Emotion Detection from Images using Machine Learning

Abstract

Convolutional Neural Networks (CNNs) haves revolutionized image classification by effectively learning and extracting features from images. In this experiment, proposing a novel CNN architecture with multiple convolutional layers, pooling layers, and fully connected layers. The approach incorporates ReLU activation functions and dropout regularization to enhance performance and mitigate overfitting. Extensive experiments on CIFAR-10 and ImageNet datasets demonstrate the superiority of our CNN in terms of classification accuracy. The learned hierarchical representations capture discriminative features, contributing to improved performance.

Task Description

Building a machine learning model capable of detecting the emotions expressed by a person in an image using Python. Emotion detection from images is an exciting and challenging problem in the field of computer vision and artificial intelligence. The goal is to leverage the power of machine learning algorithms to accurately predict the emotional state of individuals.

Dataset

The data consists of 48x48 pixel grayscale images of faces. The faces have been automatically registered so that the face is centered and occupies about the same amount of space in each image.

The task is to categorize each face based on the emotion shown in the facial expression into one of seven categories (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral). The training set consists of 28,709 examples and the public test set consists of 3,589 examples.

Table of Contents

Imports

```
import pandas as pd
import math
import numpy as np
import time
import matplotlib.pyplot as plt
from skimage.io import imread, imshow
import tensorflow as tf
from tensorflow.keras.models import Model
from sklearn.model_selection import train_test_split
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.metrics import f1_score, precision_score, recall_score, confusion_matrix ,classification_report
from tensorflow.keras.utils import to_categorical
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import Flatten, Dense ,GlobalAveragePooling2D
from tensorflow.keras.models import Sequential
```

EDA

Started with Exploratory Data Analysis to check what kind of data we will be dealing with and how to prepare the data for the model.

Dataset Information:

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 35887 entries, 0 to 35886
Data columns (total 3 columns):
 #   Column   Non-Null Count  Dtype 
---  ------   --------------  ----- 
 0   emotion  35887 non-null  int64 
 1   pixels   35887 non-null  object
 2   Usage    35887 non-null  object
dtypes: int64(1), object(2)
memory usage: 841.2+ KB
```

The dataset consists of 35887 instances which represent grayscale images of faces with the emotions associated with those images.

At any given instance in the dataset , the size of image is **48 x 48**.

The dataset consists of 7 emotions (0=Angry, 1=Disgust, 2=Fear, 3=Happy, 4=Sad, 5=Surprise, 6=Neutral)

```
1 df['emotion'].unique()
```

```
array([0, 2, 4, 6, 3, 5, 1])
```

Checking for Data Distribution :

From this analysis we can notice that the samples for happy emotion are in excess and for disgust emotion the samples are very limited which Indicates a class imbalance.

```
1 df['emotion'].value_counts().max()
```

```
8989
```

```
1 df['emotion'].value_counts().min()
```

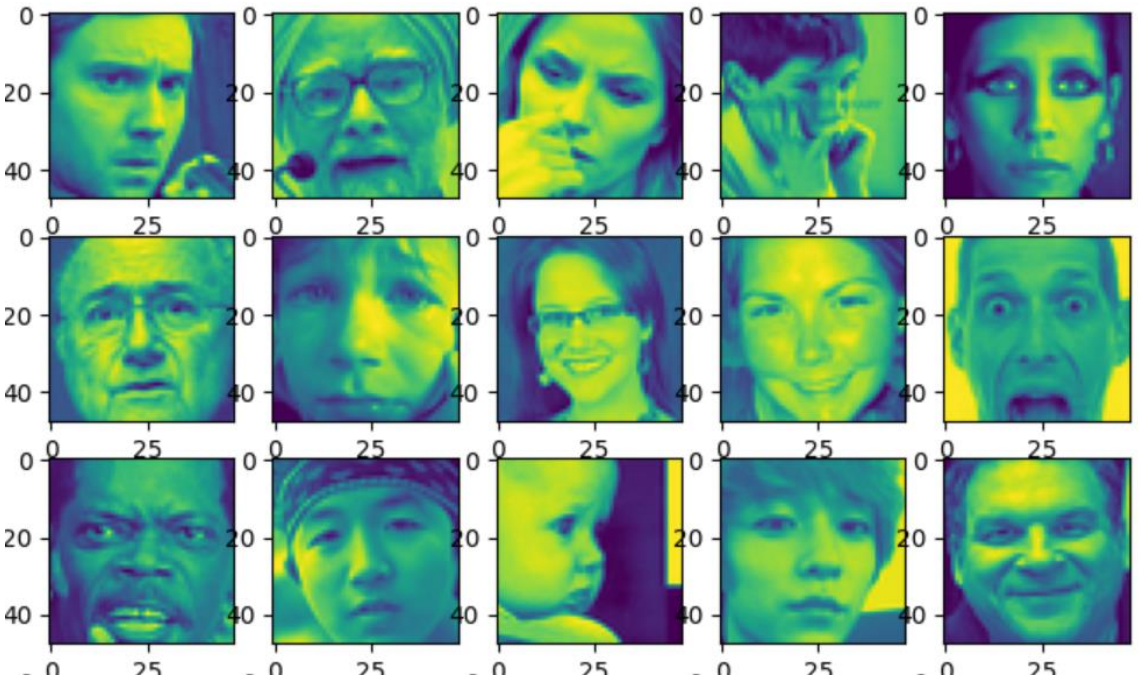
```
547
```

Checking for any null values to ensure proper learning for model:

```
1 df['emotion'].isnull().sum()
```

```
0
```

Sample Data:



Methods & Terminology

1. Handling Class Imbalance

compute_class_weight() : This function from scikit-learn is used to compute class weights for imbalanced classification problems. It calculates the weights inversely proportional to the class frequencies, allowing the model to give more importance to the minority classes during training.

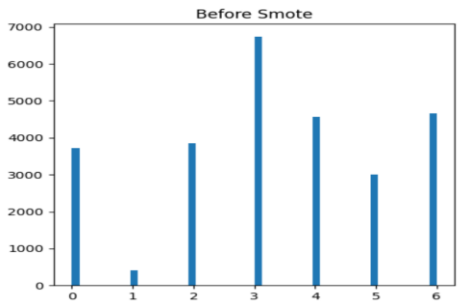
The 'balanced' strategy adjusts the weights such that they are inversely proportional to the class frequencies in the input data.

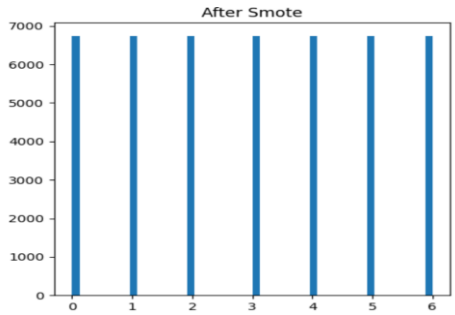
SMOTE[Not Implemented due to poor performance after trials] (Synthetic Minority Over-sampling Technique) is a popular algorithm used for oversampling the minority class in imbalanced datasets. It generates synthetic samples by interpolating between existing minority class samples.

SMOTE works by creating synthetic samples in the feature space of the minority class.

It selects a minority class instance and finds its k nearest neighbors.

It then synthesizes new instances by interpolating between the selected instance and its nearest neighbors.





2. Data Augmentation

tf.keras.preprocessing.image.ImageDataGenerator()

Efficiently loading, augmenting and preprocessing image data during the training of deep learning models.

shear_range: Shear transformation tilts the image along a particular axis, deforming it in a linear fashion.

zoom_range : Magnifies the Image.

width_shift_range : The width shift range parameter controls the range of horizontal (left or right) translations applied to the images.

height_shift_range: Like the width shift range, the height shift range parameter controls the range of vertical (up or down) translations applied to the images,

rotation_range : A rotation range of 30 means the image can be randomly rotated between -30 and +30 degrees.

3. Sequential

Sequential model is a linear stack of layers used for building deep learning models. It allows you to create a neural network by stacking multiple layers sequentially, where the output of one layer serves as the input to the next layer.

BatchNormalization()

Normalizes the activations of a neural network layer by subtracting the batch mean and dividing by the batch standard deviation. By normalizing the activations, Batch Normalization reduces the dependence of the model on the initialization and learning rate, leading to faster convergence.

Conv2D()

The convolution operation involves sliding a small filter (also known as a kernel or window) over the input data. At each position of the sliding window, an element-wise multiplication is performed between the filter and the corresponding input data patch**.** The results of these multiplications are summed up, producing a single value that represents the output of the convolution operation at that position.

kernel_size defines the size of the filters.

Padding adds extra border pixels to the input data, which can help preserve spatial dimensions and reduce information loss at the borders.

Stride defines the step size of the sliding window during the convolution operation. A stride of 1 moves the window one pixel at a time, while a larger stride value skips pixels.

Activation function to introduce non-linearity and increase the model's expressive power. This can be specified using the **activation** parameter of the **Conv2D** layer, such as 'relu' for the rectified linear unit (ReLU) activation.

MaxPooling2D() a downsampling operation commonly used in convolutional neural networks (CNNs) to reduce the spatial dimensions of feature maps while retaining the most prominent features. For each pooling window, MaxPooling2D outputs the maximum value within that region while discarding other values.

4. Optimizers

The **Adam optimizer** adapts the learning rate for each parameter during training. It incorporates the concept of momentum, which helps accelerate convergence and navigate narrow ravines or flat regions of the loss function. The Adam optimizer maintains two moving average estimates: the first moment (mean) and the second moment (variance) of the gradients. The first moment captures the average direction of the gradients, while the second moment captures the scaling of the gradients.

The **SGD** is an iterative algorithm that gradually adjusts the model's parameters to minimize the loss function. By repeatedly updating the parameters using mini-batches, SGD approximates the gradient of the entire dataset, making it computationally efficient.

5. Imshow()

For grayscale images, the **imshow** function displays the image with a colormap that represents different intensity values as different colors. By default, it uses a colormap that ranges from black (minimum intensity) to white (maximum intensity).

6. Confusion Matrix

	Predicted Negative	Predicted Positive
Actual Negative	TN	FP
Actual Positive	FN	TP

Visualizes the performance of a classification model by summarizing the predictions and actual class labels.

Accuracy measures the overall correctness of the model's predictions and is calculated as (TP + TN) / (TP + TN + FP + FN)

Precision quantifies the proportion of correctly predicted positive instances among all instances predicted as positive. It is calculated as TP / (TP + FP)

Recall measures the proportion of correctly predicted positive instances among all actual positive instances. It is calculated as TP / (TP + FN)

Specificity measures the proportion of correctly predicted negative instances among all actual negative instances. It is calculated as TN / (TN + FP)

F1 score is the harmonic mean of precision and recall, providing a balanced measure of the model's performance. It is calculated as 2 * (precision * recall) / (precision + recall)

Implementation

TRIAL 1: Baseline Model
Goal: To run an initial CNN learning on FER Dataset

Dataset	Facial Expression Recognition(FER)Challenge | Kaggle
Image size	48 x 48 x 1

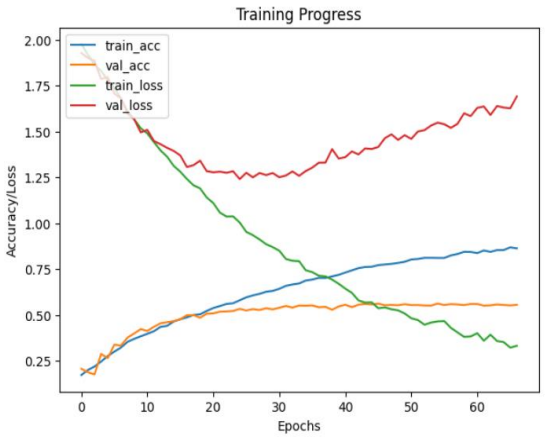
Parameters	1. optimizer = 'adam' 2. Learning rate = 0.001 3. Loss = 'sparse_categorical_crossentropy' 4. Batch size = 64 , Epochs = 100 5. Imbalanced data handled by weight balancer

Insights:

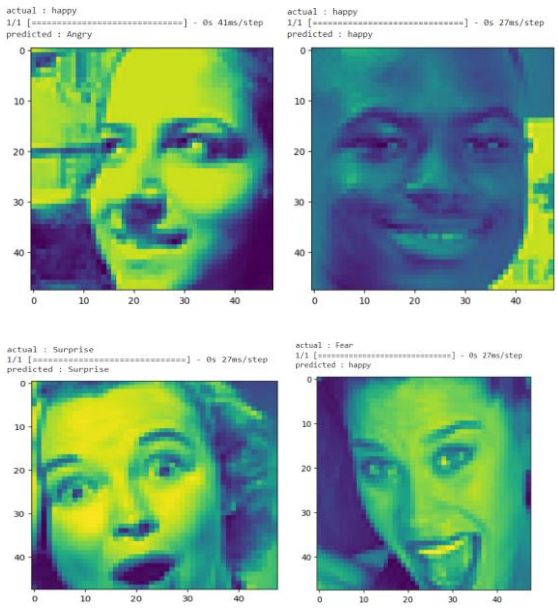
1. Trained a baseline model with 3 convolutional layers with kernel size(3,3) followed by max-pooling layer with activation function set to relu with and callbacks set for early stopping if no improvement after 12 epoch training will be stopped.
2. Handled class imbalance with weights according to the proportion of the samples in the dataset, following are the weights per class used while training : {0: 1.034993270524899, 1: 9.378048780487806, 2: 1.0010413954699298, 3: 0.5703055473153367, 4: 0.8435717419921018, 5: 1.2812395868043986, 6: 0.8272375215146299}
3. The callback was triggered after 65 + epochs .
4. After evaluation with **F1-score** metric the overall accuracy gained was **0.57** on test set and according to Loss and Accuracy curves , it shows although the training loss was converging, and the validation loss started increasing after **25-30** epochs and this eventually leads the model to be **Overfit** and model stops learning and the same is concluded from accuracy curves as well.

5. The reason for this **Overfit** is exclusion of regularization from middle layers of model and **dropout** was only included in last layer.
6. The following visuals are the representation of model parameters, Training graphs, F1 score and some predictions.

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 46, 46, 64)        640

 max_pooling2d (MaxPooling2D  (None, 23, 23, 64)       0
 )

 batch_normalization (BatchN  (None, 23, 23, 64)       92
 ormalization)

 conv2d_1 (Conv2D)           (None, 23, 23, 128)       73856

 max_pooling2d_1 (MaxPooling  (None, 11, 11, 128)      0
 2D)

 batch_normalization_1 (Batc  (None, 11, 11, 128)      44
 hNormalization)

 conv2d_2 (Conv2D)           (None, 11, 11, 256)       295168

 max_pooling2d_2 (MaxPooling  (None, 5, 5, 256)        0
 2D)

 dropout_2 (Dropout)         (None, 5, 5, 256)         0

 flatten_3 (Flatten)         (None, 6400)              0

 dense_5 (Dense)             (None, 64)                409664

 dense_6 (Dense)             (None, 32)                2080

 dense_7 (Dense)             (None, 7)                 231

=================================================================
Total params: 781,775
Trainable params: 781,707
Non-trainable params: 68
_________________________________________________________________
```

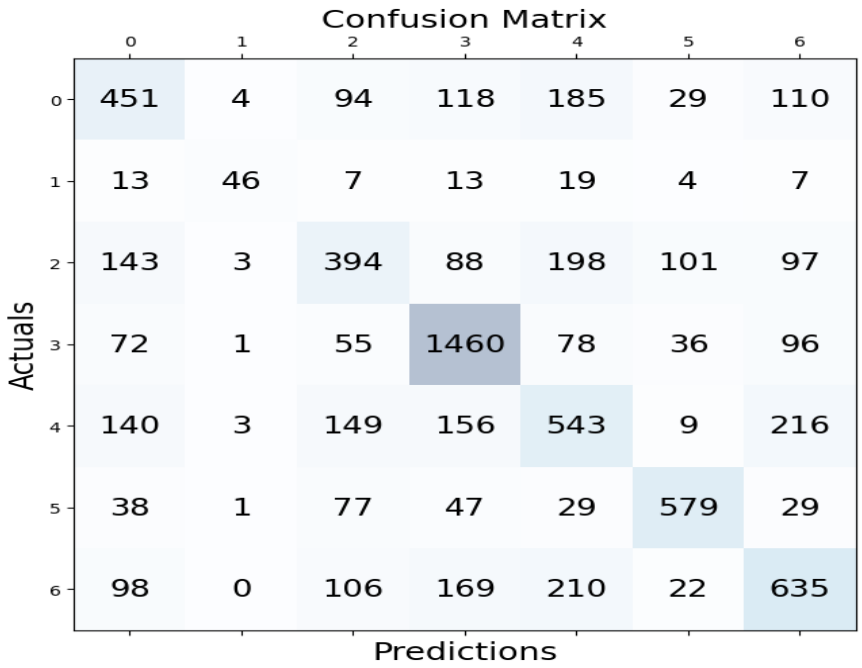


	precision	recall	f1-score	support
0	0.47	0.46	0.46	991
1	0.79	0.42	0.55	109
2	0.45	0.38	0.41	1024
3	0.71	0.81	0.76	1798
4	0.43	0.45	0.44	1216
5	0.74	0.72	0.73	800
6	0.53	0.51	0.52	1240
accuracy			0.57	7178
macro avg	0.59	0.54	0.55	7178
weighted avg	0.57	0.57	0.57	7178



Confusion Matrix:

This shows the predictability of each class, and we can conclude that the class with most samples (happy) has a better performance than other classes and the class with the lowest samples (Disgust) has the lowest performance.



TRIAL 2: Increased Complexity of Model
Goal: To run an CNN learning for better performance than baseline model

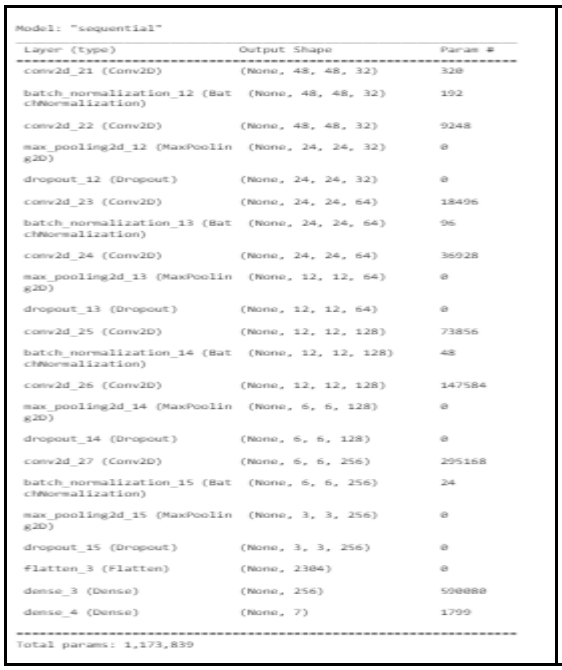
Dataset	Facial Expression Recognition(FER)Challenge | Kaggle
Image size	48 x 48 x 1

Parameters	1. optimizer = 'adam' 2. Learning rate = 0.001 3. Loss = 'categorical_crossentropy'

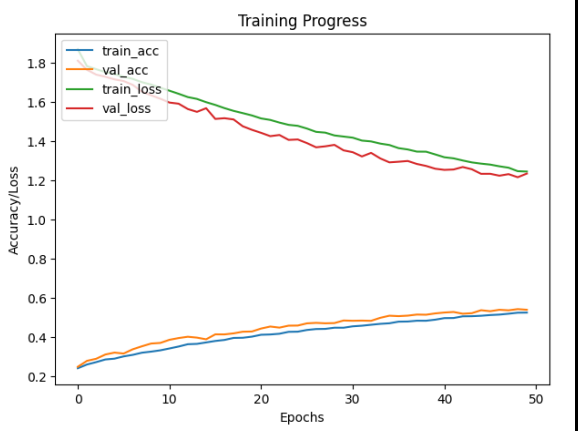
	4. Batch size = 128 , Epochs = 50 5. Data Augmentation

Insights:

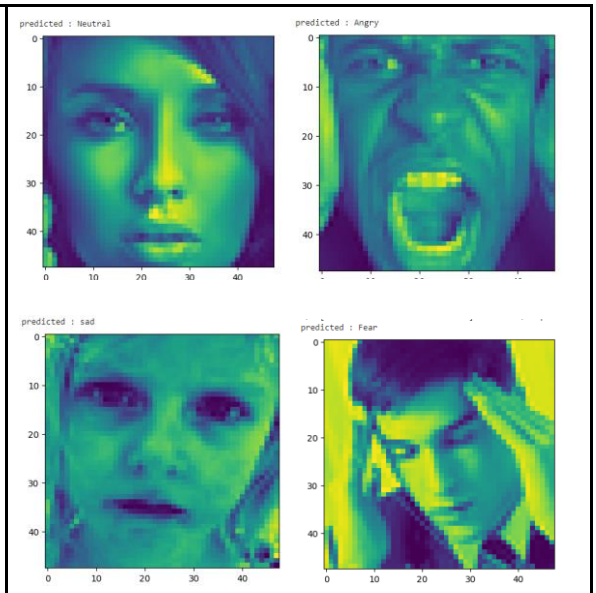
1. Trained a complex model with 7 convolutional layers with kernel size(3,3) followed by max-pooling layer with activation function set to relu and callbacks set for early stopping if no improvement after 12 epochs the training will be stopped.
2. Performed **Data Augmentation** before passing the data to model as the model complexity is now increased and there may be information loss while training and also it helpful to increase **predictability** if model comes across some unseen and real time data.
3. After evaluation with **F1-score** metric the overall accuracy gained was **0.62** which compared to baseline model is **5 %** more and according to Loss and Accuracy curves , it shows the training loss are moving towards convergence and is not **overfitting** as much compared to the baseline model and the accuracy curves depicts the same. This is because of the inclusion of dropouts in middle layers.
4. The following visuals are the representation of model parameters, Training graphs, F1 score and some predictions.



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 48, 48, 32)        320
batch_normalization_12 (Bat  (None, 48, 48, 32)        192
chNormalization)
conv2d_22 (Conv2D)           (None, 48, 48, 32)        9248
max_pooling2d_12 (MaxPoolin  (None, 24, 24, 32)        0
g2D)
dropout_12 (Dropout)         (None, 24, 24, 32)        0
conv2d_23 (Conv2D)           (None, 24, 24, 64)        18496
batch_normalization_13 (Bat  (None, 24, 24, 64)        96
chNormalization)
conv2d_24 (Conv2D)           (None, 24, 24, 64)        36928
max_pooling2d_13 (MaxPoolin  (None, 12, 12, 64)        0
g2D)
dropout_13 (Dropout)         (None, 12, 12, 64)        0
conv2d_25 (Conv2D)           (None, 12, 12, 128)       73856
batch_normalization_14 (Bat  (None, 12, 12, 128)       48
chNormalization)
conv2d_26 (Conv2D)           (None, 12, 12, 128)       147584
max_pooling2d_14 (MaxPoolin  (None, 6, 6, 128)         0
g2D)
dropout_14 (Dropout)         (None, 6, 6, 128)         0
conv2d_27 (Conv2D)           (None, 6, 6, 256)         295168
batch_normalization_15 (Bat  (None, 6, 6, 256)         24
chNormalization)
max_pooling2d_15 (MaxPoolin  (None, 3, 3, 256)         0
g2D)
dropout_15 (Dropout)         (None, 3, 3, 256)         0
flatten_3 (Flatten)          (None, 2304)              0
dense_3 (Dense)              (None, 256)               590080
dense_4 (Dense)              (None, 7)                 1799
=================================================================
Total params: 1,173,839
```

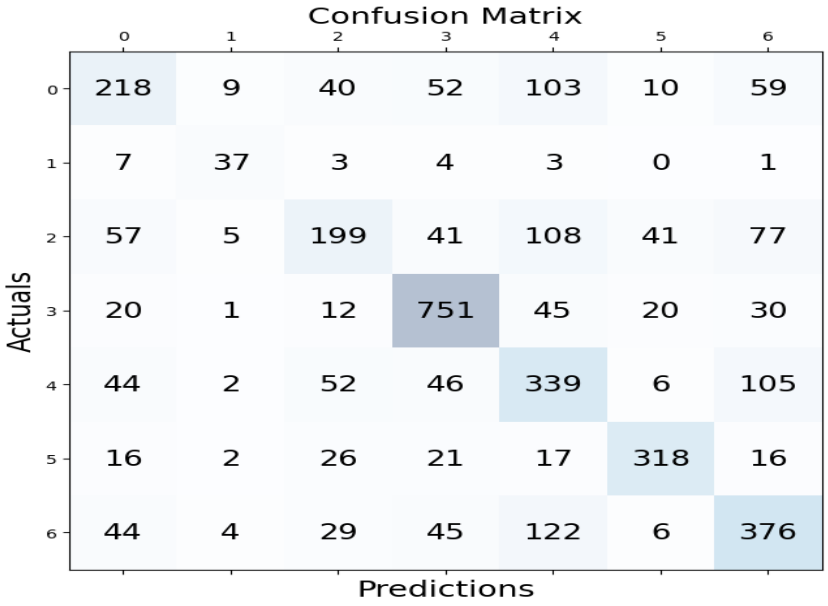


	precision	recall	f1-score	support
0	0.54	0.44	0.49	491
1	0.62	0.67	0.64	55
2	0.55	0.38	0.45	528
3	0.78	0.85	0.82	879
4	0.46	0.57	0.51	594
5	0.79	0.76	0.78	416
6	0.57	0.60	0.58	626
accuracy			0.62	3589
macro avg	0.62	0.61	0.61	3589
weighted avg	0.62	0.62	0.62	3589



Confusion Matrix:

This shows the predictability of each class, compared to the baseline the overall predictability is increased significantly and there is difference in samples as I used different data split for TRIAL 2 to check if there are any improvements after using original data distribution.



Transfer Learning
Goal: To run a VGG19 Model with **Imagenet** dataset weights to check for better performance compared to the normal models

Dataset	Facial Expression Recognition(FER)Challenge | Kaggle
Image size	48 x 48 x 3 (Modified for Transfer Learning models)

Parameters	1. optimizer = 'SGD' 2. Learning rate = 0.001 3. Loss = 'categorical_crossentropy' 4. Batch size = 32 , Epochs = 50 5. Data Augmentation

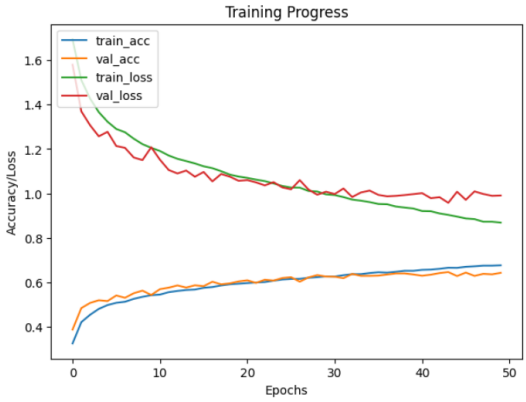
Insights:

1. Trained a model with VGG19 backbone and freezed the first two CNN blocks to retain the **weights** of **Imagenet** and other layers will be trained by the FER dataset as Imagenet is more optimized for **Face** recognition than the **Face Expression** recognition and callbacks set for early stopping if no improvement after 12 epoch training will be stopped.
2. Performed **Data Augmentation** before passing the data to model as the model complexity is now increased and there may be information loss while training and also it helpful to increase **predictability** if model comes across unseen and real time data.
3. After evaluation with **F1-score** metric the overall accuracy gained was **0.64** which compared to **TRIAL 2** model is slight better and according to Loss and Accuracy curves , it shows a slight increase in **overfitting** at the end as validation loss is saturated and training loss is still moving towards convergence.
4. I have trained till 50 epochs due to limited GPU resources in Colab and avoiding runtime disconnection while training the model.
5. Model predictability also increased compared to the previous models as shown below.

6. The following visuals are the representation of model parameters, Training graphs, F1 score and some predictions which shows increase in performance.

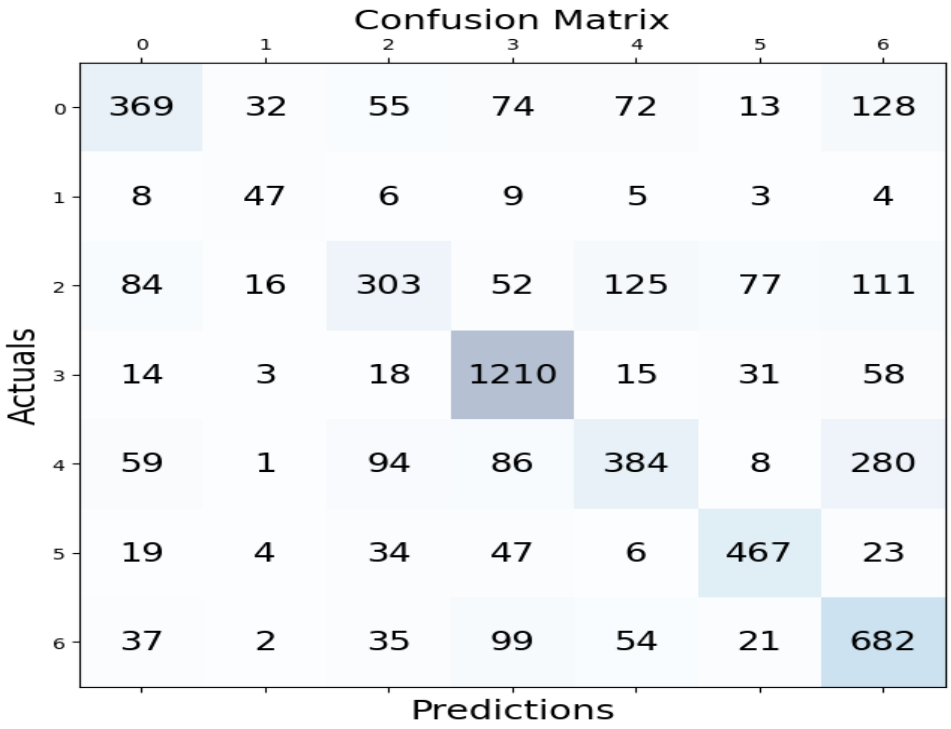
Layer (type)	Output Shape	Param
input_2 (InputLayer)	[(None, 48, 48, 3)]	0
block1_conv1 (Conv2D)	(None, 48, 48, 64)	1792
block1_conv2 (Conv2D)	(None, 48, 48, 64)	36928
block1_pool (MaxPooling2D)	(None, 24, 24, 64)	0
block2_conv1 (Conv2D)	(None, 24, 24, 128)	73856
block2_conv2 (Conv2D)	(None, 24, 24, 128)	147584
block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
block3_conv1 (Conv2D)	(None, 12, 12, 256)	295168
block3_conv2 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv3 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv4 (Conv2D)	(None, 12, 12, 256)	590080
block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
block4_conv1 (Conv2D)	(None, 6, 6, 512)	118016
block4_conv2 (Conv2D)	(None, 6, 6, 512)	235980
block4_conv3 (Conv2D)	(None, 6, 6, 512)	235980
block4_conv4 (Conv2D)	(None, 6, 6, 512)	235980
block4_pool (MaxPooling2D)	(None, 3, 3, 512)	0
block5_conv1 (Conv2D)	(None, 3, 3, 512)	235980
block5_conv2 (Conv2D)	(None, 3, 3, 512)	235980
block5_conv3 (Conv2D)	(None, 3, 3, 512)	235980
block5_conv4 (Conv2D)	(None, 3, 3, 512)	235980
flatten_2 (Flatten)	(None, 4608)	0
dense_3 (Dense)	(None, 512)	235980
dropout_1 (Dropout)	(None, 512)	0
dense_4 (Dense)	(None, 7)	3591

Total params: 22,387,783
Trainable params: 22,127,623
Non-trainable params: 260,160



	precision	recall	f1-score	support
0	0.63	0.50	0.55	743
1	0.45	0.57	0.50	82
2	0.56	0.39	0.46	768
3	0.77	0.90	0.83	1349
4	0.58	0.42	0.49	912
5	0.75	0.78	0.77	600
6	0.53	0.73	0.62	930
accuracy			0.64	5384
macro avg	0.61	0.61	0.60	5384
weighted avg	0.64	0.64	0.63	5384

Confusion Matrix:



Conclusion

- This report has provided a comprehensive overview of Convolutional Neural Networks (CNNs) and its application on **FER dataset**.
- Explored the basic building blocks of CNNs, such as convolutional layers, pooling layers, and fully connected layers, were examined, along with how these elements affect the network's capacity to learn and recognize complicated patterns.
- Best performing model with **64%** F1- score was achieved and good predictability on the test set as well and an average of performance of **61%** was achieved while experimenting with all the 3 models above.
- Data Distribution of classes is a very critical in improving overall predictability and the dataset had class imbalance and should be handled to achieve a well generalized model.
- Due to computation limitations the models were trained for less epochs for Trial 2 and Transfer learning because of their complexity , if trained for more epochs the models can perform better comparatively and learn all the parameters smoothly.
- It's crucial to understand that CNNs have their limitations. They make training and deploying large-scale models difficult since they need a lot of computational resources. Furthermore, CNNs are vulnerable to adversarial assaults, wherein slight changes to the input data might result in classification errors. By investigating model compression approaches, transfer learning, and enhancing resilience against adversarial attacks, researchers and practitioners are actively attempting to address these difficulties.

Future Work

- Working on data distribution of models with **SMOTETomek** which is a hybrid data sampling technique used in machine learning to address the issue of imbalanced datasets. It combines two methods: Synthetic Minority Over-sampling Technique (SMOTE) and Tomek links.
- Exploring on Generative Deep learning to generate reconstructed samples for better performance and generalization of model with reference to https://keras.io/examples/generative/vq_vae/
- **Bonus Task:** Planning to implement a model with enhancements from above techniques for emotion detection using video inputs with approach referenced to https://github.com/JustinShenk/fer . I have Implemented the similar in my academic project which included the processing of video to images and building a model for **Image Segmentation**.
- **GridSearchCV** for hyperparameter optimizations and to find the best parameters such as batch size , learning rates , activation functions , optimizers.
- Exploring ways to implement **Convolutional Transformers** and its benefits include parameter-efficiency and self-attention to process long-range and interactions between different regions in an image.

References

- https://keras.io/api/applications/vgg/
- https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/
- https://www.tensorflow.org/
- https://scikit-learn.org/stable/index.html
- https://imbalanced-learn.org/stable/references/generated/imblearn.combine.SMOTETomek.html
- https://keras.io/examples/vision/cct/