Deep Learning: Facial Emotion Detection

By
Yi-Hua, Amber, Chang
MIT Applied Data Science
Dec 10th, 2022

Abstract

Affective computing or Emotion AI stands for the study and development of technologies that can analyze human emotions using body gestures, facial expressions, voice tone, and so forth and react accordingly to them. Facial Emotion Recognition (FER) is critical in human-machine interaction, and recent research has suggested that roughly 50% of communication of sentiments occurs through facial expressions and other visual cues. Hence, training a model to identify facial emotions accurately is essential to developing emotionally intelligent behaviors in machines with AI capabilities. Some automatic facial expression recognition applications that require human behavior understanding include healthcare, branding exposure, customer services, and travel recommendations. This project aims to use Deep Learning and Artificial Intelligence techniques to create a computer vision model that can accurately detect facial emotions. The model should be able to perform multi-class classification on images of facial expressions to classify the expressions according to the associated emotion.

Introduction

Deep learning models, such as Convolutional Neural Networks (CNNs), have shown the potential to accurately identify emotions thanks to their computational efficiency and feature extraction capabilities. These benefits make them suitable for image classification.

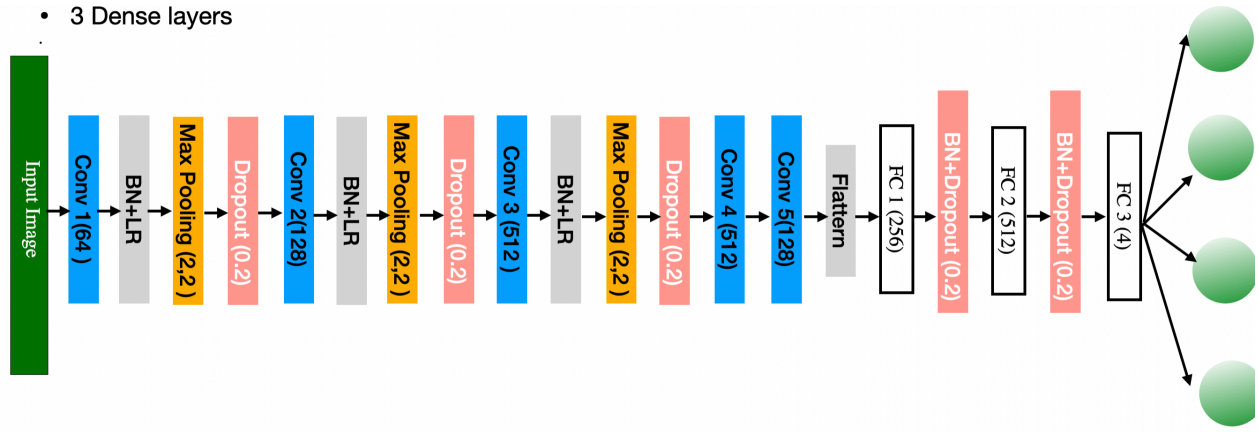
Nonetheless, accurate Facial Emotion Recognition by computer vision models remains challenging due to the heterogeneity of human face poses and some natural conditions. Thus, how we achieve high accuracy by building a model, which learns from a relatively small amount of dataset, becomes the primary purpose throughout the whole experimentation.

Firstly, we will adopt data augmentation during data preprocessing to address the limited number of data samples. Secondly, to significantly improve our model's accuracy, we will conduct various experiments to explore different ways of optimizing the convolutional neural network. For example, we will try different optimization algorithms and a neural network's architecture design. Thirdly, to tackle efficiency due to the scarcity of computational resources, we aim to explore further some pre-trained models, such as VGG16, ResNet v2, and Efficient Net**.** We found that by designing a complex neural network architecture and tuning our model's hyper-parameters, we have gained state-of-the-art results, achieving approximately 74.22% accuracy.

Executive Summary

A. *Data Exploration and Augmentation*

Figure 1 Model 3 architecture. A face expression image is fed into the model. The five convolutional blocks (Conv) extract high-level features of the image and the fully-connected (FC) layers classify the emotion of the image.

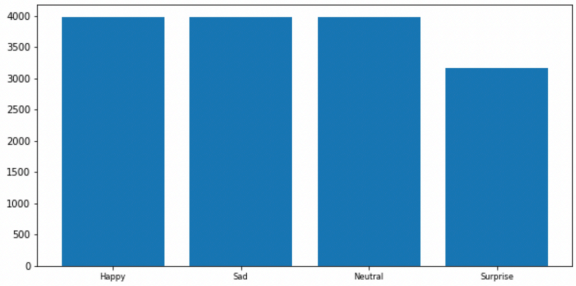


In training our models, we have a training dataset of 15109, a validation dataset of 4977, and a testing dataset of 128 images. All the images are of 4 different emotions (happiness, sadness, neutral, and surprise). It has been shown in Kaggle that people have achieved an accuracy of 65-68% when trying to recognize these emotions.

After checking the distribution of four classes by visualizing through the histogram of figure 2, we observed that the frequency for three categories, "Happy," "Sad," and "Neutral," is roughly distributed evenly with approximately 4000 samples. In contrast, the number for the category "SURPRISE" is slightly less than the others, with only around 3000 samples. This distribution indicates that the class imbalance problem is not severe, so we do not need to apply resampling further to address the bias issue derived from class imbalance.

Data augmentation is a technique used to increase the data available for training a machine learning model. We stick to the same data augmentation techniques during the data preprocessing phase in our experiments with various CNN models. We utilized this technique to account for the variability in facial expression recognition, which means that the images are brightened ± 20%, sheared ± 30%, and flipped horizontally randomly. Then, the images are cropped into 48x48. Finally, each pixel is divided by 255 to normalize the image

Figure 2 Class Distribution



B. *Training*

Cross-entropy loss measures how well a machine learning model can predict the correct output. We ran all experiments for 20-35 epochs to optimize cross-entropy loss, which means we ran the model several times to see how well it could predict the correct output. In the following sections, we used the same optimizer, **Adam,** but varied the learning steps from 0.001 to 0.003. We chose to use Adam because it combines both advantages of Momentum and AdaGrad to use a weighted sum of the current gradient with past gradients to be less erratic and adaptively set step size for each network weight. To ensure that the results are accurate, we keep all other parameters the same. We are only changing epochs and learning steps, not other factors, such as optimizing algorithms and learning rate scheduler. All experiments are run with an early stopping point and adopted learning rate to avoid the explosion or the vanishing of gradient descent.

C. Evaluating

We evaluate each model using validation accuracy and confusion matrix. The success of a model depends on its accuracy in terms of the classification of images. Specifically, the measures of success in image classification hinge on reducing the misclassification rate in test data. Here, we will introduce a **confusion matrix and classification report** to assess each model's performance in classifying an image accurately. Under this hood, we are especially interested in the **f1-score as our critical measurement to** determine which model best mitigates the misclassification problem.

D. Design of architectures

While it is crucial to have the proper optimizers in training our deep learning models, the backbone of a successful computer vision model is the design of a convolution neural network. Our experiment intends to find the best design of architecture to maximize performance in classifying facial recognition associated with emotions.

D-1 Neurons, Layers, Activation

A neural network is arranged with multiple simple units arranged in layers. Complicated decisions are broken into as simple questions solved by each unit in a hierarchical representation of multiple layers, which functions as a linear "classifier." It has been shown that more units often learn more effectively. Our first approach is adding complexity to a model's depth and width to improve its accuracy in classifying images associated with emotions. We run two variations of CNN models to compare their performance on the dataset. The significant difference between the two models—Model1 and Model2— is various levels of complexity in their depth and width. Model 2, which consists of 830,928 trainable parameters, four convolutional layers, and three fully connected layers, can only achieve approximately 0.54 accuracy in testing data. Nonetheless, model 1, which only has 605,060 trainable parameters, three convolutional layers, and two fully connected layers, outperforms model 2, with evaluated accuracy of around 0.6719. In comparing the model 1 and model 2 performance, we observe that adding more neurons and layers increases a model's complexity but not increases accuracy. We may boldly assume that given the same amount of training dataset, a model may have reached its best capacity even though we have added more trainable parameters to a model unless we apply more dropout layers to regulate a model.

D-2 Transfer Learning Architectures

Our second approach is to explore transfer learning architectures, an artificial neural network that processes and recognizes patterns in large-scale images. In the transfer learning section, our approach is that for VGGNet, ResNet, and EfficientNet, we directly used the convolutional and pooling layers and froze their weights, i.e., no training will perform on them. We removed the already-present fully-connected layers and added

our fully-connected layers for this multi-class classification task.

Problem and Solution Summary

A.Generalization

The continuing improvement of a machine learning model's performance usually involves a model's capability in generalization. While we are busy adding complexity to architecture, we should not ignore the significance of checking the validation accuracy to see whether a model overfits the training dataset. This check is essential because it can help generalize and prevent overfitting problems. This improvement in generalization requires regularizing techniques, such as BatchNormalization, Dropout, and MaxPooling, to achieve better generalization. MaxPooling is a technique used to abstract away locality which facilitates reducing the size of the input data and helping the model generalize better. *Dropout* is a technique that prevents the model from overfitting, which is when the model performs well on the training data but not on new data. Batch normalization is a technique that helps prevent the gradient from vanishing or exploding, which can cause the model not to learn appropriately. Early stopping is a regularization technique; the training is enforced to stop immediately when the performance worsens. This technique helps prevent the model from overfitting the training data as the accuracy gap between the training and validation sets increases.

B. Information Extraction, Filters' size: Information detectors

In convolution layers, we have filters that work as "local detectors" and stride over the entire image to detect underlying information available from the image, such as an edge or a curve and many features. To accurately detect an image, we rely on a filter's size to precisely capture its shape. We speculate that since in our previous transfer learning section, the kernel size is fixed to 3*3, it could be challenging for our transfer learning architectures to accurately extract images like facial recognitions that require more precise detection. Hence, in the following experiment, we propose to decrease the kernel size from 3*3 to 2*2 and maintain the same complexity of training parameters to detect objects and patterns more efficiently.

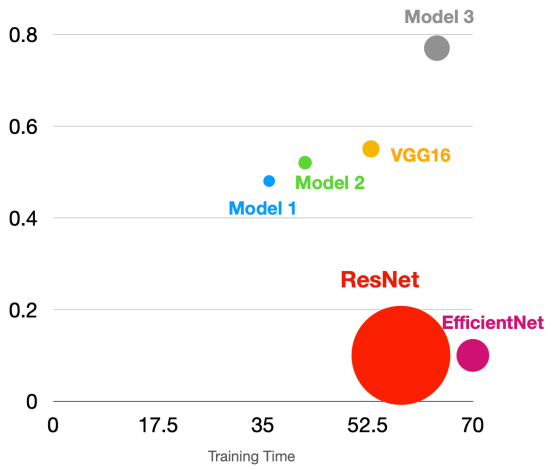
C. Learning Rate Convergence and Epochs

It is important to note that setting a reasonable learning rate is essential in loss optimization while training a model. Setting the right learning rate can achieve the lowest loss. A small learning rate converges slowly and gets stuck in false local minima; significant learning rates tend to overshoot, become unstable and diverge; stable learning rates converge smoothly and avoid local minima.

Recommendation for Implementation

A. Extract Insights and Combined Analysis

We tested Model 3 and achieved an initial accuracy of 0.7422, which was way better than any other transfer learning network accuracy reported before. We recommend three optimization techniques for improving a convolutional neural



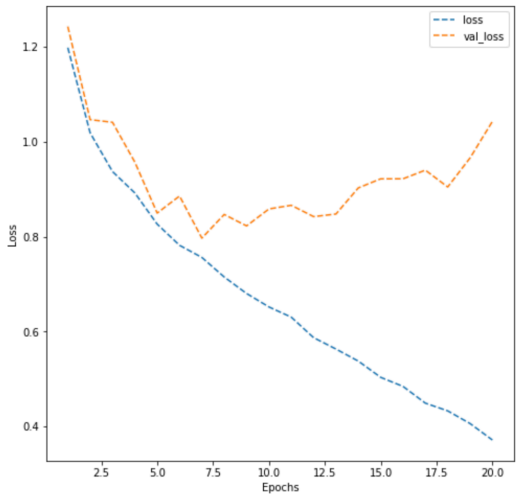
network: a neural network's complexity design, regulation, and hyper-parameter tunings, such as a kernel's size, learning rates, and training epochs.

Building an adequately complex neural network architecture provides a very structured optimizing plan. In Model1 and Model2, their limited depth and breadth have seemingly impeded reaching higher accuracy. However, their intricate layers with millions of parameters hardly fit our training samples in pre-trained models. Our transfer learning architectures did not perform as well as we had expected. Among the three pre-trained models, even though VGGNet outperforms ResNet and EfficientNet by roughly 30% accuracy, its testing accuracy of 52% is relatively low compared to model1's accuracy of 61% and Model2's 54%. ResNet and Efficient generally have not improved throughout 20 epochs with as low as roughly 20% accuracy. In this section, it is highly plausible that these complex models were barely learning. At this point, we speculate that some risks could be related to a model's best capacity. Since pertained models are used to train on a large dataset, our limited training samples may not have fitted these transfer learning architectures, which have been trained on ImageNet previously.

Hence, it is essential to consider a model's best capacity, given that we currently do not have extra resources to collect more training data. Given a limited number of training data, an appropriately complex model with no excessive number of training parameters, therefore, is significant to curb under-fitting problems. As we increase convolutional blocks to model 3, model 3's trainable parameters are nearly as many as those in transfer learning models. However, Model3s' parameters are not frozen so that they can reduce their weight loss during each epoch. For example, our model 3, which has five convolutional blocks and three fully connected layers, performs far better than any transfer learning architectures, achieving a testing accuracy of roughly 75% after 35 epochs. When evaluating a model's success through a confusion matrix, we observed that, in model 3, the under-fitting issue in Model1 and Model2 in "Neutral" and "Happy" seem to have been drastically minimized by 24% and 21%.

Another optimizing plan that a model can utilize to address overfitting is regulation. Due to high complexity, overfitting is a prevalent issue while training a neural network. Regularization can facilitate improving a model's accuracy as its complexity increases. Regularization techniques include Early Stopping, Data Augmentation, Dropout, Batch Normalization, and MaxPooling. During all experimentations, Early Stopping, Data augmentation, and MaxPooling have been applied, whereas Dropout and Batch Normalization is selectively applied. Figure 3 shows that in Model 2, which does not apply dropout layers, the overfitting problem becomes especially obvious after seven epochs. Due to its absence of dropout layers, after 7.5 epochs, the validation loss starts worsening while training loss experiences a continuous decrease, signifying it is overfitting on the training dataset.

Figure 3 Model 2 Training History



To thoroughly improve a model, we have slightly changed Model 3's structure to l the challenges of overfitting. We have vigorously added BatchNormalization and LeakyReLU between convolutional layers and MaxPooling, after which we have added dropout layers. Furthermore, we have added BatchNormalization and dropout layers for each fully-connected layer. As we expected, figure 4 demonstrates that risks

related to overfitting seem to have diminished after reducing the co-dependence of each neuron and enhancing individual neurons' power. Our Model 3 has reached 77% testing accuracy.

The final element in successfully improving a testing accuracy is fine-tuning hyperparameters. It is important to note that setting a reasonable learning rate is essential in loss optimization while training a model. Setting the right learning rate can achieve the lowest loss. A small learning rate converges slowly and gets stuck in false local minima; significant learning rates tend to overshoot, become unstable and diverge; stable learning rates converge smoothly and avoid local minima. When compiling and training model 3, we have increased its epochs and learning rates to fit. This design in backward propagation is a tradeoff between the resources it has and the improvement we aim to achieve. In our model 3 experiment, in terms of the design of a neural network architecture, we have much more trainable parameters than model 1 and model 2 and more epochs to train. However, to speed up the complete cycle of the training process, we increased Adam optimizer's learning rate to 0.003. Table 1.

Table 1 hyperparameter tuning

Methods		Kernel Size	
Model 3	Testing Accuracy	3*3	2*2
Adam (Learning rate)	0.001	0.77	0.773
	0.003	0.789	0.75

B. Suitable Measures

Aside from plotting accuracy for both training and validation dataset, the measures of success in image classification depends on the rate for misclassification in test data. Here, we will introduce confusion matrix. Under this hood, f1-score will determine which model works better in terms of misclassification problem. Our model3, in which we have 5 convolutional blocks and 3 fully connected layers and we have applied multiple regularization techniques, performs far better than transfer learning architecture, achieving testing accuracy of roughly 75% after 35 epochs. In terms of evaluation a model's success through a confusion matrix, we have observed in figure 5 that, in model3, the issue in model 1's misclassification in "Neutral" and "Sad" seem to have been drastically minimized by 24% and 21%. Figure 6 also shows the final result for confusion matrix, reaching 88% f1 score for "Happy", 59 % for "Sad", 72 %for "Neutral", and 91 % for "Surprise".

Figure 4 Model 3 Training History

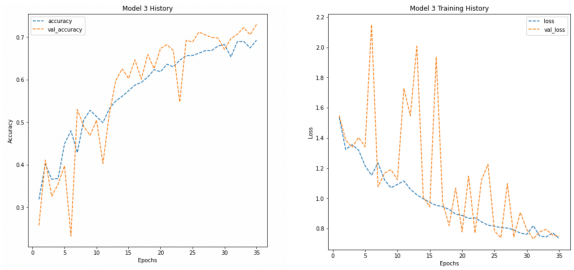


Figure 5 Performances in Emotion Classification

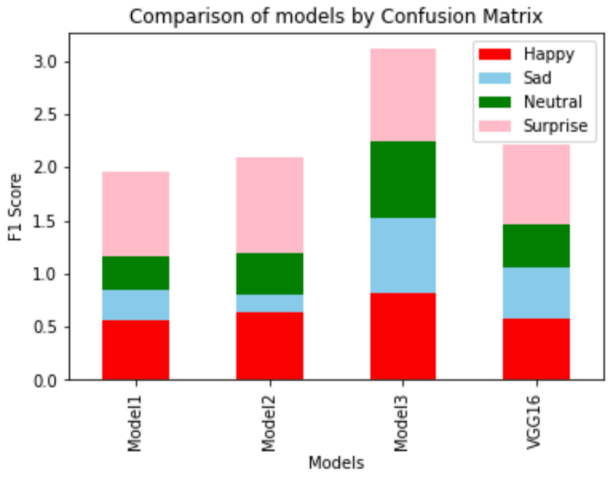
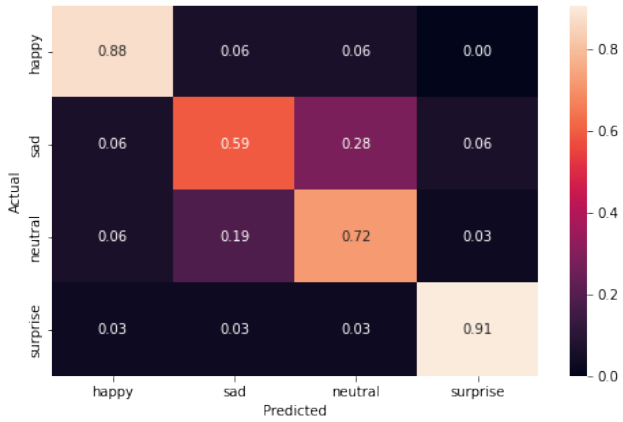


Figure 6 The confusion matrix of our final model 3 test set.



Appendix

Early stopping

is a technique used to prevent overfitting in a model. It works by splitting the training data into two parts: a training set and a validation set. During the training process, the model's performance is monitored on the validation set. If the performance starts to worsen, the training is stopped immediately. This helps to prevent the model from overfitting on the training data, as the accuracy gap between the training and validation sets starts to increase.

Data augmentation

is a technique used to increase the amount of data available for training a model. It works by taking existing data and making slight modifications to it, such as rotating or flipping an image, to create new data points. This helps to reduce the overfitting problem, which is when a model performs well on the training data but not on new data.

Dropout

is a regularization technique used to prevent overfitting of the training data. It works by randomly dropping a few features and neurons in the hidden layer during the training phase. This helps to reduce the co-dependency between neurons, which can otherwise lead to overfitting.

Batch normalization

is a technique used to improve the performance and stability of neural networks. It works by normalizing the inputs of each layer so that they have a mean output activation of zero and a standard deviation of one. This means that the range of input distribution of each layer stays the same, no matter what changes occur in the previous layer.