Emotion Recognition in Facial Images using Convolutional Neural Networks

1. Abstract

Facial image recognition refers to the use of computer algorithms to analyze and identify emotions displayed on a person's face.This report explores the use of facial emotion recognition through deep learning using Convolutional Neural Networks (CNNs). The aim is to classify emotions from facial features such as the position and shape of the mouth, eyes, and eyebrows. Three datasets from Kaggle were selected, ensuring they had enough classes and images to train the model effectively, and all had different numbers of classes. The MobileNet v2, ShuffleNet v2, and ResNet 18 architectures were used to analyze how different hyperparameters and models impact the accuracy of facial emotion recognition. We selected three datasets and limited their size as specified. FER-2013 dataset was reduced to 5000 images with 3 classes, AffectNet-HQ dataset to 10000 images with 8 classes, and Facial Emotion Recognition tiny dataset to 15000 images with 13 classes. After that, we partitioned each dataset into training, testing, and validation subsets. The training and testing data were distributed in an 80/20 ratio, and the training data was further divided into an 80/20 ratio for training and validation all of the datasets above mentioned are available on kaggle.

2. Introduction

Facial emotion recognition is the process of using artificial intelligence to analyze and identify emotions displayed on a person's face..Facial expressions are important for conveying emotions and provide insights into a person's feelings that may not be apparent from verbal communication alone. Research suggests that up to 70% of communication is nonverbal, and facial expressions play a significant role [1]. By analyzing facial expressions with technology, we can better understand emotional states and improve communication and human interaction. It also has the potential to revolutionize fields like healthcare, education, and marketing. Facial emotion recognition through deep learning aims to classify emotions from an image of a person's face using Convolutional Neural Networks.CNN is used to process the image data and automatically learn and extract relevant features from the facial features, such as the position and shape of the mouth, eyes, and eyebrows. These learned features are then used to classify the person's emotional state.One of the associated challenges with facial emotion recognition is dataset bias, which can occur when the dataset used for training is not diverse enough and not representative of the real-world population, resulting in poor performance and generalization of the model. This challenge has been addressed in the literature by using diverse and representative datasets, and by employing data augmentation techniques to increase the diversity of the dataset.Another challenge is overfitting, which happens when the model learns the training data too well and does not generalize well to unseen data. This was addressed by using early stopping technique during training. Additionally, we carefully selected hyperparameters to reduce overfitting.While the existing techniques such as using a diverse dataset and using regularization, dropout and cross-validation can improve facial emotion recognition accuracy, they require more computing resources and time for training.One way to address the issue mentioned in facial emotion recognition which we have used is by using transfer learning, which involves leveraging pre-trained models on large datasets and fine-tuning them on smaller, more diverse datasets. This approach allows the model to learn general features from the pre-trained model and then fine-tune them on the smaller dataset to adapt to the specific task of facial emotion recognition. Transfer learning has been shown to improve performance on smaller datasets and can reduce the impact of dataset bias.

3. Methodologies

3.1. Datasets

For this project, three datasets were selected, each with different numbers of classes and varying numbers of images per class. The choice of these datasets is motivated by their large amount of annotations, diversity of the content and classes. All three datasets were obtained from Kaggle.The FER2013 [2] dataset consists of 35,887 grayscale images with a size of 48x48 pixels. Each image is labelled with one of seven emotions, including anger, disgust, fear, happiness, sadness, surprise, or neutral. The dataset was collected from social media websites like Flickr, Google Images, and Yahoo Images, using both human annotators and automated processes.
AffectNet is another dataset of facial expressions that contains over 1 million images with a size of 224x224 pixels. The dataset is labelled with one of 11 emotions, including

neutral, happy, sad, surprise, fear, disgust, anger, contempt, valence, arousal, and dominance. The dataset was collected using a combination of human annotators and automated processes to be more diverse and balanced than previous facial expression datasets. A subset of the AffectNet dataset, called AffectNet HQ [3], contains 414,799 high-quality images that were manually selected. This dataset's class representations are more complex than FER2013, with additional classes representing more subtle emotions and affective states, making it a more challenging dataset for building and testing models for facial expression recognition.

The FER Tiny [4] dataset is a publicly available dataset with 15,000 greyscale images of size 48x48 pixels. Each image is labelled with one of thirteen emotions, including anger, disgust, fear, happiness, sadness, surprise, neutral etc. The dataset was collected using a web-based data collection tool, with each image labelled by at least three human annotators. The FER Tiny dataset was designed for training and testing lightweight models for facial emotion recognition on resource-constrained devices, such as smartphones or embedded systems.

Dataset	Total Images	Total Classes	Image Size	Image Format
Facial emotion recognition tiny [4]	15000	13	224x224	Grayscale
AffectNet-HQ [3]	10000	8	224x224	Grayscale
FER-2013 [2]	5000	3	224x224	Grayscale

Table 1. Datasets

To manage the computational complexity and memory usage during training, the sizes of the three datasets were limited, as shown in Tab. 1. Various pre-processing and filtering techniques were applied to standardize the input images and reduce the size of large datasets. Each image in the dataset was resized to a 224x224 resolution and converted to grayscale to standardize the image sizes and reduce the dimensionality of the dataset. To improve model performance and stability during training, the mean and standard deviation of the pixel values in each of the datasets were calculated and used to normalize the pixel values in the image dataset. Each dataset was divided into training, testing, and validation subsets, with the training and testing data being split in an 80/20 ratio, and the training data being further divided into an 80/20 ratio for training and validation.

3.2. CNN Models

MobileNetV2 [5] is a deep learning model that is optimized for use on mobile and embedded devices. Its architecture is based on an inverted residual structure that connects the residual layers to the bottleneck layers,

Figure 1. Images from AffectNetHq Dataset

Figure 2. Images from FER2013 Dataset

Figure 3. Images from Tiny Dataset

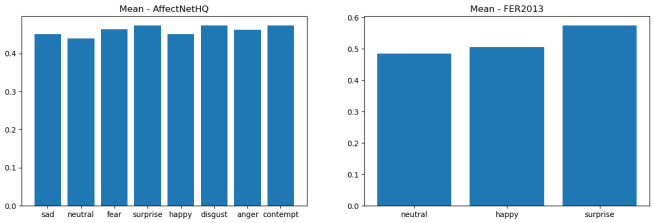

Figure 4. Mean Values for Fer and AffectNetHq Dataset

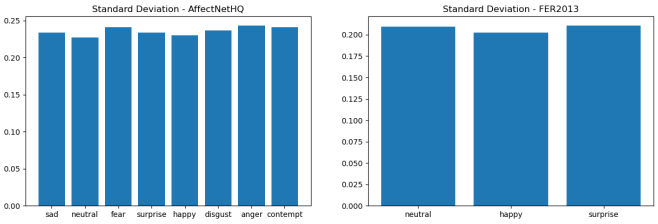

Figure 5. Standard Deviation for Fer and AffectNetHq Dataset

allowing for efficient information flow between layers. The model also employs depth wise separable convolutions, which are used to reduce the computational cost while maintaining accuracy. It contains an initial fully convolution layer with 32 filters, followed by 19 residual bottleneck layers. Overall, MobileNetV2 is designed to balance accuracy and efficiency, making it an excellent choice for mobile and embedded vision applications.

ShuffleNetV2 [6] is a deep neural network architecture designed for mobile and embedded devices, which achieves high accuracy while keeping computational cost low. It uses a channel shuffling operation, enabling information exchange between channels, and depth wise separable and point wise group convolutions to reduce parameters and computational cost. Its architecture includes a stem block, multiple shuffle units, and a fully connected layer. ShuffleNetV2 achieves state-of-the-art performance in image classification benchmarks with small model size.

ResNet-18 [7] is a type of neural network architecture that has been specifically designed to be used for image classification tasks. It comprises a total of 18 layers, which includes a convolution layer followed by four residual blocks. Each block has two or three convolution layers, as well as a skip connection that facilitates gradient flow and a batch normalization layer. To complete the network, here is a global average pooling layer followed by a fully connected layer that handles the classification. ResNet-18 is popular due to its impressive performance in image classification tasks, as well as its relatively low computational cost.

The chosen models strike a balance between computational complexity and accuracy, making them well-suited for various use cases. ResNet18 is ideal for more complex image classification tasks, whereas MobileNetV2 and ShuffleNetV2 are better suited for lightweight and mobile applications. This flexibility enables the selection of the most appropriate model for specific needs while still achieving high levels of accuracy. To further understand the computational complexities of the selected CNN models, they were evaluated based on the wall clock time required for one-epoch training and the number of FLOPS calculation performed during training and validation phases. The metrics for each model is in shown in Tab. 2 .

Architecture Name	Dataset	Time per Epoc
MobileNet v2	FER2013	30.66 seconds
MobileNet v2	AffectNetHq	60.36 seconds
MobileNet v2	Tiny	77.58 seconds
ShuffleNet v2	FER2013	14.83 seconds
ShuffleNet v2	AffectNetHq	15.13 seconds
ShuffleNet v2	Tiny	45.11 seconds
ResNet 18	FER2013	22.51 seconds
ResNet 18	AffectNetHq	55.39 seconds
ResNet 18	Tiny	85.52 seconds

Table 2. Flops and Average Time per Epoc

In terms of computational complexity, ShuffleNet v2 has the lowest number of FLOPS calculations (3.978 billion), making it the most computationally efficient model. MobileNet v2, on the other hand, has the highest number of FLOPS calculations (4018.5 billion), making it the most computationally demanding model. Resnet 18 falls in the middle in terms of FLOPS calculations (990.43 billion). The time required for one-epoch training varied across the models and datasets. ShuffleNet v2 had the fastest training time per epoch for all three datasets as it had the lowest number of flops, whereas Mobilenet v2 took the most amount of time for FER2013 and AffectNetHQ dataset. Overall, the selection of the model depends on the specific use case and the trade-off between accuracy and computational complexity. In terms of computational efficiency ShuffleNet v2 is the best choice. On the other hand, if accuracy is the top priority, then ResNet 18 is the best choice, despite its higher computational demands. MobileNet v2 offers a good balance between the two.

3.3. Optimization Algorithm

In order to train all of our 9 CNN models we used the Adam optimizer [8] which is an adaptive optimization algorithm used in deep learning models, particularly in image classification tasks. It adjusts the learning rate for each parameter based on historical gradient information, allowing for faster convergence and avoiding overshooting or undershooting. It uses first and second moments of gradients for efficient updates and is robust to noisy gradients due to an exponentially decaying average of past gradients and squared gradients. Additionally, it includes a regularization term to prevent overfitting. These features make Adam optimizer a favourable choice for our deep learning image classification models.
To optimize the models during training CrossEntropyLoss [9] function proved to be a good fit for multi-class classification tasks. It computes the negative log likelihood between the predicted class probabilities and the true class probabilities, which penalizes the model for assigning low probabilities to the correct class. Compared to other loss functions like mean squared error or mean absolute error, CrossEntropyLoss function handles imbalanced class distributions well, which is often the case in image classification where certain classes may have more or fewer samples. Secondly, it provides better gradients for backpropagation, which can result in faster convergence and better performance. Overall, these were the reason which motivated the use of CrossEntropyLoss function in this project. While using both of these methods the learning rate and the batch size were fixed to 0.001 and 128 respectively, keeping in mind the available computational resources.

To guarantee that the model was both optimized and validated, a three-pronged approach was employed. Ini-

tially, the model was trained using the training dataset. Subsequently,the hyperparamter learning rate was fine tuned by evaluating the model's performance on the validation set. The learning rate ranged from 0.1 to 0. 0.000000001. Finally, the model's capability to generalize on unseen data was assessed by testing it on the test set. The results indicated that 0.0001 was the best learning among all the options. Moreover, accuracy, precision, F1 score, support, recall and confusion matrix were used to evaluate the performance of all the 9 models.

4. Results

4.1. Experimental Setup

We trained and validated our facial recognition system collected from three different dataset. The training and testing data have been split into a ratio of 80/20 with the training data being further split into an 80/20 ratio for training and validation. To optimize our models, we used a combination of data augmentation techniques and hyperparameter tuning. We have added transforms to each image in the training set to increase its variability. Additionally, we have also used transfer learning with a pre-trained convolutional neural network.We experimented with a range of hyperparameters, including the learning rate, batch size, and number of training epochs. We have explored different combinations of hyperparameters and selected the set of hyperparameters that achieved the highest validation accuracy.To validate our models, we evaluated their performance on the validation set using standard classification metrics such as precision, recall, and F1-score. We achieved our results with a learning rate of 0.001, batch size of 128, , and 25 training epochs as we found that these hyperparameters provided a good balance between model complexity and generalization performance on our dataset.

4.2. Main Results

After implementing ResNet, MobileNetV2, and ShuffleNetV2 CNN models on three different datasets, namely FER2013, Tiny, and AffectNetHQ, it can be concluded that ResNet18 provided the best performance on the datasets, followed by ShuffleNetV2 and MobileNetV2. The evaluation metrics used to compare the models were precision, recall, F1-score, confusion matrix, and accuracy. From the results, it can be observed that the FER2013 dataset had the best results in terms of accuracy, precision, recall, and F1-score, likely because of the larger number of learning examples and well-balanced data. The AffectNetHQ dataset had the lowest performance, with the Tiny dataset performing slightly better. Overall, ResNet18 was the best-performing model on all three datasets, with the highest accuracy and F1-score.

Looking at the FER2013 dataset in Tab. 3 , MobileNetV2 outperforms ResNet18 and ShuffleNetV2 on all metrics. MobileNetV2 achieves an accuracy of 72.25, which is higher than the accuracy of ResNet18 and ShuffleNetV2 at 70.00 and 66.86, respectively. MobileNetV2 also has a higher precision, recall, and F1-score compared to ResNet18 and ShuffleNetV2. Moreover, the MobileNetV2 model has a validation accuracy of 75.77, which is higher than the validation accuracy of ResNet18 and ShuffleNetV2 at 72.31 and 69.22, respectively.

FER2013	ResNet18	ShuffleNetV2	MobileNetV2
Accuracy	0.70	0.67	0.72
Precision	0.76	0.67	0.74
Recall	0.70	0.67	0.72
F1-Score	0.70	0.67	0.72
Train Accuracy	91.90	90.87	82.73
Test Accuracy	70.00	66.86	72.25
Validation Accuracy	72.31	69.22	75.77

Table 3. Comparision between metrics on the FER2013 dataset

On the AffectNetHQ dataset we can see from Tab. 4, ResNet18 outperforms MobileNetV2 and ShuffleNetV2 on all metrics. However, the performance of all models is relatively poor compared to their performance on FER2013. ResNet18 achieves an accuracy of 56.53, which is the highest among the three models. ShuffleNetV2 and MobileNetV2 achieve accuracies of 46.80 and 50.75, respectively.

AffectNetHQ	ResNet18	ShuffleNetV2	MobileNetV2
Accuracy	0.56	0.47	0.51
Precision	0.61	0.47	0.52
Recall	0.56	0.47	0.51
F1-Score	0.56	0.46	0.49
Train Accuracy	97.97	92.47	88.81
Test Accuracy	56.35	46.80	50.75
Validation Accuracy	54.25	47.50	50.25

Table 4. Comparision between metrics on the AffectNetHQ dataset

Based on Tab. 5 , it is evident that ResNet18 is the best performing model on the Tiny dataset. The model achieves an accuracy of 68.33., which is significantly higher than the accuracies of MobileNetV2 and ShuffleNetV2 at 66.99 and 58.70, respectively.

The graphs in Figures 6 and 7 shows the training and validation loss and accuracy for the resNet18 architecture, showing the performance of pre-trained and non-pretrained models. The results indicate that after transfer learning, there was a significant improvement in the accuracy of the

Tiny	ResNet18	ShuffleNetV2	MobileNetV2
Accuracy	0.67	0.59	0.67
Precision	0.70	0.63	0.70
Recall	0.70	0.59	0.67
F1-Score	0.70	0.59	0.67
Train Accuracy	95.11	87.02	79.55
Test Accuracy	68.33	58.70	66.99
Validation Accuracy	68	58.34	65.34

Table 5. Comparision between metrics on the Tiny dataset

model. The pre-trained resNet18-FER model achieved an accuracy of 78.82, compared to 76.18 for the non-pretrained model. Similarly, the pre-trained resNet18-Tiny model outperformed its non-pretrained counterpart, achieving an accuracy of 79.43 compared to 68.33. The graph demonstrates the effectiveness of transfer learning in improving model performance and highlights the importance of leveraging pre-trained models for improved accuracy.

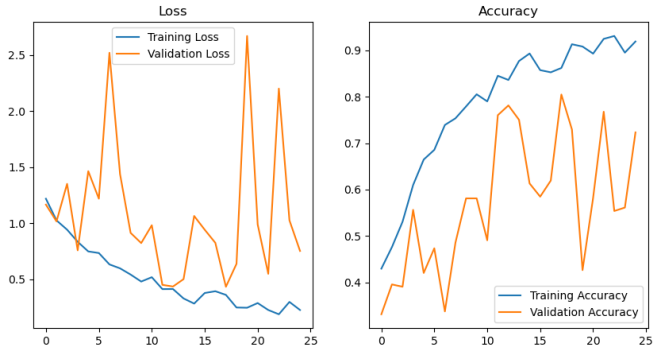

Figure 6. Validation and training loss and accuracy's for FER

Figure 7. Validation and training loss and accuracy's for FER pre-trained

4.3. Ablative Study

After Hyperparameter Optimisation, the highest accuracy was achieved with ResNet-18 architecture on the FER2013 dataset with an accuracy of 82.35. For all datasets, we used a batch size of 128, and after experimenting with a number of epochs, we have found epochs of 25 works well with our datasets. We used a learning rate of 0.001 after using hyperparamter tuning to find the optimal learning rate,results of which are seen in as seen in the figure 8 and Adam optimizer. We observed that a higher learning rate caused the model to converge quickly to a local minimum of the training loss function, but the model failed to generalize well to new data. To address this, we increased the learning rate from 0.0001 to 0.001. Additionally, we found that a very large batch size led to overfitting, as the model would only see a limited number of unique examples in each iteration, while a very small batch size resulted in noisy gradients. We experimented with different batch sizes, and a batch size of 128 seemed to work best for all our models. Early stopping was also incorporated to avoid overfitting, requiring us to monitor the validation loss throughout training and terminate the training process when the validation loss stopped improving. Lastly, we observed that the fine-tuned models attained higher accuracies during the initial epochs, likely due to the advantageous weights they were initialized with, while the deep-tuned models exhibited notably inferior performance compared to the other models, likely due to a modification in the weights of the convolution layers. The ablation study provided further insights into the impact of different hyperparameters on model performance. We found that changing the number of classes for training, number of images per class, and learning rate had a significant impact on model performance.

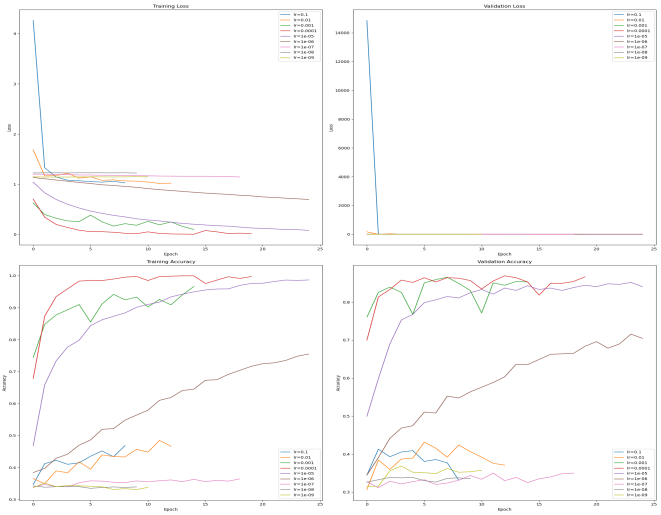

Figure 8. Hyperparameter tuning for ResNet on the FER dataset

References

[1] https://online.utpb.edu/about-us/articles/communication/how-much-of-communication-is nonverbal/. [Online]. Available: https://online.utpb.edu/about-us/articles/communication/how-much-of-communication-is-nonverbal/ 1

[2] https://www.kaggle.com/datasets/msambare/fer2013. [Online]. Available: https://www.kaggle.com/datasets/msambare/fer2013 1, 2

[3] https://www.kaggle.com/datasets/tom99763/affectnethq. [Online]. Available: https://www.kaggle.com/datasets/tom99763/affectnethq 2

[4] https://www.kaggle.com/datasets/sakuraisana/facial-emotion-recognition tiny?select=validation. [Online]. Available: https://www.kaggle.com/datasets/sakuraisana/facial-emotion-recognition-tiny?select=validation 2

[5] https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification 8febb490e61c. [Online]. Available: https://towardsdatascience.com/review-mobilenetv2-light-weight-model-image-classification-8febb490e61c 2

[6] https://sh-tsang.medium.com/reading-shufflenet-v2-practical-guidelines-for-e-fficient-cnn-architecture-design-image 287b05abc08a. [Online]. Available: https://sh-tsang.medium.com/reading-shufflenet-v2-practical-guidelines-for-e-fficient-cnn-architecture-design-image-287b05abc08a 3

[7] https://towardsdatascience.com/understanding-and-visualizing-resnets 442284831be8. [Online]. Available: https://towardsdatascience.com/understanding-and-visualizing-resnets-442284831be8 3

[8] https://towardsdatascience.com/complete-guide-to-adam-optimization 1e5f29532c3d. [Online]. Available: https://towardsdatascience.com/complete-guide-to-adam-optimization-1e5f29532c3d 3

[9] https://towardsdatascience.com/cross-entropy-loss-function f38c4ec8643e. [Online]. Available: https://towardsdatascience.com/cross-entropy-loss-function-f38c4ec8643e 3