EMOTION RECOGNITION FROM FACIAL EXPRESSIONS

Venkata Sai Mahendra Somineni
50465102
vsominen@buffalo.edu

Naveen Bhogavalli
50471010
nbhogavalli@buffalo.edu

Dhanush Kumar Reddy Yarragonda
50468303
dhanushk@buffalo.edu

Abstract

This project aims to detect human emotions from images of facial expressions. A deep learning model involving Convolutional Neural Networks (CNN) are used to perform multi-class classification task among seven different types of emotions. The images are sourced from the FER2013 dataset where each image belongs to one of the seven emotion classes. The Problem statement is to classify facial data into different expression classes. This can be achieved using various machine learning models including Neural network, Convolutional neural network and many more. Here we intend to solve this problem using CNN architecture. The model learns to discriminate among images of different classes through training and when tested on a set of unseen images.

1 Dataset

1.1 Description of the Dataset

The FER2013 (Facial Expression Recognition 2013) dataset is a collection of images and categories that describe the emotion of the person in the image. The dataset contains 35,887 grayscale images of faces with seven different emotions. The emotions are Angry, Disgust, Fear, Happy, Sad, Surprise, and Neutral. The image resolution is 48x48 pixels. The faces have been automatically registered so that the face is more or less centered and occupies about the same amount of space in each image.

This dataset was prepared by Pierre-Luc Carrier and Aaron Courville, as part of an ongoing research project. The dataset was created by searching Google images for each emotion and synonyms of the emotions. The training set contains 28,709 examples and the public test set contains 3,589 examples. This dataset includes training and test datasets where data of each expression class is present in the respective class folder. We transform the input images to have a mean of 0.485, 0.456, and 0.406 for each layer of an RGB image, with standard deviations of 0.229, 0.224, and 0.225.

1.2 Data Engineering

To increase the number and diversity of training data while lowering overfitting, data augmentation techniques including image rotation, scaling, and flipping are used.

We included:

- **Random horizontal flip:** This Flips the image horizontally with an 80% probability. This helps the model generalize better by learning from variations in the dataset.

- **Color jitter:** Adjusts brightness, contrast, saturation, and hue randomly. This introduces diversity in the color distribution of the images, making the model more robust to different lighting conditions.
- **RandomRotation**: Rotates the image by a random angle (up to 30 degrees in this case). This augmentation helps the model learn features from rotated versions of the images.
- **RandomAffine:** Applies a random affine transformation to the image. The degrees parameter controls the range of rotation, and translate specifies the maximum absolute fraction for horizontal and vertical translations. This introduces additional geometric transformations.

These data augmentation when applied to the training dataset gave very good generalized results without overfitting.

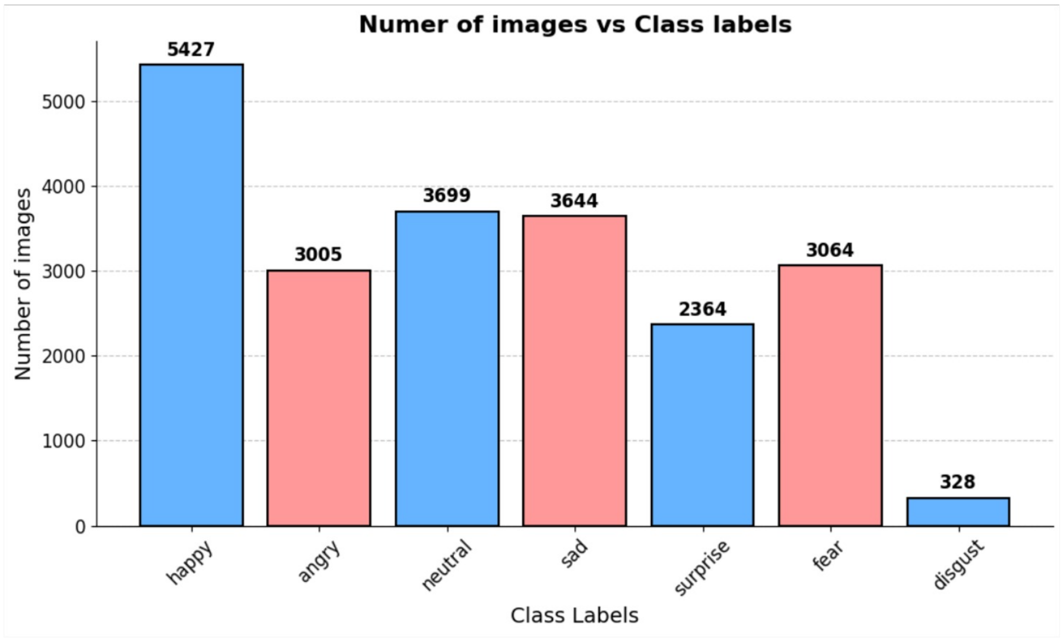

Figure 1: Count of Classes in the Dataset

This graph shows the plotting of Number of images vs Class lables . Upon analyzing the distribution of images across classes, a notable gap between "happy" and "disgust" caught our attention. To address potential accuracy challenges caused by imbalances, we implemented data augmentation techniques, such as random horizontal flip, color jitter, random rotation, and random affine transformations.

2 Model Description

In the development of our project, we have undertaken a thorough examination of six distinct models. Our objective was to assess and compare these models to identify the most effective solution for our project's needs. The models we have evaluated include:

2.1 BASIC CNN MODEL:

This is a basic Sequential CNN model that contains batch normalization and dropout layers on each CNN unit and a fully connected layer for classification.

CNNs are specifically designed to automatically and adaptively learn spatial hierarchies of features from images. They can capture the spatial and temporal dependencies in an image through the application of relevant filters, making them suitable for image recognition and classification tasks where the relationship between pixels is crucial.

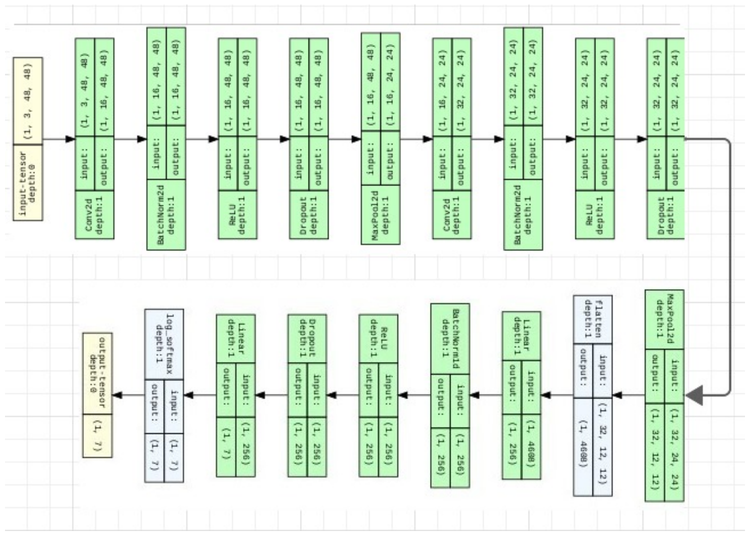
input-tensor depth:0 (1, 3, 48, 48)
Conv2d depth:1 input: (1, 3, 48, 48) output: (1, 16, 48, 48)
BatchNorm2d depth:1 input: (1, 16, 48, 48) output: (1, 16, 48, 48)
ReLU depth:1 input: (1, 16, 48, 48) output: (1, 16, 48, 48)
Dropout depth:1 input: (1, 16, 48, 48) output: (1, 16, 48, 48)
MaxPool2d depth:1 input: (1, 16, 48, 48) output: (1, 16, 24, 24)
Conv2d depth:1 input: (1, 16, 24, 24) output: (1, 32, 24, 24)
BatchNorm2d depth:1 input: (1, 32, 24, 24) output: (1, 32, 24, 24)
ReLU depth:1 input: (1, 32, 24, 24) output: (1, 32, 24, 24)
Dropout depth:1 input: (1, 32, 24, 24) output: (1, 32, 24, 24)
MaxPool2d depth:1 input: (1, 32, 24, 24) output: (1, 32, 12, 12)
flatten depth:1 input: (1, 32, 12, 12) output: (1, 4608)
Linear depth:1 input: (1, 4608) output: (1, 256)
BatchNorm1d depth:1 input: (1, 256) output: (1, 256)
ReLU depth:1 input: (1, 256) output: (1, 256)
Dropout depth:1 input: (1, 256) output: (1, 256)
Linear depth:1 input: (1, 256) output: (1, 7)
log_softmax depth:1 input: (1, 7) output: (1, 7)
output-tensor depth:0 (1, 7)

Figure 2: Basic CNN Model

2.2 Simple Resnet Model

ResNet, short for Residual Network, introduced by Kaiming He et al. in their seminal paper "Deep Residual Learning for Image Recognition," addresses the degradation problem encountered in training deep neural networks. Traditional deep networks suffer from performance degradation as they deepen, characterized by accuracy saturation, and then decline. ResNet's innovation lies in the use of residual learning blocks that incorporate skip connections or shortcuts, enabling the network to learn residual mappings instead of attempting to fit the desired underlying mapping. These skip connections allow the network to bypass certain layers, mitigating vanishing gradient issues and facilitating the training of extremely deep networks—up to hundreds or even thousands of layers. By introducing residual blocks and skip connections, ResNet effectively eased the optimization of deep networks, leading to significant improvements in accuracy and convergence rates across various computer vision tasks. Our implementation is similar but with only a few layers making it a similar but simple version of the original resnet model.

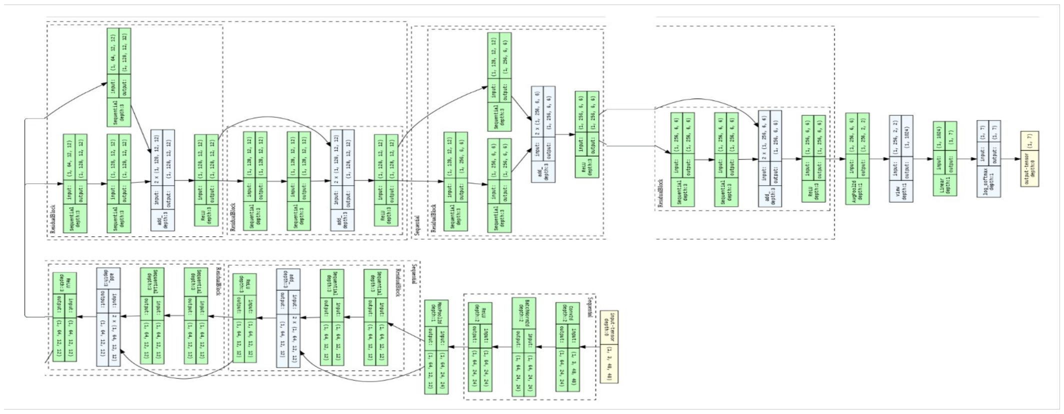

Figure 3: Simple Resnet Model

2.3 Variation of Google net model

GoogleNet, also known as Inception-v1, is a deep convolutional neural network architecture introduced by Szegedy et al. in the paper "Going Deeper with Convolutions." It comprises a unique module called the Inception module, which is designed to capture multi-scale features efficiently. The Inception module utilizes different filter sizes (1x1, 3x3, 5x5) and pooling operations in parallel and concatenates their outputs, allowing the network to capture both local and global features effectively. Additionally, GoogleNet incorporates global average pooling, which helps to reduce the number of parameters and prevents overfitting. The network architecture also includes auxiliary classifiers at intermediate layers during training to mitigate the vanishing gradient problem and encourage features to be informative across the network. Our custom model have a structure inspired by GoogleNet. It starts with convolutional layers followed by two Inception modules (inception1 and inception2) that capture multi-scale features through different filter sizes. Dropout layers are added to prevent overfitting, and adaptive average pooling is used to reduce the spatial dimensions before passing the data through fully connected layers (fc) for classification. Finally, a log softmax function is applied for output.

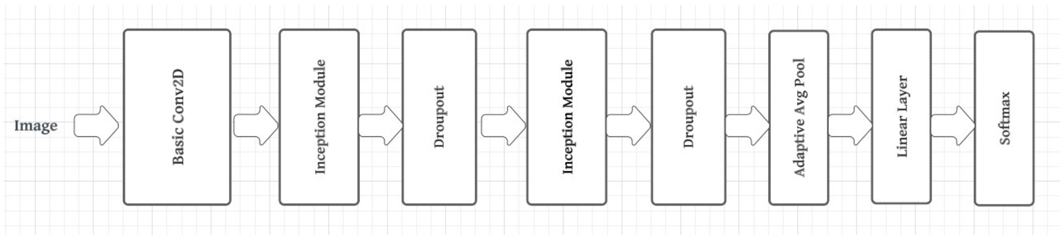

Figure 4: Google Net Model Architecture

2.4 Unet model (Encoder and Decoder)

The U-Net architecture is a popular convolutional neural network (CNN) commonly used in image segmentation tasks. It consists of an encoder-decoder structure with skip connections between corresponding encoder and decoder layers to capture both local and global information effectively. In a typical U-Net model, the last layer in the decoder part usually involves a convolutional layer that reduces the number of channels to match the desired output (e.g., segmentation masks). However, here we further add an Average pooling layer followed by a fully connected layer with a softmax activation function to make this custom classifier.

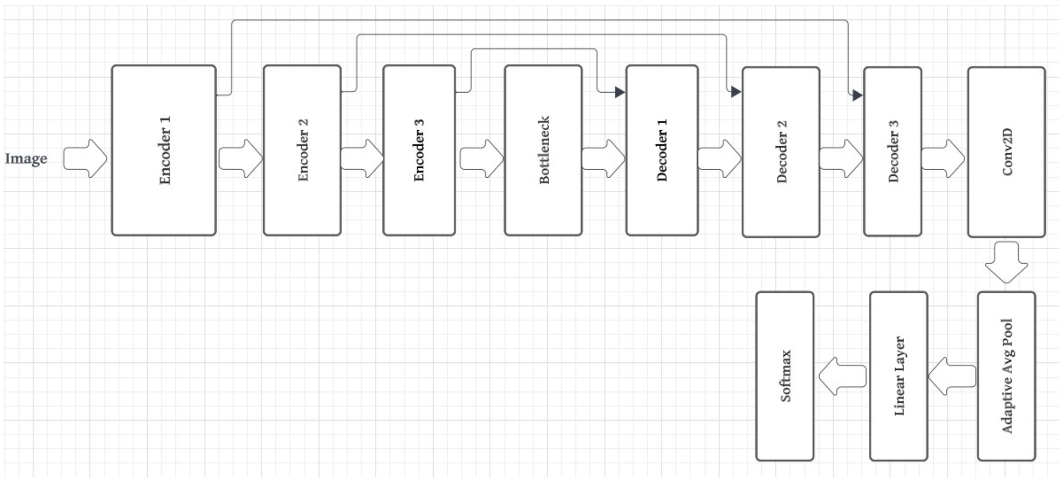

Figure 5: Unet Model Architecture

2.5 NewNet Model

Our model architecture is a fusion of two powerful architectures: the U-Net and GoogleNet (Inception model). We start with an encoder-decoder structure reminiscent of U-Net, designed to capture detailed features and spatial information. The end component handles the encoding and decoding process, transforming the input data (x) into an encoded feature representation. After this encoding stage, we integrate elements inspired by GoogleNet's Inception blocks. This involves passing the encoded features through convolutional layers (conv1) followed by an Inception block (inception1). This Inception block likely employs various filter sizes and pooling operations in parallel to extract multi-scale features effectively. Subsequent operations such as dropout regularization (dropout1 and dropout2), average pooling (avgpool), additional convolutions (conv3), fully connected layers (FC), and a log softmax function are applied to process the extracted features for classification purposes.

By merging the U-Net's encoder-decoder architecture with GoogleNet's Inception block, our model aims to benefit from the strengths of both. The U-Net structure specializes in detailed segmentation and spatial localization, while the Inception block excels in extracting multi-scale features. However, thorough experimentation and validation on our specific dataset or task will be crucial to assess the performance and effectiveness of this combined architecture.

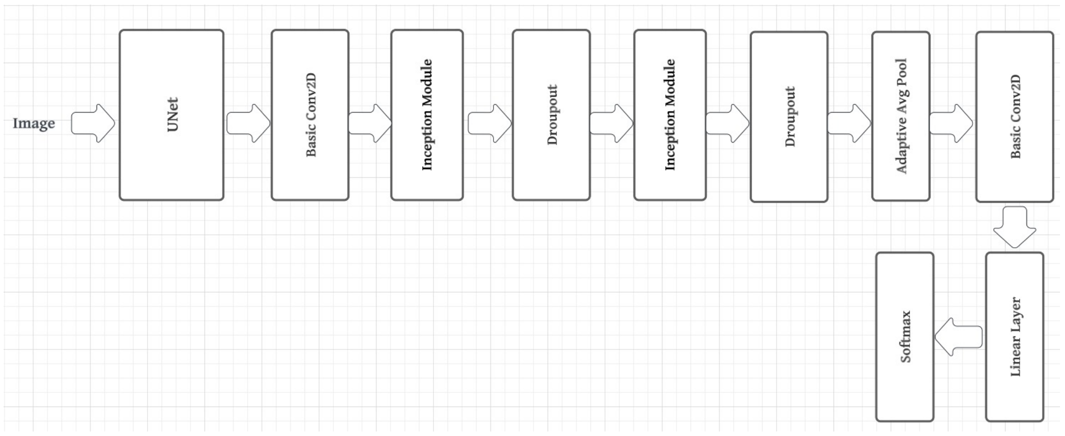

Figure 6: Newnet Model Architecture

2.6 Rogle net model (Custom model using resnet and google net)

Our model architecture seamlessly integrates aspects from both GoogleNet and ResNet models, combining their distinctive features for enhanced performance. The architecture begins with three pathways: pathway1, pathway2, and pathway3, each processing the input data (x) through different convolutions: pathway1 through a single convolution, pathway2 through a 1x1 convolution followed by a 3x3 convolution, and pathway3 via a 1x1 convolution followed by a 5x5 convolution. The outputs from these pathways are concatenated to capture diverse and multi-scale features effectively.

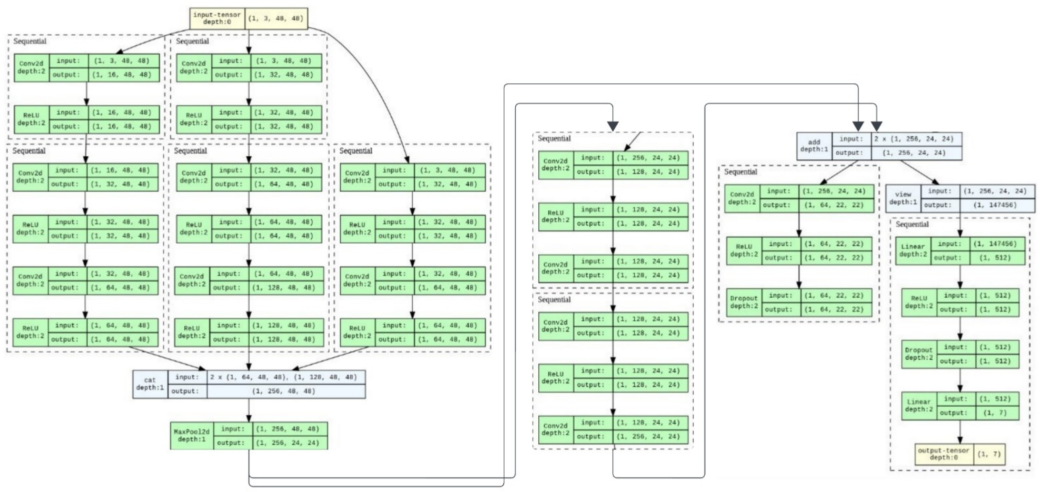
input-tensor depth:0 (1, 3, 48, 48)
Sequential
Conv2d depth:2 input: (1, 3, 48, 48) output: (1, 16, 48, 48)
ReLU depth:2 input: (1, 16, 48, 48) output: (1, 16, 48, 48)
Sequential
Conv2d depth:2 input: (1, 3, 48, 48) output: (1, 32, 48, 48)
ReLU depth:2 input: (1, 32, 48, 48) output: (1, 32, 48, 48)
Sequential
Conv2d depth:2 input: (1, 16, 48, 48) output: (1, 32, 48, 48)
ReLU depth:2 input: (1, 32, 48, 48) output: (1, 32, 48, 48)
Conv2d depth:2 input: (1, 32, 48, 48) output: (1, 64, 48, 48)
ReLU depth:2 input: (1, 64, 48, 48) output: (1, 64, 48, 48)
Sequential
Conv2d depth:2 input: (1, 32, 48, 48) output: (1, 64, 48, 48)
ReLU depth:2 input: (1, 64, 48, 48) output: (1, 64, 48, 48)
Conv2d depth:2 input: (1, 64, 48, 48) output: (1, 128, 48, 48)
ReLU depth:2 input: (1, 128, 48, 48) output: (1, 128, 48, 48)
Sequential
Conv2d depth:2 input: (1, 3, 48, 48) output: (1, 32, 48, 48)
ReLU depth:2 input: (1, 32, 48, 48) output: (1, 32, 48, 48)
Conv2d depth:2 input: (1, 32, 48, 48) output: (1, 64, 48, 48)
ReLU depth:2 input: (1, 64, 48, 48) output: (1, 64, 48, 48)
cat depth:1 input: 2 x (1, 64, 48, 48), (1, 128, 48, 48) output: (1, 256, 48, 48)
MaxPool2d depth:1 input: (1, 256, 48, 48) output: (1, 256, 24, 24)
Sequential
Conv2d depth:2 input: (1, 256, 24, 24) output: (1, 128, 24, 24)
ReLU depth:2 input: (1, 128, 24, 24) output: (1, 128, 24, 24)
Conv2d depth:2 input: (1, 128, 24, 24) output: (1, 128, 24, 24)
Sequential
Conv2d depth:2 input: (1, 128, 24, 24) output: (1, 128, 24, 24)
ReLU depth:2 input: (1, 128, 24, 24) output: (1, 128, 24, 24)
Conv2d depth:2 input: (1, 128, 24, 24) output: (1, 256, 24, 24)
add depth:1 input: 2 x (1, 256, 24, 24) output: (1, 256, 24, 24)
Sequential
Conv2d depth:2 input: (1, 256, 24, 24) output: (1, 64, 22, 22)
ReLU depth:2 input: (1, 64, 22, 22) output: (1, 64, 22, 22)
Dropout depth:2 input: (1, 64, 22, 22) output: (1, 64, 22, 22)
view depth:1 input: (1, 256, 24, 24) output: (1, 147456)
Sequential
Linear depth:2 input: (1, 147456) output: (1, 512)
ReLU depth:2 input: (1, 512) output: (1, 512)
Dropout depth:2 input: (1, 512) output: (1, 512)
Linear depth:2 input: (1, 512) output: (1, 7)
output-tensor depth:0 (1, 7)

Figure 7: Rogle Net Model Architecture

Following the concatenation, a max pooling operation is applied to merge the pathway outputs. Subsequently, residual connections, a hallmark of ResNet models, are incorporated. These connections facilitate the bypassing of certain layers, promoting smoother gradient flow and aiding in training deeper networks. The residual out is obtained by passing the max-pooled data through residual convolutional layers and adding it back to the original max-pooled output.

The model concludes with a series of operations including a final convolutional layer (final cnn) applied to the residual connection's output. This is followed by flattening the data and passing it through fully connected layers (fc) for classification or prediction.

By amalgamating elements from both GoogleNet and ResNet, our model aims to leverage the multi-scale feature extraction capabilities of GoogleNet while harnessing the advantages of ResNet's residual connections for improved training and convergence of deeper networks. However, rigorous evaluation and fine-tuning on specific datasets or tasks are necessary to validate the effectiveness and performance of this hybrid architecture.

3 Loss Functions

Here we used a simple Cross Entropy loss, which is the best loss function of a multiclass classification task. Other loss functions like Log loss and BCE were not suitable for this classification task and hence we stuck to Cross Entropy loss for this task.

3.1 Cross Entropy loss:

The goal of the cross-entropy loss is to minimize the difference between the predicted probability distribution and the actual distribution of the target labels. Additionally, weighted cross-entropy loss introduces the concept of assigning different weights to different classes based on their importance or frequency in the dataset. This can be particularly useful in scenarios where certain classes are more significant or imbalanced compared to others.

In our case we can see a significant imbalance in the classes present, so we have more weight on classes with fewer samples compared to to classes with more samples. This significantly increased the recall and precision of class with fewer samples, in our case disgust.

3.2 Other Loss Functions:

Log Loss is often used in binary classification problems. It's a measure of uncertainty, where the goal is to minimize the divergence between the true labels and the predicted probabilities. In a multiclass setting, it's typically less effective than Cross Entropy loss. and Binary Cross Entropy (BCE) is another loss function designed for binary classification tasks which is not the case in multiclass emotion recognition.

3.3 Innovation on the loss function:

Focal Loss which is designed for tackling class imbalance in object detection tasks, Focal Loss modulates the standard Cross Entropy loss to focus more on hard-to-classify examples. Label Smoothing: This technique can be applied to the Cross Entropy loss. It slightly alters the hard labels to make the model less confident and more regularized, potentially improving its generalization ability.

4 Optimization algorithm:

Here we experimented with SGD and Adam optimization algorithms. Adam optimizer is the best optimizer when it comes to convergence to minimize loss in fewer steps. This is because it maintains a moving average of gradient and square gradients and this is controlled by decay variables. But SGD optimizer is used to get a more optimal solution in the long run, so we use SGD optimizer for more complex models and use higher epochs to get good results. We used SGD optimizer for our best model and it gave good results.

4.1 Innovation on optimization algorithm

- **Learning Rate Scheduling:** Adjusting the learning rate during training can significantly impact the performance of SGD.
- **Momentum Tuning in SGD:** Adjusting the momentum parameter in SGD can help accelerate SGD in the relevant direction and dampen oscillations.
- **Hybrid Approaches:** Combining the strengths of both SGD and Adam in different phases of training (starting with Adam for fast convergence and switching to SGD for fine-tuning) can sometimes yield better results.

5 Metrics and Experimental Results

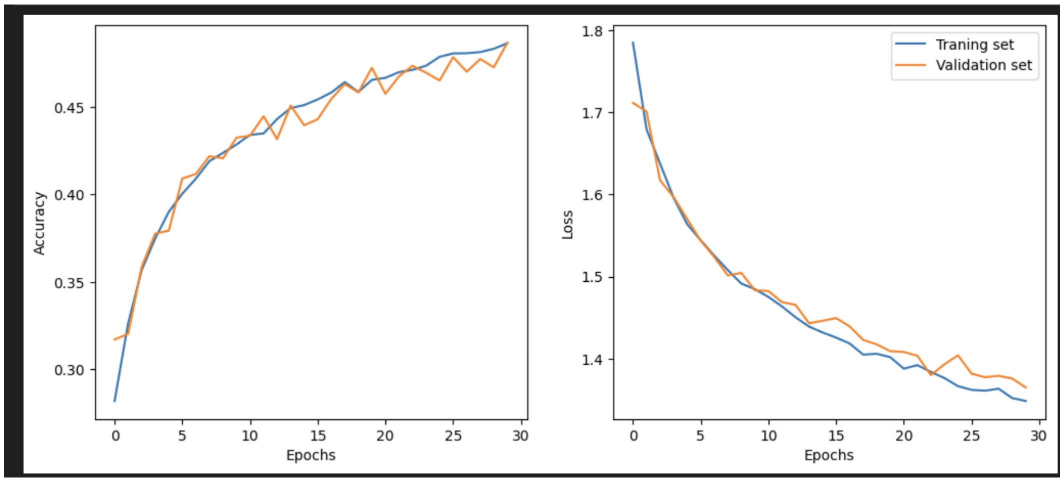

Figure 8: loss vs Accuracy of Basic CNN Model

From fig.8, The graph shows that the accuracy on both the training set and the validation set generally increases over epochs, which suggests that the model is learning and improving its predictions as it processes more data. However, the accuracy seems to plateau toward the later epochs, indicating that the model may be approaching its limit in learning from the data provided. Over time, as the model goes through more epochs, the decrease in loss tapers off, suggesting diminishing returns from additional learning. This plateau in the learning curve suggests the model may be converging on its best solution given the current architecture and dataset.

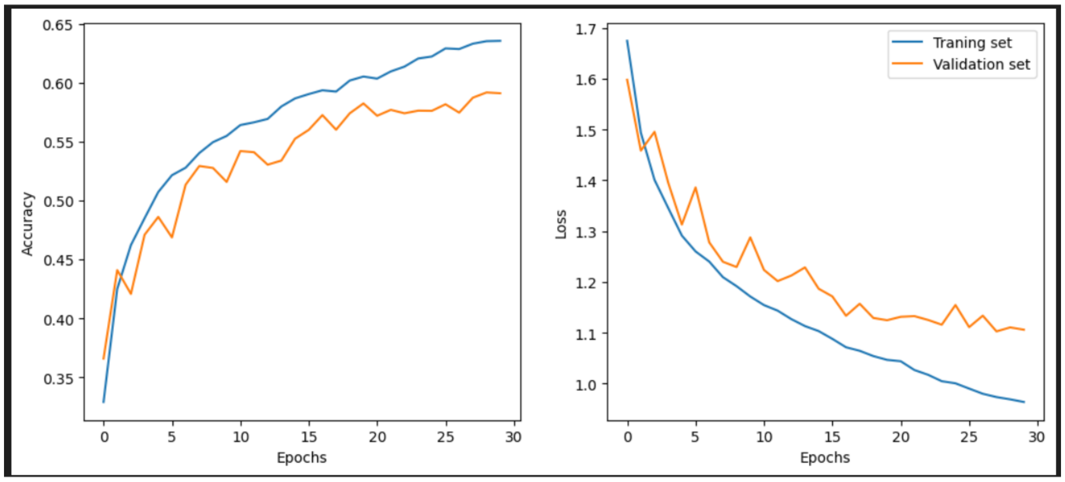

Figure 9: loss vs Accuracy of Simple RESNET Model

From Fig.9, we say that this model demonstrates improved performance over the CNN model, with both higher initial and final accuracy, and lower initial and final loss, indicating a more suitable model. While the validation loss does not show an increase as seen in the first graph, which suggests no overfitting, there are more fluctuations, pointing to potential benefits from regularization or tuning. Lastly, a learning plateau is reached at a higher accuracy, hinting at a superior learning capacity or differences in data or model structure.

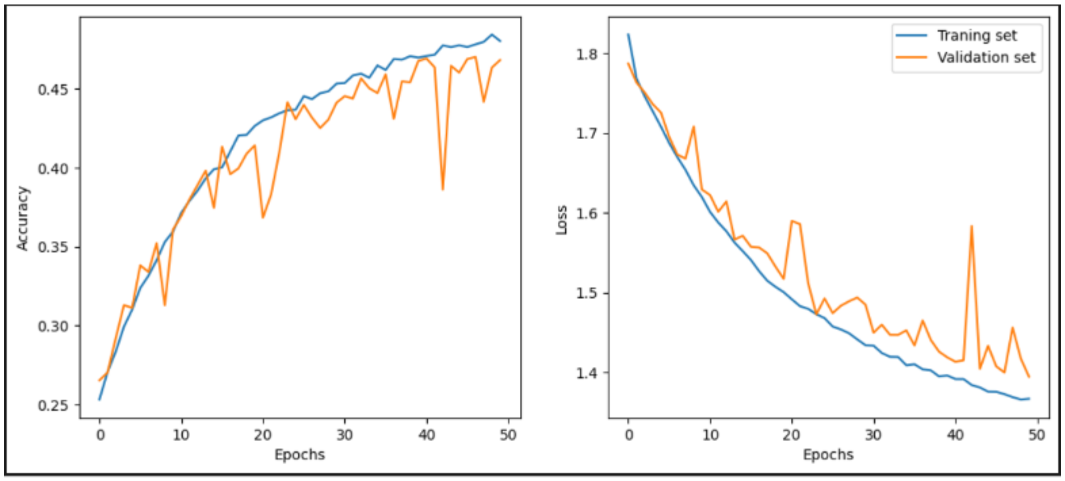

Figure 10: loss vs Accuracy of Google Net Model

The accuracy in the Google Net (Fig.10) does not improve as steadily as in the previous model(resnet), and the loss is not decreasing as smoothly, which suggests that the Google Net model is likely performing worse in terms of generalization on the validation data. The fluctuations and spikes in validation loss are more pronounced in this graph, which is typically a sign of a model that is not well-tuned or a dataset that is not well-prepped

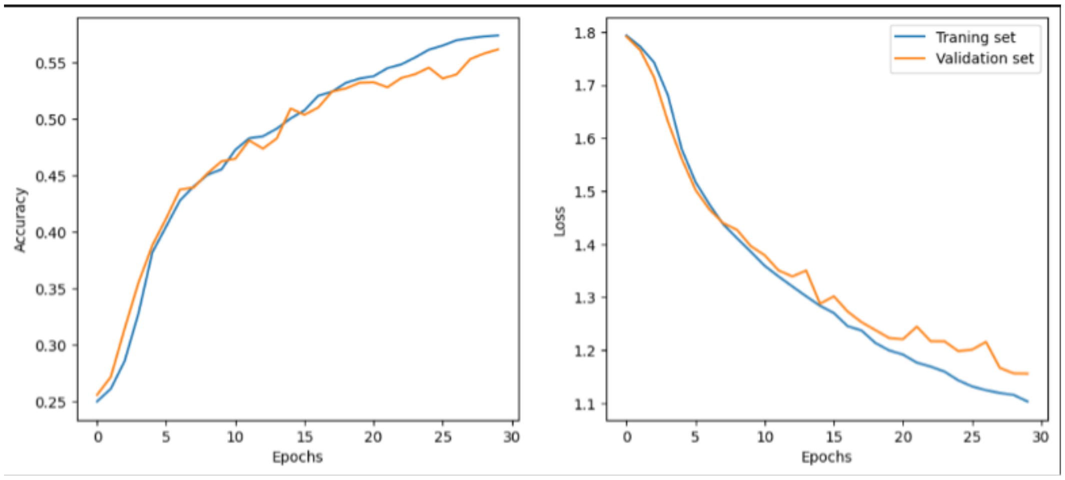

Figure 11: loss vs Accuracy of UNET Model

The accuracy and loss in this graph (fig.11) show less fluctuation, indicating a more stable training process than the one observed in the graph with 50 epochs. Unlike the previous graph(fig.10), where the validation metrics showed high volatility, this graph displays a smoother curve for both accuracy and loss, suggesting better model performance and more reliable learning. The convergence of training and validation lines in this graph is much tighter, which typically points to a model that is generalizing well, as opposed to the larger gaps seen in the Google Net Model.

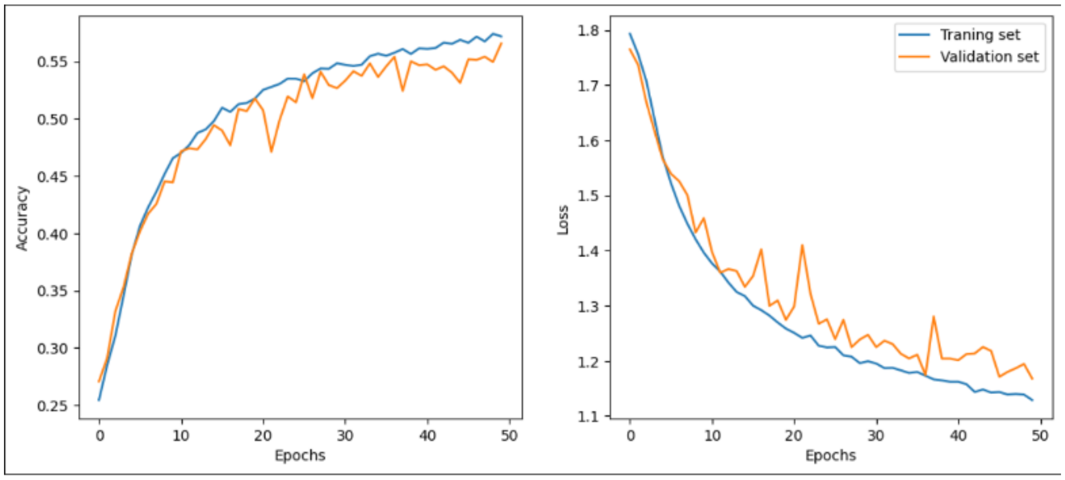

Figure 12: loss vs Accuracy of NewNet Model

The accuracy steadily improves over time, with the validation accuracy initially lagging but then closely following the training accuracy, indicating that the model is learning and generalizing well (fig.12). The loss for both training and validation sets trends downward, though the validation loss exhibits some volatility with spikes, suggesting potential overfitting or issues with the validation data.

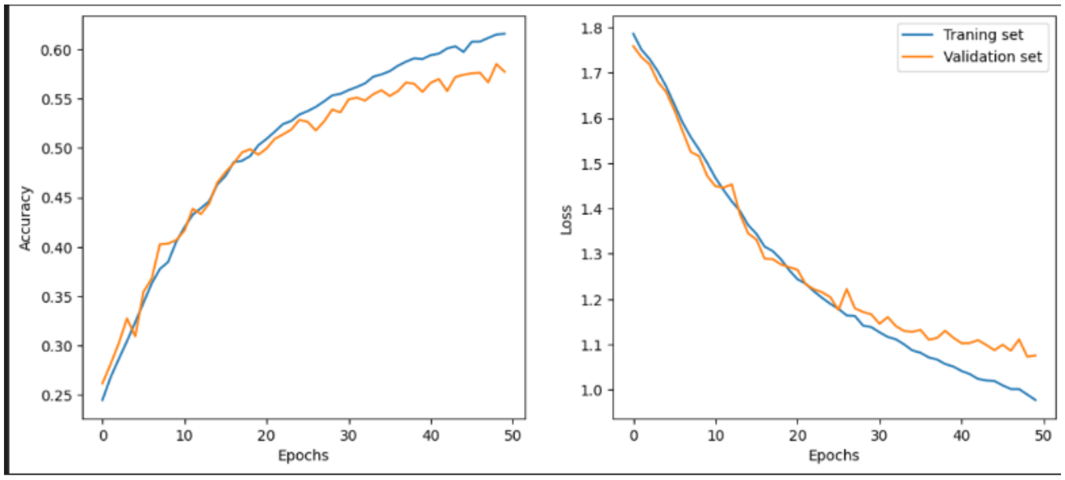

Figure 13: loss vs Accuracy of Roglenet model

The model's accuracy on both the training and validation sets increases steadily as the number of epochs grows, which is indicative of the model learning and improving its predictions over time (fig.13). The training set shows a slightly higher accuracy, which is typical as the model is directly learning from this data. Both the training and validation loss decrease over the epochs, reflecting the model's improving predictions. There is a clear downward trend in loss, with the training loss being consistently lower than the validation loss, which is expected.

Here we have used various metrics such as accuracy, precision, recall, F1 score, and confusion matrix. We also have accuracy and loss values of the train and validation set concerning the epoch while training the model.

We can observe that happy and surprise expressions are easily classified compared to other expressions since it's very complex to distinguish other expressions. Angry and neutral expressions were also well classified. From the data, we can observe that Sad, neutral, and disgust were the most confused expressions for our model. Future work may employ unsupervised pre-training transfer learning methods to further cut down on recognition failures. Pre-processing and feature extraction techniques can be applied before training models. Transfer Learning can also be tested using these datasets

Model	Accuracy
Basic CNN Model	53.9%
Simple ResNet Model	60.9%
Variation of GoogleNet Model	48.4%
UNet Model (Encoder and Decoder)	60.7%
NewNet Model	60.6%
Custom Model (RogleNet) using ResNet and GoogleNet	63.6%

Table 1: Accuracy of different models

6 Contributions and Github

Below are the percentage of contributions made by each team member.

Team Member	Contribution Percentage	Responsibilities
vsominen	33.33%	Basic CNN Model ,Custom model, Documentation
nbhogavalli	33.33%	Simple Resnet Model,Custom model, Documentation
dhanushk	33.33%	Variation of Google net model, Unet model, NewNet Model

Table 2: Contributions of each team member to the project.

6.1 Github Repository Link

https://github.com/mahendrasomineni/FacialEmotionRecognition

References

1. Y. Ho and S. Wookey, "The Real-World-Weight Cross-Entropy Loss Function: Modeling the Costs of Mislabeling," in IEEE Access, vol. 8, pp. 4806-4813, 2020, doi: 10.1109/ACCESS.2019.2962617.

2. Kaiming He,Xiangyu Zhang,Shaoqing Ren,Jian Sun," Deep Residual Learning for Image Recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

3. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich, https://arxiv.org/abs/1409.4842 //

4. Olaf Ronneberger, Philipp Fischer, Thomas Brox, U-Net: Convolutional Networks for Biomedical Image Segmentation, https://arxiv.org/abs/1505.04597//