Emotion Detection Model using Convolutional Neural Network

Shangit Cruze (2102420)

GitHub: github.com/sangitcruze

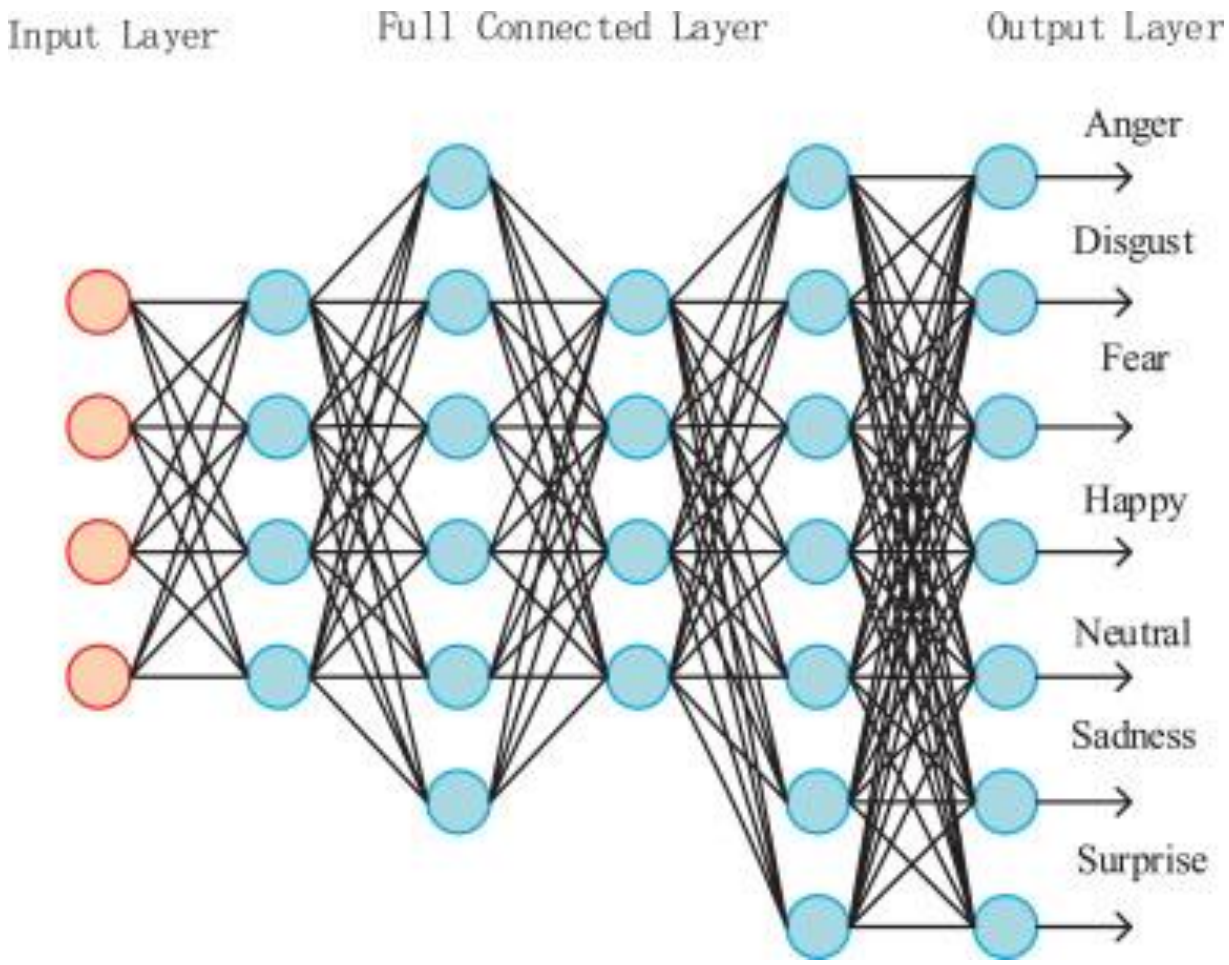


Table of Contents

Introduction and Background

I have chosen to implement an emotion detection model using keras and convolutional neural networks & OpenCV to test the trained model. The purpose of this model is to recognize various emotions by analyzing patterns based on the person's facial textures and shape.

This is accomplished by converting characteristic details about a person's face, such as distance between the eyes or chin shape, into a mathematical representation and comparing it to data on other faces collected in a face recognition database.

Because there are multiple label classes, the application additionally employs the Adam algorithm optimizer and a categorical cross-entropy function. Using this strategy, the model will seek a set of weights with the smallest difference between the model's prediction and the probabilities in the training dataset.

In this project, a convolutional neural network (CNN) is built and trained in Keras to recognise face expressions. The dataset collection comprises of grayscale photos of faces 48x48 pixels in size. Depending on the emotion displayed, each face is allocated to one of seven groups. (0 = angry, 1 = disgust, 2 = fear, 3 = happy, 4 = sad, 5 = surprised, 6 = neutral). OpenCV is used to recognize faces in images and generate bounding boxes around them automatically. Once trained, the CNN will use video and image data to recognize facial expressions in real time.

I have used multiple online tutorials to implement the project.

Aim and Purpose

We humans can swiftly detect other people's emotions. Because it is employed in real-time applications, automatic emotion recognition on a human face is critical. Because of recent developments in GPU technology, many applications, such as face recognition, handwritten digit identification, and object recognition, have evolved. This application aims to accurately detect and label various emotions based on facial patterns.

Data Specification

Dataset:

This dataset collection comprises of 48x48 pixel grayscale photos of faces. The faces have been automatically registered so that each face roughly fills the same amount of space in each image and is roughly centered. Each facial expression falls into one of seven categories: angry, disgusted, afraid, happy, sad, surprised, or neutral.

The dataset (“Face expression recognition dataset”) was prepared by Jonathan Oheix via Kaggle.

(source: Face expression recognition dataset | Kaggle)

Methodology

Using Keras, we design and train the CNN model. We utilize OpenCV's face detection classifier to create bounding boxes around the faces that are automatically discovered. The network is trained and saved following the creation of the facial model. We run the main Python script after training the Emotion Recognition model, which loads the trained model and saves weights before finally applying the model to a real-time video stream captured by a camera.

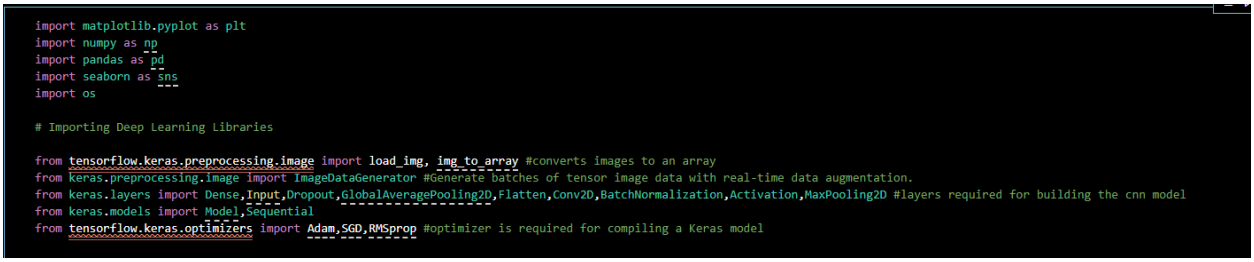
Pseudocode:

1. Prepare the data: we start collecting and preparing the training data. In this case, we use the pre-existing dataset of images labeled with different emotions.

2. Preprocess the data: we need to preprocess the images to get them ready for training. which include resizing the images to a consistent size, normalizing the pixel values, and splitting the dataset into training and validation sets.

3. Building the model: we start by importing the necessary libraries and creating the Keras model. We include a few Conv2D layers with max pooling, followed by some fully connected layers in the CNN model.

Importing the libraries:



```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import os

# Importing Deep Learning Libraries

from tensorflow.keras.preprocessing.image import load_img, img_to_array #converts images to an array
from keras.preprocessing.image import ImageDataGenerator #Generate batches of tensor image data with real-time data augmentation.
from keras.layers import Dense,Input,Dropout,GlobalAveragePooling2D,Flatten,Conv2D,BatchNormalization,Activation,MaxPooling2D #layers required for building the cnn model
from keras.models import Model,Sequential
from tensorflow.keras.optimizers import Adam,SGD,RMSprop #optimizer is required for compiling a Keras model
```

Keras:

Keras provides a high-level API for building and training deep learning models, including convolutional neural networks (CNNs) for image recognition tasks.

NumPy: NumPy is designed for scientific computing and supports arrays, matrices, and a variety of mathematical operations on these arrays. An N-dimensional array object (ndarray) is provided by NumPy and can be used to represent arrays and matrices of various shapes and sizes. Compared to Python's built-in data structures, the ndarray is a highly optimised array that is quicker and more memory-efficient. The input data from the dataset, which is commonly supplied as multidimensional arrays, is handled and modified using NumPy.

Pandas: Pandas is a Python library for manipulating and analysing data. It offers resources for working with time series and other types of organized data, such tables. Pandas, which is based on NumPy, is frequently used in conjunction with other libraries, including Scikit-learn for machine learning and Matplotlib for data visualisation.

Seaborn: Seaborn, which is based on Matplotlib, offers a high-level interface for producing useful and appealing statistical visuals. Since Seaborn is based on Matplotlib and works well with Pandas data structures, it is a highly liked option for Python data visualisation.

Metrics like accuracy and loss can be used to track the model's performance while it is being trained. To better understand how the model is functioning over time, these metrics can be visualised using Seaborn's charting tools.

After the model has been trained, Seaborn can be used to display the model's predictions on fresh data or to show how the emotions in the dataset are distributed. This might be helpful for finding trends in the data and comprehending how the model performs in practical situations.

Model architecture: we set the number of classes as 7 since there are 7 different possible outcomes and we are using the Sequential model since that fits the requirements of what we are trying to achieve.

A layer is a key building block in a Convolutional Neural Network (CNN) that analyses input data and generates output data. In a CNN, there are several kinds of layers, each with a unique purpose and set of requirements.

Here are the layers that we are using:

Convolutional layer: Using a group of trainable filters, this layer conducts convolutional operations on the input data. Each filter creates a feature map by sliding over the input data and highlighting specific patterns or characteristics. The hyperparameters of the layer control the size, quantity, and stride of the filters.

Pooling layer: By performing a pooling operation, such as max pooling or average pooling, over relatively tiny areas of the feature map, this layer down-samples the output of the convolutional layer. This keeps the most crucial elements while reducing the feature map's spatial dimensions.

Activation layer: The output of the preceding layer is applied to by the activation layer together with a nonlinear activation function, in our case we are using relu. The network gains nonlinearity as a result, allowing it to perceive complex connections and patterns in the incoming data.

Batch normalisation layer: This layer normalises the output of the layer before by subtracting the mean of the activations and dividing by their standard deviation. As a result, training stability and speed are improved, and internal covariate shift is decreased.

Dropout layer: During training, this layer at random removes a portion of the activations from the preceding layer, which aids in lowering overfitting and enhancing generalisation performance.

Flatten layer: The output of the preceding layer is flattened in this layer into a 1D vector so that it can be fed into a fully connected layer. This is frequently used in networks to change from convolutional layers to fully linked layers.

Fully connected layer: Every neuron in the layer above it is connected to every neuron in the layer below it, and this layer applies a linear transformation followed by an activation function. For classification or regression tasks, this layer is utilised at the network's very end.

These layers are stacked together to form a deep neural network that can learn complex representations of the input data. The architecture and hyperparameters of the layers are customized based on the task and dataset.

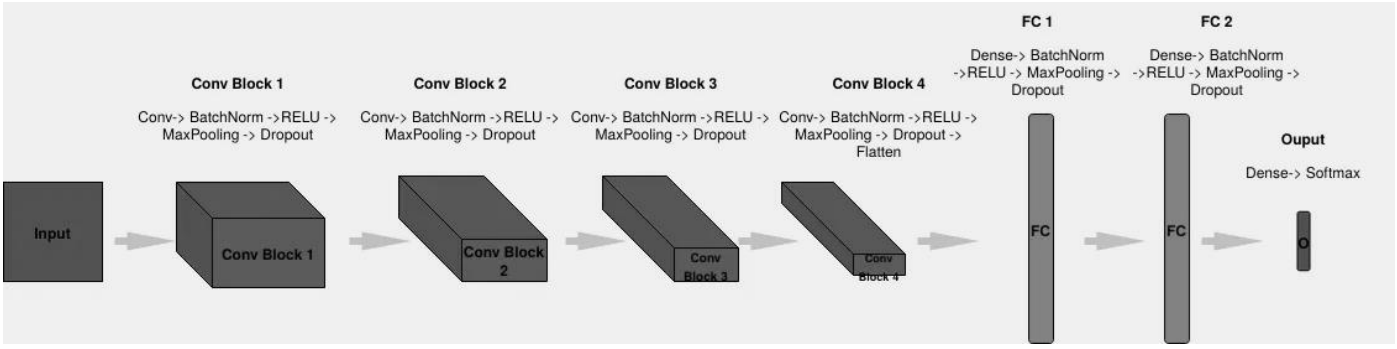
```
no_of_classes = 7

model = Sequential()

#1st CNN layer
model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.25))
```

The 4 convolutional layers are followed by max pooling layers and a few fully connected layers for classification in the model's architecture. Here we use the “MaxPooling2D” layer to down-samples the output of the convolutional layer. The “Dropout” layer helps lower overfitting and enhance generalisation performance.

The output label for each of the seven emotions is predicted using the dense layer with reLU Activation. Model training is sped up per epoch by using the Adam optimizer with a learning rate of 0.001. We are using “Activation” as ReLU because ReLU uses straightforward element-wise operations, which makes it computationally more efficient than sigmoid and tanh and due to the fact that ReLU introduces sparsity into the network, as it sets negative values to zero. By doing so, the network's generalisation performance can be enhanced, and the risk of overfitting can be decreased. We also add the Flatten layer to transition from the convolutional layers to the fully connected layers in the network.



Training and Validation

Data from the training and test folders are automatically fed into Keras to build mini-batches for training and validation (test). The data loader has some established hyper-parameters. The batch size is 128 and the image size is 48x48. The batch size is the number that depicts the amount of training examples the model should take per iteration.

```
epochs = 25
model.compile(loss='categorical_crossentropy',
              optimizer = Adam(lr=0.001),
              metrics=['accuracy'])
```

First, 25 epochs are chosen as the number of epochs. A gradient descent hyperparameter that regulates the quantity of full iterations across the training dataset is the number of epochs. A hyperparameter that may be changed to increase the model's precision is the number of epochs. The model often learns more about the patterns and correlations in the data the more epochs it is trained on, up to a point where overfitting may happen. The number of images in the training generator is divided by the training generator's batch size on the floor to get the number of steps per epoch. For the validation set, this is done once more. The training is then supplemented by three callbacks. A callback is a type of object that can execute commands at different points during training, such as the beginning or conclusion of an epoch or a single batch. The first callback, ReduceLROnPlateau, lowers the learning rate when the validation loss does not improve after two epochs.

```
metric = 'val_accuracy'
checkpoint = ModelCheckpoint("./modelTrained.h5", monitor= metric, verbose=1, save_best_only=True, mode='max')
```

The ModelCheckpoint callback saves model weights with greater validation accuracy. The weights of the models are recorded in HDF5 format. This is a grid format for storing multi-dimensional numerical arrays.

Trained Model:

The main.py script is executed to build the app and serve the Model's predictions through a interface. The camera class transmits the picture stream to the pre-trained CNN model, receives the model's predictions, adds labels to the video frames, and ultimately returns the image to the interface. Both recorded videos and live footage captured by a camera can be used with the model. This is achieved by using a cascade classifier. We use OpenCV to apply the cascade classifier to detect faces in an image/video.

```
face_classifier = cv2.CascadeClassifier(r'C:\Users\x77wi\Desktop\Emotion_Detection_CNN-
main\haarcascade_frontalface_default.xml') #CascadeClassifier class to detect objects in a video stream.
classifier =load_model(r'C:\Users\x77wi\Desktop\Emotion_Detection_CNN-main\model2.h5') #Loading the trained model
```

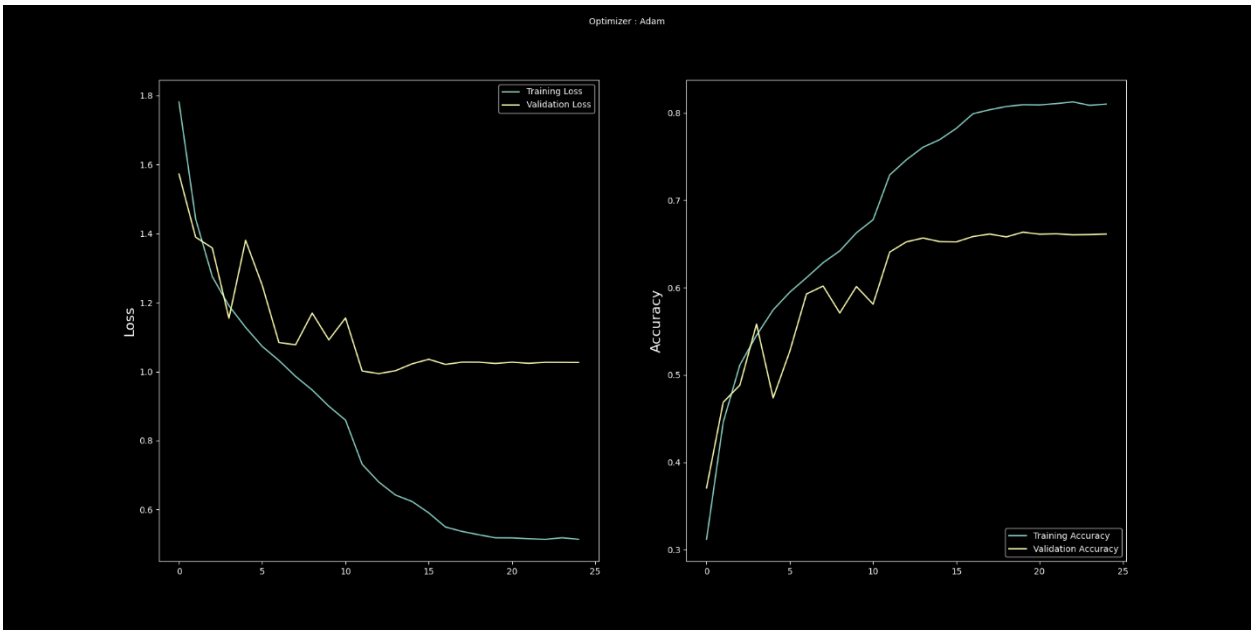
```
emotion_labels = ['Angry','Disgust','Fear','Happy','Neutral', 'Sad', 'Surprise'] #labeling all the emotions


cap = cv2.VideoCapture(0) #Capture video via webcam
```

Performance:

The training and evaluation of the model was done on this computer. Here are the following specification AMD Ryzen 7 5800x Processor 8-Core Processor, 3801 Mhz, 8 Core(s), 16 Logical Processor(s). 8000 Megabytes of RAM and a RTX 3070TI GPU.

Results and Conclusions



We can see the model training progress in each of the epochs. On the training and validation data, we observe an increase in accuracy over time. We can examine the model's past as well. It will demonstrate the loss at each epoch and how it decreases to cause the model parameters to converge.

If we look more closely at the image above, we can see how the training loss continues to decrease as the validation error decreases up to a certain point before beginning to rise. The model may have been overfitted, according to this observation.

```
Epoch 25/25

250/250 [==============================] - ETA: 0s - loss: 0.5140 - accuracy: 0.8100
```

Epoch 25: val_accuracy did not improve from 0.66335

The accuracy for the training data (0.81) is greater than that for the validation data (0.66), which is less accurate. Even in the aforementioned image, we can see that the validation loss rises with the number of epochs. The model may not have performed well on the validation data, according to this as well. As a result, we shall proceed and change the parameters.

First, we increase the value of all the Dropout layers from 10% to 40% and the batch size to 16. We also change the "patience" parameter of the "ReduceLROnPlateau" from 3 instances to only 2 so that the algorithm reduces the learning rate more often, this can speed up convergence and improve performance by enabling the model to make fewer steps towards the ideal answer.

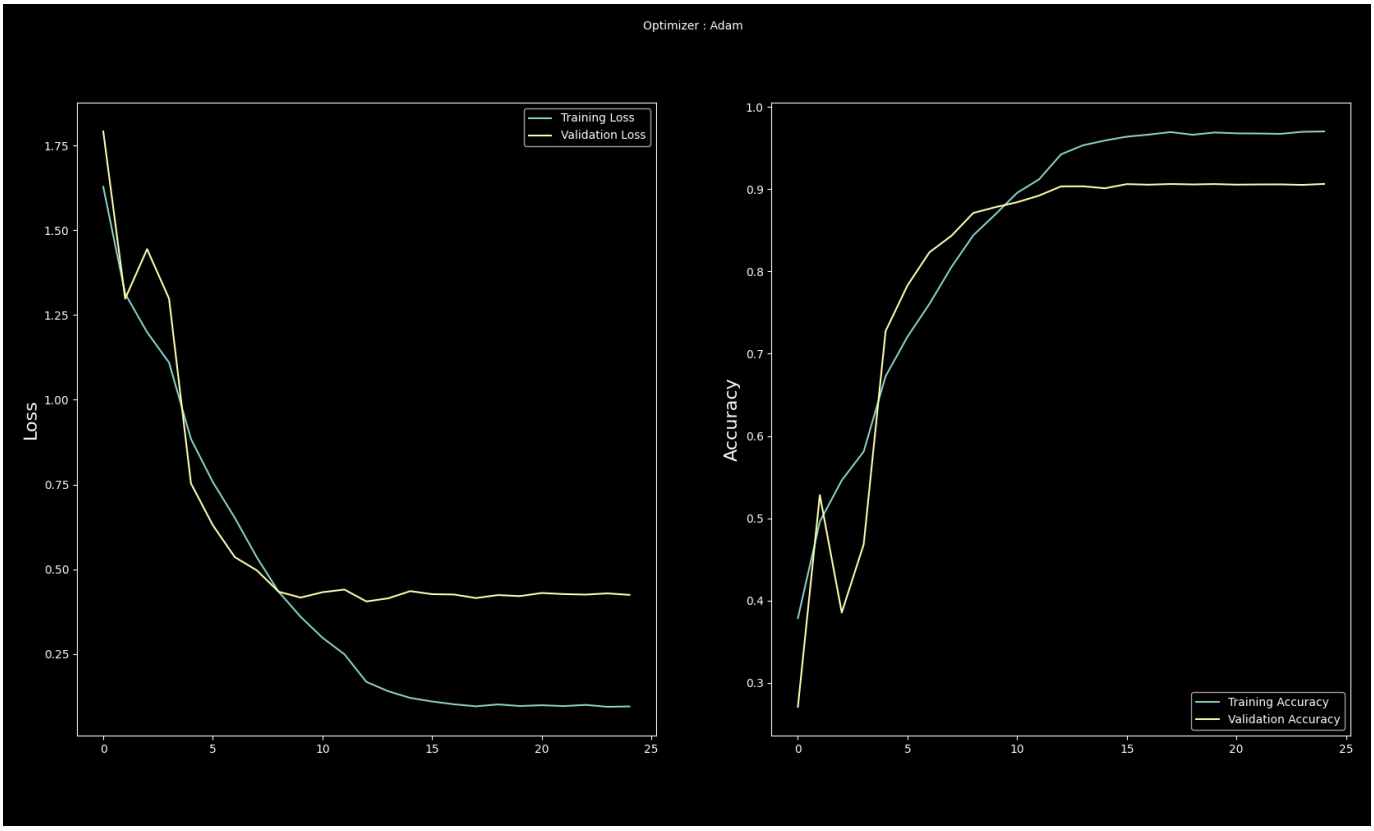
```
#1st CNN layer
model.add(Conv2D(64,(3,3),padding = 'same',input_shape = (48,48,1)))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size = (2,2)))
model.add(Dropout(0.40))
```

We also shuffle our validation data order unlike before.

```
test_set = datagen_val.flow_from_directory(folder_path+"validation",
                                           target_size = (picture_size,picture_size),
                                           color_mode = "grayscale",
                                           batch_size=batch_size,
                                           class_mode='categorical',
                                           shuffle=True)
```

Improved Results:

We now train the model once again and here are the results:



We can draw the conclusion that our overfitting issue has been fixed because both validation and training losses are decreasing a lot more than previously.

Epoch 25: ReduceLROnPlateau reducing learning rate to 2.5600002118153498e-09.

1800/1800 [==============================] - 203s 113ms/step - loss: 0.0950 - accuracy: 0.9703 - val_loss: 0.4240 - val_accuracy: 0.9064 - lr: 1.2800e-08

We now have a new training accuracy of 97% and validation accuracy of 90%. This is because of various factors such as the increase of dropout rates in all the layers and the increased batch size.

Increasing the dropout rate aided in improving the results because dropout regularization randomly removes neurons from the neural network during each iteration's training along with their connections. It's the same as training different neural networks when we remove different sets of neurons. Therefore, the dropout process is similar to averaging the results of numerous distinct networks. Dropout will ultimately reduce overfitting since different networks will be overfit in different ways.

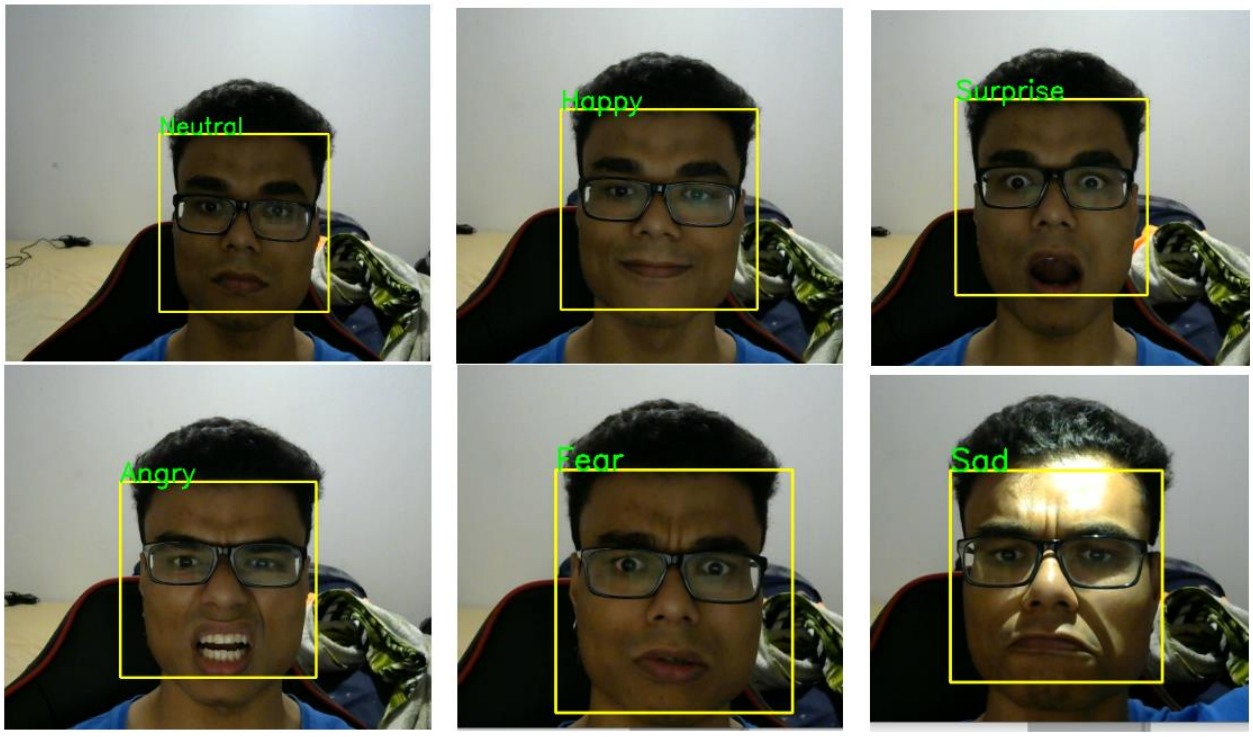
Shuffling our validation data order has also aided in improving the results because the network receives the training data in batches during the training process. Over all repetitions, this frequently occurs in a predetermined order. The network may become biased as a result of this.

Decreasing the batch size may have helped since smaller batch sizes can act as a regulator to keep overfitting at bay. This is due to the noise generated by the small batch size can aid the model's generalization.

First Training	Training	Validation
Accuracy	0.8100	0.6612
Loss	0.5140	1.0266

Final Training	Training	Validation
Accuracy	0.9703(19.79% increase)	0.9064(37.09% increase)
Loss	0.0950(81.52% decrease)	0.4240(58.70% decrease)

Final results:



Final Trained model classifying emotions in low lighting.

As we can see from the above image, the trained AI model is able to classify different emotions correctly.

Ethical concerns:

Privacy: Citizens' main issue with facial recognition is the absence of user permission involved in the deployment process. Many governments throughout the world are already using CCTV surveillance systems. In most cases, user consent is not sought when collecting citizens' facial data in public locations. This permits automated live monitoring of people. Governments may track residents' every step, jeopardizing their privacy. When utilized recklessly, every person can become a walking ID card, raising privacy, moral, and security concerns, this is especially dangerous in situations where people do not wish to be identified or monitored, such as in public places or during protests.

Identity fraud: Because sensitive data about persons may be retained and potentially accessed by hackers or other criminal actors, facial recognition technology can also offer security problems. If facial data is compromised, both governments and average citizens are at serious risk. If facial recognition security measures are not rigorous enough, hackers can easily assume the identities of other individuals to commit illegal activities. Theft or duplication of the data necessary for financial transactions could cause enormous financial losses.

Bias and discrimination: Certain groups of individuals, such as individuals with darker skin tones or women, may be favored by facial recognition technology. This can result in prejudice and injury to these groups, as well as the reinforcement of pre-existing biases and preconceptions.

Technical impairments:

During the process of training the data I have encountered various technical impairments which slowed down my progress. For example, the batch size of the module was initially set to 128 which was done to reduce the impact of any noisy or outliner data points. However, large batch size utilizes more memory and is less stable during training. This caused my computer to freeze up and crash multiple times, so I had to find an ideal batch size which uses the optimal amount of computer memory (RAM). Here are the numbers I have analyzed:

Batch Size	Ram usage (Average)	Total Ram usage (Average)
16	~1.452 Megabytes	~6.952/7.900 MB Available
32	~1.652 Megabytes	~7.152/7.900 MB Available
64	~2.052 Megabytes	~7.465/7.900 MB Available
128	~2.452 Megabytes	~7.850/7.900 MB Available

In training, our model will complete each epoch more quickly the higher the batch size. This is so that our system can process much more than one sample at a time, depending on our computational capabilities.

The trade-off is that even while our machine can handle very big batches, the model's quality could deteriorate as we increase the batch size, which might ultimately prevent the model from generalizing successfully to data that it hasn't seen before.

From this analysis I have found the size 32 to be the most ideal since I did not experience any major performance issues with it.

Another technical issue present was that the trained model would struggle to recognize facial emotions in poor lighting conditions and especially in individuals with darker skin such as myself. This is due to the fact that inadequate lighting can result in low contrast and shadow areas in the image, making it harder for the model to discern between different face characteristics especially in individuals with darker skin colour.

One option to overcome this issue is to acquire a more diversified dataset that includes people with a wide range of skin tones. This can help to ensure that the model is trained on a representative sample of the population and can recognize emotions on people of all skin tones.

Another strategy is to employ data augmentation techniques to replicate a wide range of skin tones during training. This can include applying colour transformations or other approaches to the training photos to change the skin tone of the persons in the images. This can help the model learn to recognize emotions on people with a variety of skin tones.

Conclusion:

In conclusion, the application achieves its core goal of accurately recognizing distinct moods based on facial patterns. Despite some restrictions, I believe I have done a fair job at training, analyzing, and understanding the general notion of machine learning using CNN. While working on this project, I became aware of the power of machine learning, particularly CNNs. An automated emotion detection approach, for example, means that emotion analysis does not require human interaction. This is useful when human analysis is impractical or inefficient, such as in large-scale research or surveillance systems. CNNs can also detect emotions without requiring physical contact with the person being analyzed.

In the future I plan on expanding my model to adapt some of these concepts and overcoming its limitations.

References

Aslam, N. (2022) *Training CNN from scratch using the custom dataset*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2022/07/training-cnn-from-scratch-using-the-custom-dataset/ (Accessed: 11 May 2023).

Badrulhisham1, N.A.S. and Mangshor1, N.N.A. (2021) *IOPscience*, *Journal of Physics: Conference Series*. Available at: https://iopscience.iop.org/article/10.1088/1742-6596/1962/1/012040/meta (Accessed: 11 May 2023).

Brownlee, J. (2020) *How to develop a CNN from scratch for CIFAR-10 photo classification*, *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification/ (Accessed: 10 May 2023).

Convolutional Neural Network (CNN) : Tensorflow Core (no date) *TensorFlow*. Available at: https://www.tensorflow.org/tutorials/images/cnn (Accessed: 13 May 2023).

Madan, A. (2021) *Emotion detection using convolutional neural networks and OpenCV | Keras | Realtime*, *YouTube*. Available at: https://www.youtube.com/watch?v=Bb4Wvl57LIk (Accessed: 12 May 2023).

code implementation

Nouman (2021) *Writing cnns from scratch in Pytorch*, *Paperspace Blog*. Available at: https://blog.paperspace.com/writing-cnns-from-scratch-in-pytorch/ (Accessed: 13 May 2023).

Shabrina, N.H. *et al.* (no date) *Emotion recognition using convolutional neural network in virtual meeting environment*, *Ultima Computing : Jurnal Sistem Komputer*. Available at: https://ejournals.umn.ac.id/index.php/SK/article/view/2108 (Accessed: 11 May 2023).

TheassemblyAe (2021) *Detect emotions with convolutional neural networks*, *YouTube*. Available at: https://www.youtube.com/watch?v=ctjkZnQF_FY&list=PLwZ_jwUrTXNiNmqj3ybkTT6vtF0pehMSE (Accessed: 10 May 2023).

Thetechwriters (2022) *Facial emotion detection using CNN*, *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/11/facial-emotion-detection-using-cnn/ (Accessed: 11 May 2023).

Vaidya, K. (2023) *Emotion recognition*, *Medium*. Available at: https://towardsdatascience.com/emotion-recognition-4fba48dabb6e (Accessed: 11 May 2023).

Oheix, J. (2019) *Face expression recognition dataset*, *Kaggle*. Available at: https://www.kaggle.com/datasets/jonathanoheix/face-expression-recognition-dataset (Accessed: 10 May 2023).