# Classifying Emotions with Synthetic Data Augmentation

*By: Abhimanyu Warrier, Manav Gurnani, Daivik Bhardwaj*

## Introduction

Many digital systems still struggle to recognize or react to emotional cues, creating a gap between how people feel and how technology or institutions respond. In areas like online communication, customer service, education, and healthcare, systems often rely only on text or static data and ignore facial expressions, leading to poor user experiences, misinterpretations, and missed chances for timely support.

This project aims to improve how machines understand and respond to emotion using computer vision. We built and evaluated multiple models, from a simple single-layer network to deeper ResNet-style architectures, and boosted performance with both standard and synthetic data augmentation. In particular, we trained a Deep Convolutional GAN (DCGAN) to generate refined images that reinforced our baseline model and helped improve overall accuracy. The rest of this report walks through our dataset, model designs, experiments, and results to show how each step contributed to better emotion detection and image classification.

## Dataset Description

https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset

### A. Overview

Our chosen dataset was the Face Expression Recognition Dataset which contains a total of 35.9k photos among 7 different classes: angry, disgust, fear, happy, neutral, sad, surprise. We chose this dataset as it well represented the main categories of facial emotional expressions and gave us enough data to train our models.

### B. Specifications

Training Size: 28,821 images, Testing Size: 7,066 images. Total data size: 56.61 MB. Creator: Jonathan Oheix.

| Class | Number of Totals Samples | Total Proportion (%) |
|---|---|---|
| Angry | 4953 | 13.80 |
| Disgust | 547 | 1.52 |
| Fear | 5121 | 14.27 |
| Happy | 8989 | 25.04 |
| Neutral | 6198 | 17.27 |
| Sad | 6077 | 16.93 |
| Surprise | 4002 | 11.15 |

**Table 1: Class-wise breakdown of the dataset's samples.**

## Experimentation Methods

### A. Data Augmentation

Considering our dataset, we see that disgust class only takes up ~1.5% of the entire dataset, while happy images take up over 25% of our dataset. This causes a major imbalance in our dataset, which, if used to train a large model from scratch, prevents our model from adequately learning from a set of disgusted faces. As our samples for train and test are small for the disgusted images, the model might have great training accuracy, but it could fail to generalize images outside the dataset.

To counter this issue, data augmentation is used to improve the number of samples for each class, or even improve selective classes in our dataset. A more balanced dataset is more likely to train better across all classes as it prevents the model from over/underlearning patterns for a single class. There are two types of data augmentation we explored within this paper, linear and synthetic data augmentation.

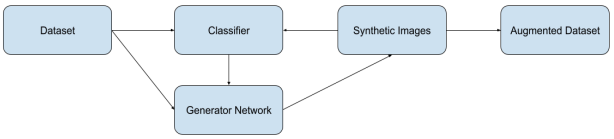



Figure 1: Synthetic Data Augmentation Pipeline

### A.1. Synthetic Data Augmentation Pipeline

Considering the imbalanced nature of our dataset, we decided that linear transformations (i.e. flips, translations, blurring) would only augment our currently existing dataset. We wanted to balance our dataset with equal proportions of images in each class, which motivated us to design a Synthetic Data Augmentation Pipeline. This system involved training a Generator Network and a Classifier on the current dataset, where we use the results to selectively augment our unbalanced classes by selecting high-confidence predictions from feeding our images into the classifier. This system would only allow high-confidence, rich-quality samples to be appended into our dataset.

Generative Adversarial Networks (GAN) - Generator Network

In order to take a unique approach to increase our data, we decided to try a GAN. GANs allow us to synthetically expand our dataset by producing new samples that would replicate the distribution of the original dataset. This approach helps us introduce subtle differences that would aid the model in generalizing, while also reducing the reliance on preprocessing and manually collecting the data. The hope with the model was that the GAN can enhance our dataset in a scalable manner.

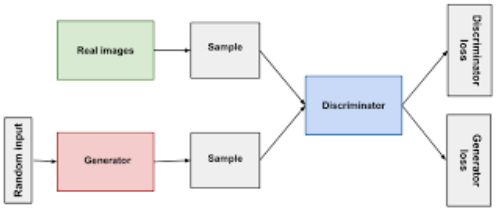



Figure 2: GAN Architecture (Credit: Google)

The image above represents the model architecture of our simple GAN Model. The code starts off by defining and constructing two neural networks, a Generator and Discriminator, that are trained against each other. The Generator starts with a small block of noise (a latent vector equaling 100 channels) and gradually expands it to the full extent of the image. At each respective stage, batch normalization smooths the feature extractions, ReLU helps build expressive final results, and Tanh squeezes all the pixels so the output is a clean image. The Discriminator does the exact opposite, trying to seek out the fake pictures, creating a system where the GAN should produce realistic synthetic samples that resemble the original dataset. Our results are found below. (See Image D)

DCGAN (Deep Convolutional Generative Adversarial Network)

Not satisfied with our blocky images from the GAN, we decided to do a bit more research into a more robust and sustainable model but with a very similar architecture. In our research, we found out that DCGANs may be more appropriate for the task at hand, especially because of their ability to preserve spatial locality, while also optimizing the gradient flow of the model.

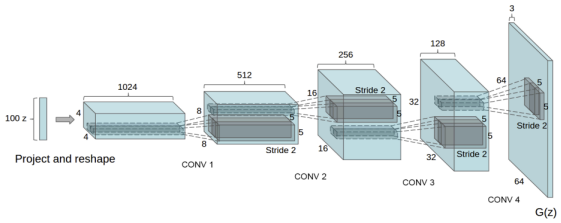



Figure 3: DCGAN Architecture (Credit: PyTorch)

The image above illustrates the model architecture we tried to replicate. After defining the key hyperparameters, tuned compared to the GAN, the Generator starts upsampling the 100-dimensional latent vector, this time with the same learning rate for both Neural Networks. This allows the model to gradually transform them through convolutional layers, to finally result in a much more realistic set of images. Our results for this are found below. (See Appendix E)

**B. Modeling**

Single Layer Network (Classifier)

The first model we trained was a simple, single-layer linear classifier. The purpose of using such a model was to almost act as a baseline or control group, to display the possible improvements as the model complexity increased.

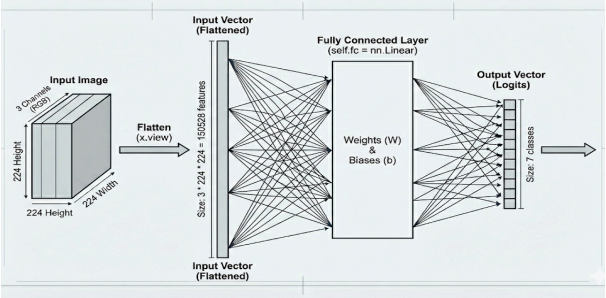



Figure 4: Single Layer Architecture (Credit: Nanobanana)

This model is a simple fully connected classifier that flattens each 3×224×224 RGB image into a 150,528-dimensional vector and passes it through a single linear layer. That layer maps directly to 7 output logits (one per emotion class), acting as a basic classifier that learns a direct mapping from pixel values to emotion labels.

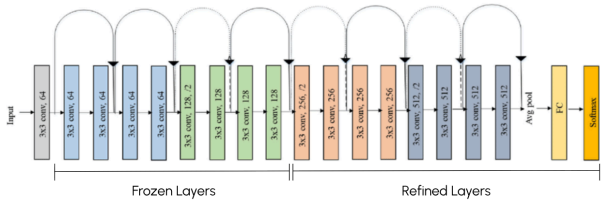



Figure 5: ResNet Architecture (Credit: Google)

ResNet18 (Classifier)

For our mainstream classifier, we used a ResNet18 architecture with pre-defined weights from the ImageNet dataset. With the imbalanced class samples, we realized that using a pre-trained model would give us better results as it would've already extracted key image features after being trained on a dataset as big as the ImageNet.

This led us to freeze the first two convolutional blocks in our architecture while refining the final two convolutional blocks with the output layers. This was done to retain the initial convolutions while pushing our model to fit the latter layers to extract features from our facial images. The benefit of using such an architecture over a Single (or Multi-layer) Perception network is the process of convolution, which allows us to harness key features from images, which would help the model better adapt to trends and patterns found in the dataset. Keeping the first two layers frozen would allow for the model to harness general object trends from our images, while refining the last two layers would allow the model to fit patterns to our facial images specifically and the emotions on those faces.

## Results & Analysis

### 1. Classifier Models

Single Layer Network (Appendix A)
The model did not perform well with training and testing accuracies of around 18% peak for training and 22% for testing. The model averaged 1/5 images correctly, sometimes hitting ⅖. As seen in the confusion matrix, the model was very bad at properly classifying images in the "disgust" category due to dataset imbalance. As expected, using a single layer architecture isn't good for classifying images, as it lacks convolutional filters as well as depth which would normally be necessary in extracting features from the image.

ResNet 18 (Pre-trained) [Appendix B]
This model was trained over 60 epochs with adaptive learning rates for different layers (0.0001 for the fine-tuned layers, 0.001 for the fully connected layer). For additional training, we added Dropout layers to ensure that all segments of the fully-connected layer were adequately trained. The results showed us a steady converging training accuracy of 90% and a high testing accuracy peaking at 67%. We see that the model is failing to predict images of class 1 (disgust), which was our most underrepresented class in the dataset.

ResNet 18 (with Synthetic Augmentation) [Appendix C]
We use the same architecture and hyperparameters as the pre-trained model as we were looking to measure the difference with our Synthetic Data Augmentation Pipeline. The results showed us a steady converging training accuracy to 99% and a lower testing accuracy peaking at 17%. However, out of the predictions from this model, we see a greater benefit class-wise predictions (disgust class has better predictions off the confusion matrix in Appendix C). This model could be further refined with hyperparameter tuning and the pipeline can be improved with a better trained DCGAN.

### 2. Generative Models

GAN (Appendix D)
The images generated by the basic GAN appeared blocky and unrealistic, which limited their usefulness for training. This was largely due to the instability of the GAN, where the generator and discriminator did not really converge properly, resulting in poor quality images. The GAN model ran for 100 epochs.

DCGAN (Appendix E)
The poor quality of images from the GAN highlighted the need for more advanced, hypertuned architectures, drastically improving the image quality. After using techniques that preserved spatial structure and created smoother, more coherent image upscaling, much better images were produced. The DCGAN model ran for 100 epochs with an average discriminator loss of 0.3026 and average generator loss of 5.2342.

## Contributions

*Warrier* - Report Setup, Single Layer architecture & testing, ResNet testing, Standard Architecture code (training, testing, random image prediction sim.), Presentation setup, Data augmentation research

*Gurnani* - Developed and fine-tuned pre-trained ResNet architectures, built the classification translation component in system, helped with GAN and DCGAN tuning.

*Bhardwaj* - Implemented core GAN and DCGAN architectures, enabling the generation of realistic synthetic data and experimentation with deep adversarial training techniques. Helped with ResNet modeling and architecture.

## References

J. Oheix, “Face expression recognition dataset,” Kaggle. [Online]. Available: https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset. Accessed: Dec. 7, 2025

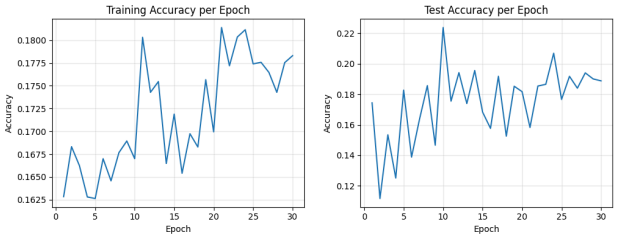
N. Inkawhich, DCGAN tutorial — PYTORCH tutorials 2.9.0+Cu128 documentation, https://docs.pytorch.org/tutorials/beginner/dcgan_faces_tutorial.html (accessed Dec. 8, 2025).

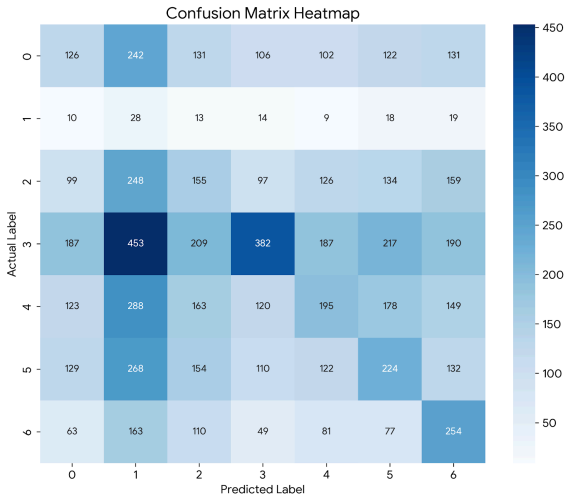
S. Chilamkurthy, “Domains,” Transfer Learning for Computer Vision Tutorial - PyTorch Tutorials 2.9.0+cu128 documentation, https://docs.pytorch.org/tutorials/beginner/transfer_learning_tutorial.html (accessed Dec. 7, 2025).

“ResNet,” PyTorch, https://pytorch.org/hub/pytorch_vision_resnet/ (accessed Dec. 7, 2025).

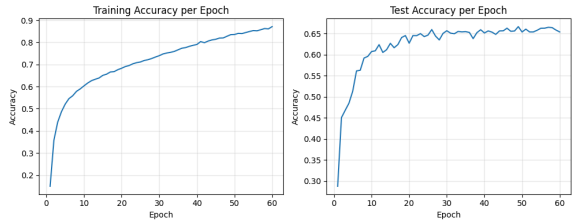
## Appendix

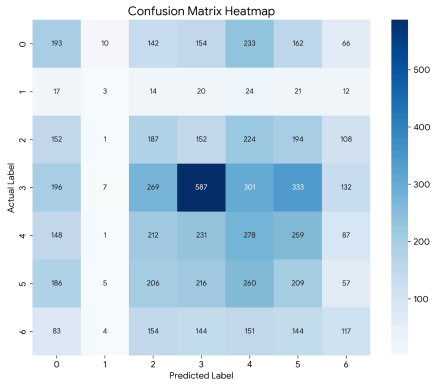
*Appendix A: Single Layer Performance Graphs*



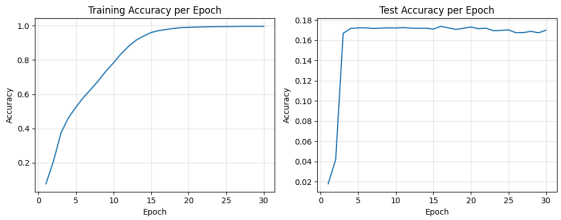



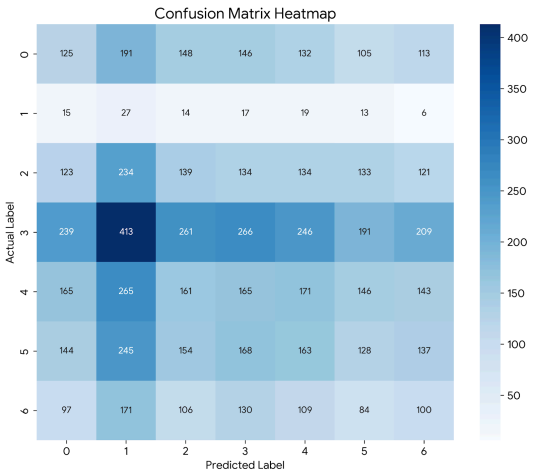

*Appendix B: ResNet18 (Pre-trained)*




*Appendix C: ResNet18 with Synthetic Augmentation*




## *Appendix D: GAN Results*


## *Appendix E: DCGAN Results*