RUTGERS
UNIVERSITY | NEW BRUNSWICK

# Human Emotion Detection


By

**Rohit Macherla**


A Research Project Submitted in
Partial Fulfillment of the
Requirements for the
Degree of Master of Science in Statistics
Specialization in Data Science

At

**Rutgers, the State University of New Jersey**

May 2024

# Table of Contents

# Abstract

In the realm of computer vision, the accurate detection of human emotions from image data holds significant importance for various applications ranging from healthcare to human-computer interaction. This project focuses on detecting emotions using convolutional neural networks (CNNs) and transformer models in vision. The research uses a dataset with three emotion classes- "angry", ”happy," and "sad” sourced from Kaggle. The approach involves preprocessing steps like data augmentation and standardizing image sizes. The project explores four models- Baseline CNN, ResNet50, EfficientNetB4 and Vision Transformer. Each model was trained for 30 epochs with the Adam optimizer to minimize categorical cross entropy loss. Key settings include a batch size of 32, a learning rate of 0.001 (5e-5 for Vision Transformer) and early stopping techniques to avoid overfitting.

Results show that the transformer based model performs well after just 6 epochs outperforming the CNN models. Among the CNN architectures EfficientNetB4 stands out for its performance while maintaining model complexity compared to ResNet50. This study contributes to improving emotion detection systems by demonstrating the effectiveness of transformer models that were primarily used for text data can also be prominent in image tasks leading to more precise and efficient emotion recognition, in real world scenarios.

# Introduction

Detecting human emotions from images is a challenging yet crucial task in numerous applications. Traditional approaches to dealing with image data relied heavily on handcrafted features and shallow machine learning models. However, recent advancements in deep learning, particularly in convolutional neural networks (CNNs) and transformer models, have revolutionized this field by enabling more accurate and efficient image recognition.

This research project (reference-1) focuses on exploring the effectiveness of CNNs and transformer models for emotion detection in images and compares the performance of different deep learning architectures, including simple Baseline CNN, complex CNNs architectures that incorporate residual connections like ResNet and EfficientNet, and finally emphasizing on the importance of transformer models in vision. The dataset used for this project was sourced from kaggle (reference-2) containing images labeled with three emotions: “angry”, “happy”, and “sad”. Although the range of emotions used in this project is very less, this is sufficient to demonstrate the research objective.

Before diving into the modeling aspect, several preprocessing steps were performed (discussed in the following sections) to enhance the robustness and generalizability of the models. The data was then split into training, validation and testing sets for training, evaluation and testing respectively. Each model was trained on the same train-test split to ensure consistency, regularization techniques like batch normalization, dropout, and L1-L2 regularizers were incorporated to avoid overfitting. Several metrics were tracked during the training process and extensive logging methods were implemented for better control and debugging.

Finally, the models were tested by plotting confusing metrics and tracked metrics to perform comparative analysis.

# Methodology

## *Dataset Specifications*

The dataset consists of 9077 images, split into 6799 training and 2278 testing images making it a 75-25 split. A quick peek into the data is shown in Fig.1. As discussed earlier, it contains images labeled into 3 categories.


Fig. 1 - Sample Data

## *Data Pre-processing & Augmentation*

The data was loaded and converted into a Tensorflow dataset to be able to work with the neural networks making the data into batches, converting the target variable into categorical values, and shuffling the data to remove any data collection bias. To standardize the data, built-in tensorflow layers “Resizing” and “Rescaling” were used to resize the images to 256*256 and rescaled by dividing with 255 to make the pixel values to have values between 0 and 1.

For image detection/classification, it is only important to detect that the image is present and the detection should not be affected due to the positional variance of the image, meaning that it should not matter if the image is rotated or shifted by a bit. To simulate this, data augmentation methods such as “RandomFlip”, “RandomRotation”, and “RandomContrast” were incorporated as part of the pre-processing. The images after the augmentation are shown in Fig. 2.


Fig. 2 - Augmented Images

## *Model Training*

For training a neural network efficiently, certain aspects such as optimizer, loss function, and metrics were defined. For controlling the gradients, the learning rate was set to an initial value of 0.001 and reduced according to the “learning_rate_scheduler” along with Adam as the optimizer. The optimizer helps from overshooting and oscillating about the minimum point by controlling the learning rate. As the problem statement is a multi-class classification, Categorical Cross Entropy was used as the loss function. As used for any classification problem, some of the key metrics such as Precision, Recall, F1-Score, AUC, and Categorical Accuracy were tracked during the training process for both training and validation datasets to understand the performance of the model.

Leveraging the tensorflow callbacks, certain logging features were implemented such as CSV Logger to log all the metrics to a csv file , Early Stopping to stop the training process earlier than the defined epoch if there is no improvement on the metrics, Model Checkpoint to save the best model weights up to the current epoch, and Reduce on Plateau to reduce the learning rate when there the loss reaches a flat surface. All of these logging features were then exported to tensorflow boards for visualization (shown later).

## 1. *Baseline Model*

In general, if a model with less complexity can solve a problem, it is often chosen for the reasons of computation and interpretability and hence a baseline model with 2 convolutional layers followed by 2 fully connected neural network dense layers was implemented as shown in Fig.3.1. At each layer, batch normalization, dropouts and were done for regularization and max pooling was done to reduce the dimensions of the output features. The model was trained for 30 epochs and the loss for both train and validation dataset was plotted as shown in Fig. 3.2. For a baseline model, the performance was not bad obtaining an accuracy of 81%.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| sequential_1 (Sequential) | (None, 256, 256, 3) | 0 |
| conv2d (Conv2D) | (None, 254, 254, 6) | 168 |
| batch_normalization (BatchNormalization) | (None, 254, 254, 6) | 24 |
| max_pooling2d (MaxPooling2D) | (None, 127, 127, 6) | 0 |
| dropout (Dropout) | (None, 127, 127, 6) | 0 |
| conv2d_1 (Conv2D) | (None, 125, 125, 16) | 880 |
| batch_normalization_1 (BatchNormalization) | (None, 125, 125, 16) | 64 |
| max_pooling2d_1 (MaxPooling2D) | (None, 62, 62, 16) | 0 |
| dropout_1 (Dropout) | (None, 62, 62, 16) | 0 |
| flatten (Flatten) | (None, 61504) | 0 |
| dense (Dense) | (None, 100) | 6150500 |
| batch_normalization_2 (BatchNormalization) | (None, 100) | 400 |
| dense_1 (Dense) | (None, 10) | 1010 |
| batch_normalization_3 (BatchNormalization) | (None, 10) | 40 |
| dense_2 (Dense) | (None, 3) | 33 |

Total params: 6153119 (23.47 MB)
Trainable params: 6152855 (23.47 MB)
Non-trainable params: 264 (1.03 KB)

Fig. 3.1 - Baseline CNN Architecture

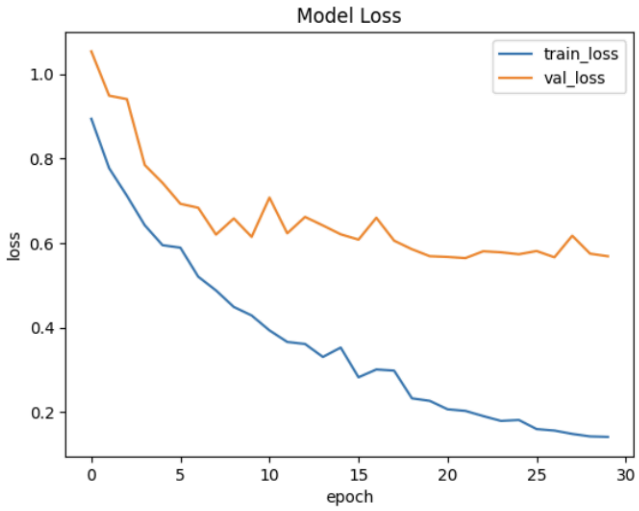


Fig. 3.2 - Training loss of Baseline CNN

## 2. *ResNet50*

Moving on to a slightly complex architecture, ResNet50 (reference-3) was implemented both using a pre-trained model and from scratch. ResNet50 is a deep CNN consisting of 50 layers, including convolutional layers, pooling layers, and fully connected layers as shown in Fig. 3.3. ResNet50 is known for its residual learning framework, which allows it to effectively train very deep networks while mitigating the vanishing gradient problem. The difference between the from scratch and pre-trained is that the pre-trained model has been pre-trained on the ImageNet dataset, thereby having more context when used for transfer learning in various computer vision applications. The model was only trained for 18 epochs (Fig.3.4) as opposed to the defined 30 epochs as there was no performance further. The performance did not improve compared to the baseline mode in terms of both accuracy and f1-score. When it comes to a choice of Baseline vs ResNet50, it is optimal to choose the Baseline as it has similar performance with much less complexity which can save computational resources. Comprehensive comparison is performed in later sections.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| resnet50 (Functional) | (None, 8, 8, 2048) | 23587712 |
| global_average_pooling2d ( GlobalAveragePooling2D) | (None, 2048) | 0 |
| dense (Dense) | (None, 1024) | 2098176 |
| batch_normalization (Batch Normalization) | (None, 1024) | 4096 |
| dense_1 (Dense) | (None, 128) | 131200 |
| dense_2 (Dense) | (None, 3) | 387 |

Total params: 25821571 (98.50 MB)
Trainable params: 2231811 (8.51 MB)
Non-trainable params: 23589760 (89.99 MB)

Fig. 3.3 - ResNet50 Architecture

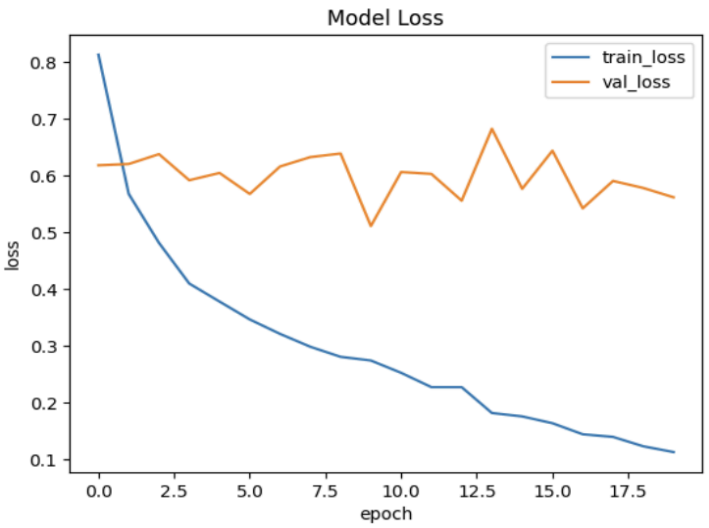


Fig. 3.4 - Training loss of ResNet50

## 3. *EfficientNetB4*

As the performance of ResNet50 was not up to the mark, a more complex model was needed. EfficientNet (reference-4) models are known for their compounding scaling method that uniformly scales the network's width, depth, and resolution. Even though It is deeper and wider compared to ResNet50, it is more parameter-efficient and has only 19.6 million parameters (Fig. 3.5) compared to 25.6 million of ResNet50. From the family of EfficientNet, B4 was selected as this model is optimal with rest to number of parameters. Similar to ResNet50, the model training stopped after 18 epochs due to a plateau of performance improvement. This model performed slightly better, obtaining an accuracy of 83% compared to 81% of previous models. The key thing to notice is that having less parameters and less size compared to ResNet50, it performs better and hence it can be a good choice with respect to deployment.

| Layer (type) | Output Shape | Param # |
|---|---|---|
| efficientnetb4 (Functional) | (None, 8, 8, 1792) | 17673823 |
| global_average_pooling2d (GlobalAveragePooling2D) | (None, 1792) | 0 |
| dense (Dense) | (None, 1024) | 1836032 |
| batch_normalization (BatchNormalization) | (None, 1024) | 4096 |
| dense_1 (Dense) | (None, 128) | 131200 |
| dense_2 (Dense) | (None, 3) | 387 |

Total params: 19645538 (74.94 MB)
Trainable params: 1969667 (7.51 MB)
Non-trainable params: 17675871 (67.43 MB)

Fig. 3.5 - EfficientNetB4 Architecture

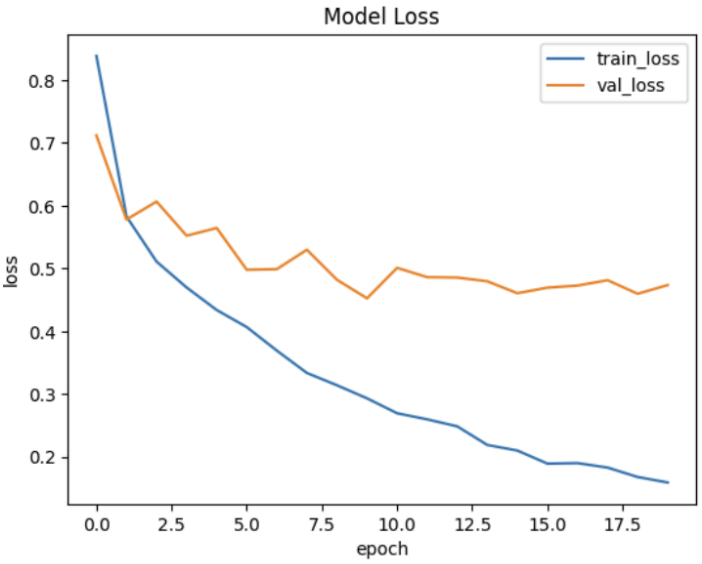


Fig. 3.6 - Training loss of EfficientNetB4

## 4. *Vision Transformer*

Transformers were primarily used for natural language processing. A recent paper (reference -5) has shown how transformers can be used for image related tasks. In NLP, the input is passed as embeddings and a similar approach is followed for images and the input image is first converted into sequence of image patches as shown in Fig. 3.7. Each patch is then treated as an embedding to input to the transformer and the rest of the architecture is used as is. In this project, a Vision Transformer (ViT) which was pre-trained on ImageNet dataset was implemented. As shown in the Fig. 3.8, ViT is a complex model with very high parameters and hence requires huge computational resources. In just 6 epochs (less due to computational restrictions), the ViT model out performed other models by achieving an accuracy of 90% compared to 83% of EfficientNetB4. The final fine-tuned model had a size of 950MB and for deploying (reference-6) the model, quantization was done to reduce the size to 90MB without losing any performance.

Fig. 3.7 - Sample Input data of ViT

| Layer (type) | Output Shape | Param # |
|---|---|---|
| input_1 (InputLayer) | [(None, 224, 224, 3)] | 0 |
| sequential_2 (Sequential) | (None, 3, 224, 224) | 0 |
| vit (TFViTMainLayer) | TFBaseModelOutputWithPooling(last_hidden_state=(None, 197, 768), pooler_output=(None, 768), hidden_states=None, attentions=None) | 86389248 |
| tf.__operators__.getitem (SlicingOpLambda) | (None, 768) | 0 |
| dense (Dense) | (None, 3) | 2307 |

Total params: 86391555 (329.56 MB)
Trainable params: 86391555 (329.56 MB)
Non-trainable params: 0 (0.00 Byte)

Fig. 3.8 - ViT Architecture

## *Model Evaluation & Comparison*

| Metrics/Model | Baseline CNN | ResNet50 (from scratch) | ResNet50 (pre-trained) | EfficientNetB4 | ViT |
|---|---|---|---|---|---|
| **Precision** | 0.82 | 0.79 | 0.82 | 0.85 | 0.9 |
| **Recall** | 0.79 | 0.72 | 0.79 | 0.81 | 0.88 |
| **Accuracy** | 0.81 | 0.76 | 0.8 | 0.83 | 0.89 |
| **F1- Score** | 0.81 | 0.78 | 0.8 | 0.83 | 0.89 |

Fig. 4.1 - Model Comparison Metrics

To have a fair comparison, all the models were evaluated on the same testing data on the same metrics such as Precision, Recall, F1-Score and Categorical Accuracy. The Baseline model was pretty good to start with and performed at the level of ResNet50. EfficientNetB4 had an improvement over the ResNet50 even with less number of parameters making it the best performing CNN architecture. And finally, the Vision Transformer outperforms the other models by a significant margin in all the metrics as shown in Fig.4.1 and Fig. 4.2

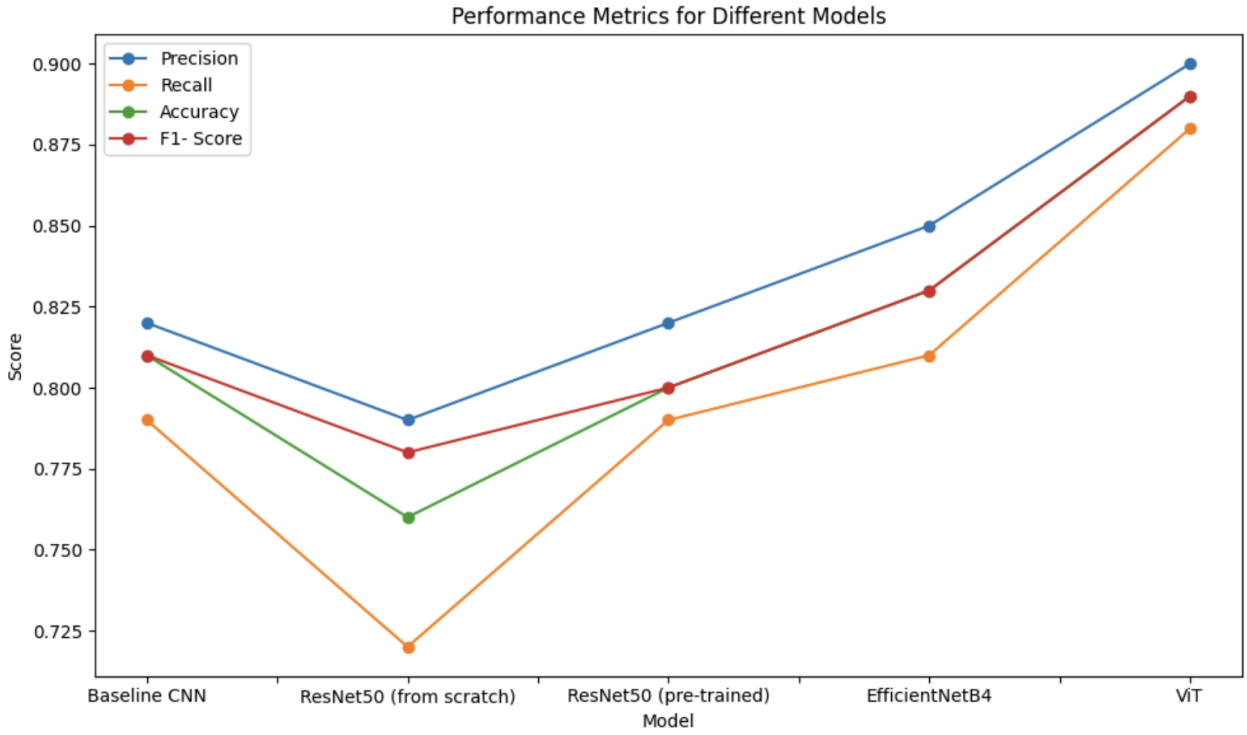


Fig. 4.2 - Visual Model Comparison

# Results

The Fig. 5.1 shows the sample predictions of the final quantized ViT model on the testing dataset. It predicts incorrectly for 1 data point. However, upon close inspection, the image that is predicted incorrectly actually looks “sad” to a naked eye as supposed to true label “happy”. Hence, if a human cannot classify it accurately, it is only natural for the model to do so. Having such ambiguous data points might make the model mis-classify images.


Fig. 5.1 - Dataset Predictions


Fig. 5.2 - Generalized Predictions

The Fig. 5.2 shows the predictions of the deployed model on a general image other than the dataset used in the project. It is evident that when the image has clear features corresponding to the class label (happy in this case), the model does a good job making more generalized predictions. Moreover, as the model was quantized and has a very small size, the time taken to make the prediction is also less as shown in fig. 5.2.

# Conclusion

In this research project, the effectiveness of different deep learning architectures for human emotion detection in images was explored. Models with different architectures such as  Baseline CNN, ResNet50, EfficientNetB4, and Vision Transformer were compared on a  dataset containing images labeled with three emotions: "angry", "happy", and "sad".

Results show that the Vision Transformer (ViT) outperformed all other models, achieving an accuracy of 90% on the testing dataset. EfficientNetB4 also showed promising results, outperforming ResNet50 with fewer parameters, making it a more efficient choice for tasks where computational resources are a constraint.

This study highlights the importance of choosing the right model architecture for image-related tasks, as more complex models like ViT and EfficientNetB4 can significantly improve performance. Additionally, it demonstrates the potential of transformer models in computer vision tasks, showcasing their adaptability and effectiveness in handling image data. Moreover, it also emphasizes the importance of quantization to reduce the model size and complexity for efficiency in deployment.

Overall, this research contributes to the advancement of emotion detection systems by showcasing the capabilities of transformer models in image tasks, paving the way for more precise and efficient emotion recognition systems in real-world scenarios.

# References


1. GitHub link to the project https://github.com/RohitMacherla3/human-emotion-detection-CV
2. Dataset - https://www.kaggle.com/datasets/muhammadhananasghar/human-emotions-datasethes/data
3. ResNet Paper - https://arxiv.org/abs/1512.03385
4. EfficeintNet Paper - https://arxiv.org/abs/1905.11946v5
5. Vision Transformer Paper - https://arxiv.org/abs/2010.11929
6. App Deployment - https://human-emotion-detection.streamlit.app/