d( ngf * 2, ngf, 6, 3, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),
            # Block5
            nn.ConvTranspose2d( ngf, ngf, 6, 2, 1, bias=False),
            nn.BatchNorm2d(ngf),
            nn.ReLU(True),

            # Block6
            nn.ConvTranspose2d( ngf, nc, 4, 2, 1, bias=False),
            nn.Tanh()

        )

    def forward(self, input):
        return self.main(input)
```

As shown in the PyTorch Code above, the Generator Architecture contains six blocks of layers, each of the first five blocks containing a Transposed Convolutional Layer, followed by Batch Normalization and a ReLU activation with the sixth block or the final block containing just a Transposed Convolutional Layer followed by a Tanh activation. One can see that the first Transposed Convolutional Layer takes the input noise vector defined by nz which has the value of 5 and performs transposed convolution into ngf x 8 channels or feature maps where ngf is equal to 64. Similarly, in the subsequent blocks the channels are progressively reduced to 1 with the final feature map as the image output. Each of the transposed convolution is performed

with kernel sizes of either 4 or 6 with a stride of 2 and padding 1. All this resulting in the Image dimension of the output being 224x224.

Discriminator

This Generator was trained with a couple of discriminators. Initially with somewhat less deep network, for a slower learning at the start of the training so that the discriminator's performance doesn't overshoot the generators. After some significant decrease in the generator's loss, the training was performed with a high deeper discriminator with a higher performance.

```
class Discriminator1(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            #
            nn.Conv2d(nc, ndf, 6, 3, 2, bias=False,),
            nn.LeakyReLU(0.2, inplace=True),
            #
            nn.Conv2d(ndf, ndf * 4, 6, 3, 2, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            nn.Dropout2d(p=0.4, inplace=False),
            #
            nn.Conv2d(ndf * 4, ndf * 8, 6, 3, 2, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
            #
            nn.Conv2d(ndf * 8, 1, 6, 3, 0, bias=False,),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

The above code in PyTorch is for the less deep Discriminator1 with 4 convolutional layers, with the first 3 followed by batch normalisation and LeakyReLU activation and the final one with sigmoid activation. This low performance discriminator also contains dropout layer before the fourth convolutional layer for the purpose of slowing down the process.

```
class Discriminator2(nn.Module):
    def __init__(self, ngpu):
        super(Discriminator, self).__init__()
        self.ngpu = ngpu
        self.main = nn.Sequential(
            #
            nn.Conv2d(nc, ndf, 4, 2, 2, bias=False,),
            nn.LeakyReLU(0.2, inplace=True),
            #
            nn.Conv2d(ndf, ndf * 4, 4, 2, 2, bias=False),
            nn.BatchNorm2d(ndf * 4),
            nn.LeakyReLU(0.2, inplace=True),
            #
            #
            nn.Conv2d(ndf * 4, ndf * 8, 4, 2, 2, bias=False),
            nn.BatchNorm2d(ndf * 8),
            nn.LeakyReLU(0.2, inplace=True),
```

```
            #
            nn.Conv2d(ndf * 8, ndf * 16, 4, 2, 2, bias=False),
            nn.BatchNorm2d(ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            #
            nn.Conv2d(ndf * 16, ndf * 16, 5, 2, 2, bias=False),
            nn.BatchNorm2d(ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            #
            nn.Conv2d(ndf * 16, ndf * 16, 5, 2, 1, bias=False),
            nn.BatchNorm2d(ndf * 16),
            nn.LeakyReLU(0.2, inplace=True),
            #
            nn.Conv2d(ndf * 16, 1, 3, 2, 0, bias=False,),
            nn.Sigmoid()
        )

    def forward(self, input):
        return self.main(input)
```

This above discriminator is the architecture for the deeper discriminator with seven convolutional layers and no dropout layers.

During training the batch size for the data loader was set to 128 and standard Adam optimizers with learning rates of around 0.00002-0.00005 for the discriminator and 0.00017-0.0002 generator on the BCE Loss.

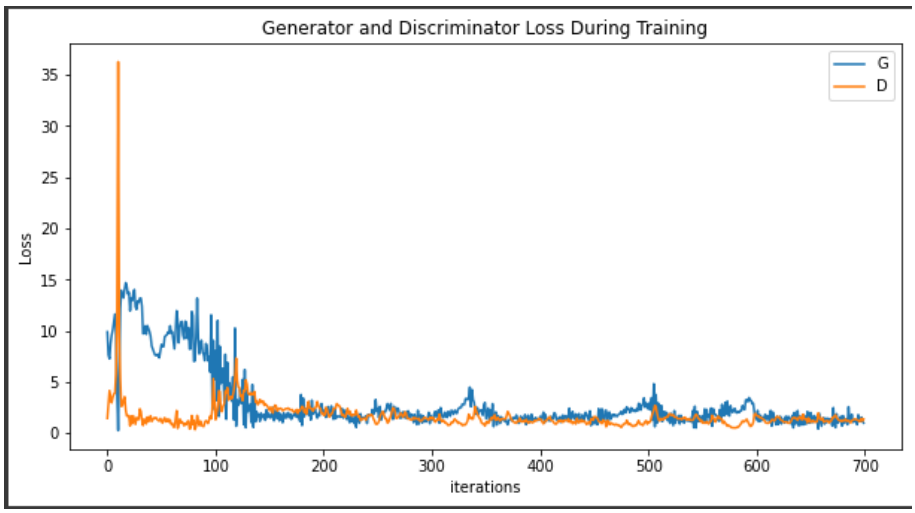


Figure 16: BCE Loss values of Generator and Discriminator during training

As a result, we got both the discriminator loss and generator loss saturate at around 1.1-1.3 with somewhat recognisable resultant generated images.

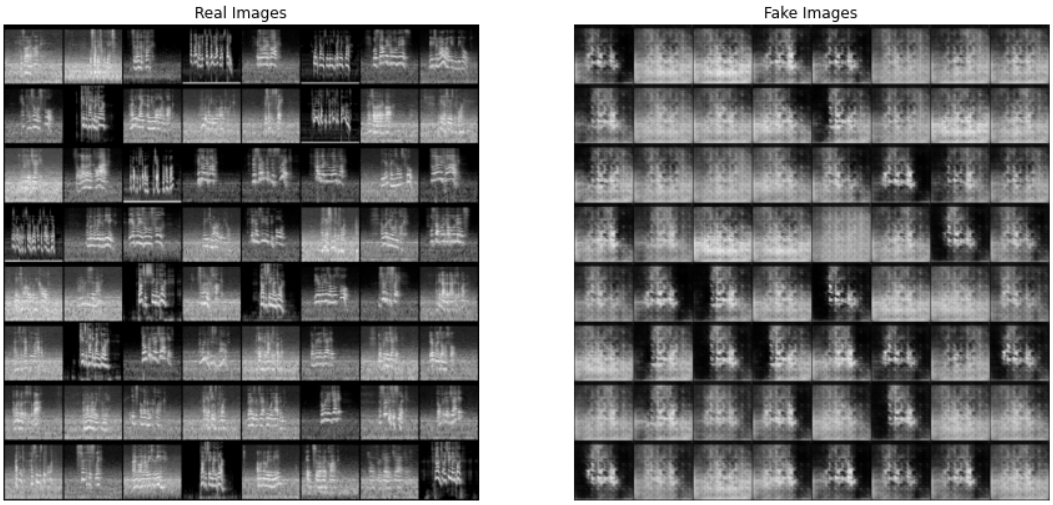


Figure 17: Generated Images

With the addition of these generated images to our original dataset, the distribution of the resultant dataset containing the augmented data has less data imbalance. The new distribution is as shown in the figure.

female_angry	1105
female_sad	1101
female_happy	1101
female_fear	1098
female_disgust	1098
female_neutral	1059
male_neutral	1089
male_fear	1077
male_disgust	1077
male_happy	1077
male_angry	1077
male_sad	1077
female_surprise	896
male_surprise	556

Figure 18: Distribution of the resultant dataset

6.2 Classification

For the task of Speech Emotion Recognition (SER) itself, we trained several models using different techniques. We decided to use transfer learning on a few state-of-the-art CNNs from the past few years of the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) like ResNETs, VGG etc. First, we worked on transfer learning on standard CNNs like ResNet-50 and vgg16 by training only the fully connected layers on them. We trained the Networks using standard Binary Cross Entropy Loss(BCE Loss) same as the one used for training our GAN as it is a multi-label classification task. Binary cross entropy loss calculates the loss for each class for a given sample independently. For each label, the loss is evaluated using the current class against the rest and is very commonly used for classification tasks. For optimisation we used the frequently used Stochastic Gradient Descent.We got validation accuracy varying from 40% to 60% on the models where only the fully connected layers were trained. Later, we got better results on a network based on [2] we trained vgg-16, freezing the weights of around half

the convolutional layers from the original vgg-16 and training the rest along with the fully connected layers. The results of which are shown below.

Dataset	Data	ResNET50_only_fl	VGG_16_only_fl	VGG_16_half_freeze
Original Dataset	Validation	47.8%	67.7%	73.9%
	Training	48.1%	80.2%	87.7%
Augmented Dataset	Validation	52.9%	73.8%	79.3%
	Training	53.6%	83.6%	99.8%

Figure 19: Accuracy of the Networks

For VGG_16_half_freeze weights of the first ten convolutional layers frozen from the original vgg16 and training for the rest and the fully connected layers on our dataset was done. As mentioned in [2] since vgg16 was originially trained on ImageNet data set which is primarily an image dataset, the high level features learned in the last few convolution layers will be different between audio and image data. Since the initial layers of original have been trained for capturing fine grain features of the images, preserving those weights can be useful as it is essential for classification of spectrograms. There is also some addition of dropout layers to reduce the overfitting of the network.

```
class Network(nn.Module):
    def __init__(self):
        super(Network,self).__init__()
        vgg16=models.vgg16(pretrained=True)
        Children = list(vgg16.children())
        Features = list(Children[0].children())
        FeaturesD = []
        FeaturesD.extend(Features[19:21])
        FeaturesD.append(nn.Dropout2d(p=0.4, inplace=False))
        FeaturesD.extend(Features[21:24])
        FeaturesD.append(nn.Dropout2d(p=0.4, inplace=False))
        FeaturesD.extend(Features[24:26])
        FeaturesD.append(nn.Dropout2d(p=0.4, inplace=False))
        FeaturesD.extend(Features[26:])

        #Frozen layers
        self.features1 = nn.Sequential(*Features[0:19])
        #Not Frozen layers
        self.features2 = nn.Sequential(*FeaturesD)
        self.pool = nn.Sequential(Children[1])
        self.classifier = nn.Sequential(
                  nn.Linear(in_features=25088, out_features=4096, bias=True),
                  nn.ReLU(inplace=True),
                  nn.Dropout(p=0.5, inplace=False),
                  nn.Linear(in_features=4096, out_features=4096, bias=True),
                  nn.ReLU(inplace=True),
                  nn.Dropout(p=0.5, inplace=False),
```

```
                    nn.Linear(in_features=4096, out_features=len(class_names),
    bias=True)
                    )
    def forward(self,x):
        x = self.features1(x)
        x = self.features2(x)
        x = self.pool(x)
        x = x.view(-1,25088)
        x = self.classifier(x)
        return x

Net = Network()

for name, p in Net.named_parameters():
        if "Features1" in name:
            p.requires_grad = False
```

```
VGG(
  (features): Sequential(
    (0): Conv2d(3, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU(inplace=True)
    (2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU(inplace=True)
    (4): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (5): Conv2d(64, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (6): ReLU(inplace=True)
    (7): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (8): ReLU(inplace=True)
    (9): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (10): Conv2d(128, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (11): ReLU(inplace=True)
    (12): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (13): ReLU(inplace=True)
    (14): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (15): ReLU(inplace=True)
    (16): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (17): Conv2d(256, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (18): ReLU(inplace=True)
    (19): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (20): ReLU(inplace=True)
    (21): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (22): ReLU(inplace=True)
    (23): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (24): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (25): ReLU(inplace=True)
    (26): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (27): ReLU(inplace=True)
    (28): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (29): ReLU(inplace=True)
    (30): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(7, 7))
  (classifier): Sequential(
    (0): Linear(in_features=25088, out_features=4096, bias=True)
    (1): ReLU(inplace=True)
    (2): Dropout(p=0.5, inplace=False)
    (3): Linear(in_features=4096, out_features=4096, bias=True)
    (4): ReLU(inplace=True)
    (5): Dropout(p=0.5, inplace=False)
    (6): Linear(in_features=4096, out_features=1000, bias=True)
  )
)
```

Frozen Layers

Layers Trained

Figure 20: VGG_16_Half_Freeze

Using this network we got a validation accuracy of around 73.9% on our validation data and training accuracy of upto 87.7% on the original dataset. On augmented dataset, with an improvement of about 5% we got an accuracy of about 79.3% on validation data while an accuracy of 99.8% which as an improvement as a result of the generated images by our GAN.

7 Conclusion:

The ImageNet models VGG-16 and ResNet-50 have been implemented on the spectrograms of the speech dataset for emotion recognition by using the concept of transfer learning, with varying degrees of success. As can be seen from the table in Figure 20, the ResNet-50 model with only the fully connected layers retrained on this dataset was the least successful, with validation accuracies of 47.9% on the unaugmented dataset and 52.9% on the augmented dataset. The VGG-16 network with only the fully connected layers retrained performed better, but not as well as the VGG-16 network with both the last convolutional block and the fully connected layers retrained, which gives us validation accuracies of 73.9% and 79.3% on the unaugmented and augmented datasets respectively.

It is also very evident that the GAN performs the task of data augmentation successfully, with validation accuracy increasing by an absolute value of around 5% for the ResNet-50 network, and around 6% in case of the VGG-16 networks.

This performance can further be improved in future work by building and using a more robust and efficient GAN. There is also scope for reducing overfitting of the model by including more regularization layers. Another path that is open to us is Bayesian networks, which model uncertainty in predictions and reducing overfitting to a large extent.

8 Appendix

Apart form the above-mentioned concepts and implementations, we had attempted to improve it by using a few other concepts as well, most notably Wasserstein GANs and Bayesian CNNs. However they could not be successfully implemented, and hence we have included an overview of them below:

8.1 Wasserstein GAN

To improve the performance of our current GAN, we tried to improve it using a different kind of GAN called the W-GAN[6]. WGAN or Wasserstein GAN is an alternate algorithm to train the generator, using a different kind of competing network from the regular discriminator, called the critic. When training, the generator using the regular discriminator using BCE loss, after the discriminator performance improves the discriminator values tend to reach closer to 1 and 0, while giving progressively smaller values of gradients causing vanishing gradient problems making it difficult for the generator to update its values. Whereas WGAN uses a network called critic which outputs real values not bounded by 0 or 1, in contrast with the discriminator which has a sigmoid activation function at the end ensuring that. This is used along with a loss called the W-Loss or Wasserstein-Loss which measures the difference in the distributions of generator and the discriminator. So, this is the critic we tried working with:

```
class Critic(nn.Module):

    def __init__(self, im_chan=1, hidden_dim=64):
        super(Discriminator, self).__init__()
        self.crit = nn.Sequential(
            self.make_crit_block(3, hidden_dim),
            self.make_crit_block(hidden_dim, hidden_dim * 2),
            self.make_crit_block(hidden_dim*2, hidden_dim * 4, stride = 3),
            self.make_crit_block(hidden_dim*4, hidden_dim * 8),
            self.make_crit_block(hidden_dim * 8, 1, final_layer=True)

        )

    def make_crit_block(self, input_channels, output_channels, kernel_size=6,
    stride=2, final_layer=False):

        if not final_layer:
            return nn.Sequential(
                nn.Conv2d(input_channels, output_channels, kernel_size, stride),
                nn.BatchNorm2d(output_channels),
                nn.LeakyReLU(0.2, inplace=True),
            )
        else:
            return nn.Sequential(
                nn.Conv2d(input_channels, output_channels, kernel_size, stride),
            )

    def forward(self, image):

        crit_pred = self.crit(image)
        return crit_pred.view(len(crit_pred), -1)
```

But unfortunately we could not achieve much stability and balance between the generator and the critic training since W-GAN required a lot of more hyperparameter tuning which is a result of the necessity to have gradient penalty to enforce the 1-Lipshzitz continuity condition on the

W-Loss for it to train properly.

8.2 Bayesian CNN

Machine learning techniques have been growing at exponential rates over the recent years, and deep neural networks are among the most advanced state-of-the-art technologies that have widespread applications in the real world, including many sensitive applications like security, banking, healthcare and so on. Deep neural networks have been very successful in solving these problems successfully, but sometimes come up with solutions based on overconfident decisions. Efforts have been made to rectify this by using techniques such as regularisation and dropout layers. However they still tend to make overconfident decisions, especially in regions where there is very little or no data. They tend to extrapolate calculations made on the data available and apply it to data from regions where data is not available. For example when a CNN classifier designed to classify images as either cats or dogs is shown the image of a human, it will still predict one of either cat or dog. This type of overfitting problem cannot be resolved with normal regularization techniques such as L2, dropout etc.

To combat this problem, a method that has been proposed is the Bayesian neural network. This technique introduces uncertainty into the model by associating all weights and biases to random distributions instead of point values, and every time the model is run, a new value is sampled from the distribution, thereby adding uncertainty. The uncertainty changes with the mean and standard deviation of the random distributions used.

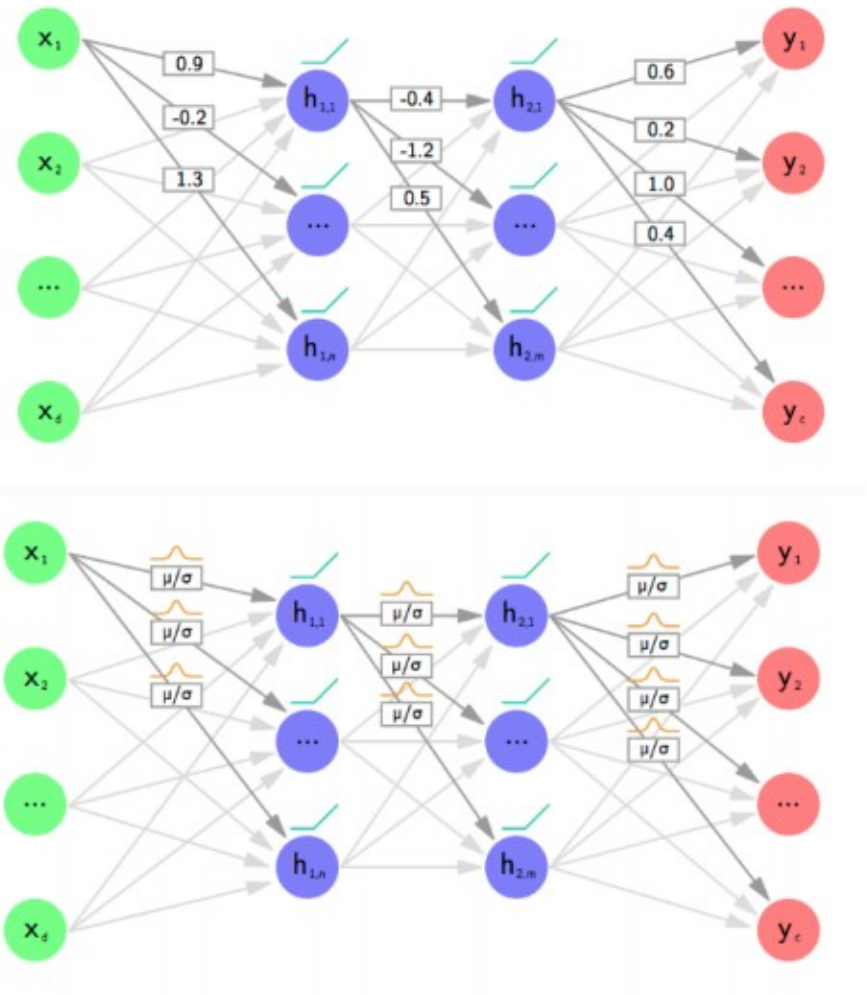


Figure 21: Top: Conventional determinstic neural network with weights having single fixed values. Bottom: Bayesian neural network with weights being distributions

The function that the neural network represents is assumed to be a set of functions, called the a priori estimates. The Bayes theorem is then applied to find the a posteriori distribution given the values in our dataset. The output corresponding to an input is found by integrating all possible functions:

$$p(y^*|x^*, X, Y) = \int p(y^*|f^*)p(f^*|x^*, X, Y)df^* \tag{1}$$

The integral is impossible to compute and hence is approximated by using a finite number of random variables w. The equation is then:

$$p(y^*|x^*, X, Y) = \int p(y^*|f^*)p(f^*|x^*, w)p(w|X, Y)df^*dw.$$

Even this is not possible to evaluate, so a variational distribution q(w) is considered for the random distribution of weights. This approximation has to be as close as possible to the true posterior. The closeness is measured by Kullback-Leibler divergence, or KL divergence for short.

$$q(y^*|x^*) = \int p(y^*|f^*)p(f^*|x^*, w)q(w)df^*dw. \tag{2}$$

The KL divergence is minimized by maximizing the log evidence lower bound

$$KL_{\text{VI}} := \int q(w)p(F|X, w)\log p(Y|F)dFdw - KL(q(w)||p(w)) \tag{3}$$

8.2.1 Implementation

TensorFlow provides a module called TensorFlow Probability which can be used to implement uncertainty-aware networks that include Bayesian neural network models. We tried a few implementations of Bayesian CNNs including the LeNet-5 and the VGG-16 architrctures, but were unfortunately unable to obtain even baseline satisfactory results.

9 References

1. Chris Donahue, Julian McAuley, Miller Puckette. Adversarial Audio Synthesis, ICLR 2019

2. Boyang Zhang, Jared Leitner, Sam Thornton. Audio Recognition using Mel Spectrograms and Convolution Neural Networks. http://noiselab.ucsd.edu/ECE228_2019/Reports/Report38.pdf

3. Aggelina Chatziagapi, Georgios Paraskevopoulos, Dimitris Sgouropoulos, GeorgiosPantazopoulos, Malvina Nikandrou, Theodoros Giannakopoulos, Athanasios Katsamanis,Alexandros Potamianos, Shrikanth Narayanan. Data Augmentation using GANs for Speech Emotion Recognition. Interspeech 2019

4. Edward Ma. Data Augmentation for Speech Recognition. https://towardsdatascience.com/data-augmentation-for-speech-recognition-e7c607482e78

5. Kumar Shridhar, Felix Laumann, Marcus Liwicki. Uncertainty Estimations by Softplus normalization in Bayesian Convolutional Neural Networks with Variational Inference. arXiv:1806.05978

6. Martin Arjovsky, Soumith Chintala, Léon Bottou. Wasserstein GAN. arXiv:1701.07875

7. Livingstone SR, Russo FA (2018) The Ryerson Audio-Visual Database of Emotional Speech and Song (RAVDESS): A dynamic, multimodal set of facial and vocal expressions in North American English. PLoS ONE 13(5): e0196391.

8. Houwei Cao, David G. Cooper, Michael K. Keutmann, Ruben C. Gur, Ani Nenkova, and Ragini Verma. CREMA-D: Crowd-sourced Emotional Multimodal Actors Dataset

9. Pichora-Fuller, M. Kathleen; Dupuis, Kate, 2020, "Toronto emotional speech set (TESS)", https://doi.org/10.5683/SP2/E8H2MF

10. Karen Simonyan, Andrew Zisserman; 2015, "Very Deep Convolutional Networks for large-scale image recognition", https://arxiv.org/pdf/1409.1556.pdf

11. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun; 2015, "Deep Residual Learning for Image Recognition", https://arxiv.org/pdf/1512.03385.pdf