Image Classification Using CNN

Teja Sree Kilari

TEAM-BRAINIACS

ABSTRACT-

Emotion recognition has a wide variety of applications in the field of artificial intelligence (AI), including humancomputer interaction, data-driven animation, robothuman discourse, and a great many more. Within the realm of computer vision, this topic has been the focus of a great deal of research thanks to the fact that it presents a challenge while also being incredibly interesting to investigate. The purpose of this study is to develop a facial emotion recognition system by utilizing a convolutional neural network together with augmented data. The first technique uses visual data to recognize and classify seven fundamental feelings: anger, contempt, fear, happy, neutral, sorrow, and surprise. These feelings can be detected and grouped using the method. In the second approach, we make use of three different data sets, each of which includes information pertaining to one of the three different categories. Adding more data to a convolutional neural network not only helps to overcome the limitations of other models that are already out there, but it also helps to enhance the validation accuracy.

I. INTRODUCTION

Facial expression is how a person shows how they feel by moving the muscles on their face. It tells you what that person's mental health is like. Emotions are states of mind that people go through. It is how he feels inside as a result of what is going on around him. In many cases, you can tell something about a person's mental health by looking at his face. Analysis of face expressions can be used in many ways, like in lie detectors, art, and robotics. Face recognition needs to get better for an intelligent agent to be able to talk to humans and other robots as part of machine-human teamwork and robot-robot interaction, respectively.

Deep Learning algorithms are being created in order to emulate the functioning of the human cerebral cortex. These expressions are representations in a neural network with a very deep layer.

Convolutional neural networks are a form of deep learning technique that can be trained on enormous datasets with millions of parameters. These networks take their input from two-dimensional images and convolve it with various filters in order to get the outputs that are wanted. In this article, a CNN model is developed, and then its performance on image recognition and detection tasks is analyzed. In order to assess the performance of the method, we apply it to both the MNIST and CIFAR-10 datasets.
The 99.6 percent accuracy that CIFAR-10 was able to attain on the MNIST benchmark was accomplished through the combination of real-time data augmentation and CPU dropout.

Researchers devised a model known as a Convolutional Neural Network (CNN) to enhance the degree of precision with which images may be identified. All aspects of accuracy, including confusion matrices, loss and accuracy curves during training and validation, and total accuracy, are improved thanks to this model's optimizations. Training is done with a sizable collection of individual photographs. Convolutional layers, pooling layers, and fully connected layers make up the architecture of a CNN. These layers provide the network with the ability to learn the hierarchical characteristics of images. Altering the learning rate, selecting the appropriate optimizer, and tinkering with the hyperparameters are all activities that fall under the purview of the optimization process. The model is tested on a reference dataset, where it is discovered to perform at the same level as the current state of the art across a variety of measures. These metrics include overall accuracy, confusion matrix, and training/validation curves of accuracy and loss. The conclusion that can be drawn from this is that the CNN model that was provided is a feasible method for increasing the recognition performance of picture classification jobs. Convolutional neural networks, often known as CNNs, have become increasingly popular over the past several years as a result of the increased accuracy with which they perform image classification tasks. However, developing a CNN model that performs well for a particular job can be difficult since there are numerous elements that need to be taken into consideration, such as the network architecture, hyperparameters, optimizer, and learning rate. Within the scope of this investigation, we offer a CNN model with the purpose of enhancing the recognition performance of picture classification tasks. More specifically, we concentrate on improving the overall accuracy, confusion matrix, and training/validation curves of accuracy and loss. The proposed design for a CNN is made up of several layers, some of which are convolutional, while others are pooling and completely connected. The model requires the hierarchical information that is provided by these layers in order to perform accurate image classification. We conducted a series of experiments in which we changed the values of the model's hyperparameters, such as the stride, the kernel size, and the number of filters, in the hopes of improving the overall performance of the model. In addition, we experimented with a

number of different optimizers and learning rates in order to identify the configuration that would provide the greatest increase in both the model's accuracy and its speed. In order to examine the usefulness of the proposed CNN model, we made use of a benchmark dataset in addition to a number of different measures, such as overall accuracy, confusion matrix size, accuracy curves throughout training and validation, and loss. The findings revealed that our model attained state-of-the-art performance on some parameters, outperforming a number of earlier CNN models that were published in the literature.

The remaining sections of this article consist of: related work on facial expression recognition, an overview of the methodology of this research, data collection and preprocessing, experiment with data augmentation, how the proposed system has been implemented, result and discussion, conclusion and future work.

II. RELATED WORK

In recent years there have been great strides in building classifiers for image detection and recognition on various datasets using various machine learning algorithms. Deep learning, in particular, has shown improvement in accuracy on various datasets. Some of the works have been described below: Norhidayu binti Abdul Hamid et al. [3] evaluated the performance on MNIST datasets using 3 different classifiers: SVM (support vector machines), KNN (Knearest Neighbor) and CNN (convolutional neural networks). The Multilayer perceptron didn't perform well on that platform as it didn't reach the global minimum rather remained stuck in the local optimal and couldn't recognize digit 9 and 6 accurately. Other classifiers performed correctly and it was concluded that performance on CNN can be improved by implementing the model on Keras platform. Mahmoud M. Abu Gosh et al. [5] implement DNN (Deep neural networks), DBF (Deep Belief networks) and CNN (convolutional neural networks) on MNIST dataset and perform a comparative study. According to the work, DNN performed the best with an accuracy of 98.08% and other had some error rates as well as the difference in their execution time. Youssouf Chherawala et al. [6] built a vote weighted RNN (Recurrent Neural networks) model to determine the significance of feature sets. The significance is determined by weighted votes and their combination and the model is an application of RNN. It extracts features from the Alex word images and then uses it to recognize handwriting. Alex krizhevsky [7] uses a 2-layer

Convolutional Deep belief network on the CIFAR-10 dataset. The model built classified the CIFAR-10 dataset with an accuracy of 78.90% on a GPU unit. Elaborative differences in filters and their performance is described in the paper which differs with every model. Yehya Abouelnaga et al. [8] built an ensemble of classifiers on KNN. They used KNN in combination with CNN and reduced the Overfitting by PCA (Principal Component Analysis). The combination of these two classifiers improved the accuracy to about 0.7%. Yann le Cunn et al. [1] give a detailed introduction to deep learning and its algorithms. The algorithms like Backpropagation with multilayer perceptron, Convolutional neural networks, and Recurrent neural networks are discussed in detail with examples. They have also mentioned the scope of unsupervised learning in future in Artificial intelligence. Li Deng [10] details a survey on deep learning, its applications, architectures, and algorithms. The generative, discriminative and hybrid architectures are discussed in detail along with the algorithms that fall under the respective categories. CNN, RNN, Auto encodes, DBN's, RBM's (Restricted Boltzmann machines) are discussed with their various applications.

III. PROJECT BACKGROUND

The output each step should be followed proper manner in order to get good results the following steps are:

a) Data Pre-processing
b) Segmentation
c) CNN layer

a)Data Pre-Processing

The given unprocessed data set may contain noise in the data so to remove this noise pre-processing is used. It is the first step in the process to achieve good results in the prediction of a flower. This stage improves the image quality and by removing unwanted image content this is called noise reduction or removing the images with the very big size or dimensions that take more time and memory for processing in neural networks. The noise in the dataset includes low contrast images, irregular borders, etc. This noise is removed from the input dataset by using various filters like speckle noise, Poisson noise, Gaussian noise, and salt and pepper noise, including median filter, mean filter, adaptive median filter, Gaussian filter, and adaptive wiener filter.

For example, in the dataset, there may be images of dimensions 1024 X 1024 X 3 which is a huge dimension image that makes its future set huge for computation and analysis. When it passes to a neural network it takes more time and memory for analysis.

There are many pre-processing techniques to remove noise in the data set like colour correction, image smoothening, localization, normalization, vintage effect removal, black frame removal and contrast adjustment. The correct combination of these techniques give more accuracy. The skin cancer images are first converted into grayscale and then they undergo smoothing techniques. This pre-processing stage is very important. If this stage is neglected it will generate many problems in the model and provide low accurate results.

b)Segmentation

After pre-processing the data on the given dataset then segmentation on flower images. In this stage, the given input image is divided into regions to extract the necessary information for processing in further stages. It is mainly a division of the region of interest from the total image. The segmentation process is mainly classified into four stages. Segmentation based on pixel

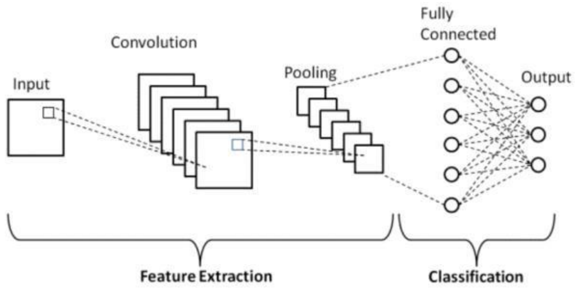
- Segmentation based region
- Segmentation based on threshold

- Segmentation based on the model

c)CNN Layers

In Convolution Neural Networks algorithm in Deep Learning has several layers through which the input image is passed through and examined to get the high accurate output. The CNN algorithm has the following layers

- Convolution layer
- Activation Layer
- Pooling layer **Convolution neural network**
- In the implementation of the Convolution Neural Networks algorithm there are several steps to build the model and predict
- Fully connected layer



1.Convolution Layer:-

The first layer extracts features from input photos. This layer performs convolution between the input picture and a filter of size MxM. The dot product is calculated by sliding the filter over the input image. (MxM).Feature maps show image corners and edges. Later, other layers use this feature map to learn more picture features. CNN's convolution layer passes the result to the next layer after convolution. CNN convolutional layers help maintain pixel spatial relationships.

2.Pooling Layer:-

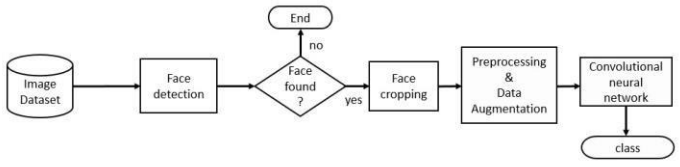
Pooling Layers usually follow Convolutional Layers. This layer reduces computing expenses by shrinking the convolved feature map. Each feature map is separately reduced via layer connections. Pooling processes vary by method. It summarizes convolution layer features. Feature map elements dominate Max Pooling. Average Pooling averages image section elements. Sum Pooling calculates the predefined section's sum. The Pooling Layer connects the Convolutional and FC Layers. This CNN model generalizes convolution layer characteristics and enables networks recognize them independently. This reduces network computations. **3.Full Connected Layer:-**
Weights, biases, and neurons form the Fully Connected (FC) layer, which links neurons from two layers. The last several layers of a CNN Architecture are these layers.Flattening and feeding the input image to the FC layer is done here. Mathematical functions are then performed on the flattened vector. At this point, classification begins. Two fully connected layers outperform one connected layer. CNN's tiers decrease human oversight.

4.Dropout:-

When all features are connected to the FC layer, the training dataset can overfit. When a model performs well on training data, it overfits and performs poorly on new data.A dropout layer is used during training to reduce model size by dropping a few neurons from the neural network. After a dropout of 0.3, 30% of the neural network's nodes are randomly removed. Dropout makes a machine learning model function better by preventing overfitting and simplifying the network. During training, it removes neurons from neural networks.

5.Activation Function:-

The CNN model's activation function is also crucial. They are used to learn and estimate any continuous and complex relationship between network variables. Simply said, it decides which model information should shoot ahead and which should not at the network's end. It makes the network non-linear. Activation functions like ReLU, Softmax, tanH, and Sigmoid are utilized often. These functions have specific uses. For binary classification CNN models, sigmoid and softmax functions are recommended, and for multi-class classification, softmax is utilized. CNN models use activation functions to activate or deactivate neurons. It determines whether mathematical procedures should be used to forecast input.



IV METHODOLOGY

<u>1.Emotion Recognition Data:</u> For Emotion Recognition Data details we had used Kaggle images as input. The URL for the input data is https://www.kaggle.com/datasets/jonathanoheix/faceexpression-recognition-dataset.

Emotion recognition data preparation and analysis approaches include:

Face detection and cropping<u>:</u> This entails finding a face in an image and cropping it. Face detection algorithms like the Haar Cascade classifier are used.

Image normalization: Subtracting the mean and dividing by the standard deviation normalizes image data. Lighting and contrast variations are reduced by this.

Data augmentation: This generates new training data by rotating, resizing, and flipping the original images. Overfitting is reduced and training data is increased.

Deep learning models: These examine picture data and anticipate emotions using deep learning models like convolutional neural networks (CNNs). CNNs can analyze picture data and recognize emotions with high accuracy.

The Kaggle Emotion Recognition Data dataset can be used with TensorFlow, Keras, PyTorch, and Scikit-learn. Preprocessing data, constructing models, and evaluating model performance are embedded into these frameworks. Keras and TensorFlow appear to offer the libraries needed to develop and train convolutional neural networks (CNNs) for image categorization. matplotlib. Data visualization library pyplot plots pictures and graphs.

Python accesses and manipulates files with pathlib and os. Python's NumPy package is used for array operations. Structured data can be analyzed with Python's pandas package. Google's open-source machine learning package, TensorFlow, is utilized for deep learning model training. Keras, a Python-based high-level neural networks API that runs atop TensorFlow, helps build and train deep learning models.

Keras and TensorFlow modules imported include:

Multi-class classification loss functions include categorical_crossentropy.

Model is a class that defines Keras models, which are directed acyclic graphs of layers.

Stochastic gradient descent uses Adam optimization.

Layers for CNNs include Dense, Flatten, Conv2D, MaxPooling2D, Activation, BatchNormalization, and Dropout.

Keras' utility class ImageDataGenerator augments and preprocesses image data.

These Keras and TensorFlow libraries and modules are used to build and train CNNs for image classification. The Keras ImageDataGenerator class preprocesses image input and generates training and validation sets in this code. Rescaling, shearing, zooming, and horizontal flipping are applied to training photos via the train_datagen object. These methods can help the model generalize and avoid overfitting.

The validation set does not require data augmentation, thus the val_datagen object just rescales the validation photos. Training and validation set generators are then created using flow_from_directory. The target picture size, batch size, and class mode are input to the method.

In this code, the target size is (48, 48) and the batch size is 128. Image labels are one-hot encoded vectors since the class mode is "categorical."This code creates data generators to efficiently train a deep learning model using image data.

The Keras Sequential API is used to define a convolutional neural network (CNN) in this code. The network has several layers, including convolutional layers, pooling layers, dropout layers, and thick layers.

The network's first layer is a Conv2D layer with 64 filters, a filter size of 3x3, and a relu activation function. The padding option is set to "same," which means the output size is the same as the input size. The input_shape parameter is set to (48,48,3), which means that the input images are 48x48 RGB images.

The second layer is a MaxPooling2D layer with a pool size of 2x2. This lowers the spatial dimensions of the output from the first layer by a factor of 2.

The third and fourth layers are like the first and second layers, but they have more filters and bigger filters.

The fifth and seventh layers are Dropout layers, which have a dropout rate of 0.25. Dropout is a regularization method that gets rid of some neurons at random during training to avoid overfitting.The sixth layer is another Conv2D layer with 512 filters of size 3x3.The eighth layer is a Flatten layer, which turns the result of the previous layer into a 1D array.The ninth and eleventh layers are both Dense layers with 256 and 512 units, respectively, and a relu activation function.The tenth and twelveth layers are Dropout layers with a dropout rate of 0.5.

The last layer is a Dense layer with the same number of units as the number of folders in the dataset and a softmax activation function. This layer gives out a probability distribution over the different classes, which lets the model predict the class of a new image.

Using the fit method in Keras, this code fits the stated CNN model to the training data. The fit method trains the model for a certain number of epochs using the training data. At the end of each epoch, the model is also tested using the validation data. The ImageDataGenerator class was used to make the training and validation sets, which are represented by the training_set and val_set variables.

The steps_per_epoch and validation_steps arguments tell the program how many steps (batches) to take from the training and validation data sources in each epoch, respectively. In this code, they are set to the length of the training and validation sets. This means that the full dataset is used in each epoch. The epochs statement says how many times the whole training dataset is used to train the model. It is set to 50 in this code, which means the model will be learned for 50 iterations.

Overall, this code is training a CNN model with the picture data and the hyperparameters that were given. At the end of each epoch, the model's performance is tested on the validation set.

2.Don’t touch your face:-

For Part2 Project we had used the data set fixed with the images of the students in the class. We had three classes for the image data set

“Class 1 Touch”

“Class 2 No hands”

“Class 3 No Touch w/ Hands”

In these we need to classify the images along with the the model loss and accuracy and confusion matrix. We had used Google collab to build our own CNN model.The libraries required in this experiment are Keras,Tensorflow,Numpy,Pandas and Matplotlib We imported image dataset as Zip file to the google drive. Tensorflow was used as system backend whereas keras helped the system by providing built-in functions like activation functions, optimizers, layers etc. Image preprocessing such as face detection (Cascade Classifier), grayscale conversion, image normalization. Data augmentation was performed by keras API. Matplotlib has been used to generate confusion matrix.

ImageDataGenerator creates training, validation, and test data using flow from directory.Images are rescaled, sheared, zoomed, and flipped during training. The training set's diversity and model performance improve.

This code creates two ImageDataGenerator objects. Train_datagen augments the training set, while val_datagen scales the validation and test sets.

The ImageDataGenerator flow from directory method generates training, validation, and test data. The dataset's directory location, photos' goal size, batch size, and class mode are inputs.

The target size option in this code enlarges all photographs to 48x48 pixels. (48, 48). If batch size is 128, the generator will batch 128 photographs. Categorical class mode displays labels as one-hot encoded vectors.

This code uses Keras' ImageDataGenerator class to augment training data and improve model performance for machine learning model training, validation, and testing. The CNN model's layers are stacked sequentially using the Sequential class.

Overfitting is mitigated through a series of max-pooling and dropout layers following each of the model's three convolutional layers. Convolutional operations are carried out by the Conv2D layer, which uses 128 3x3 filters, 64 3x3 filters, and 32 3x3 filters. By using a max-pooling operation with a pool size of 2x2, the MaxPooling2D layer can minimize the output feature maps' spatial dimensionality. Overfitting can be avoided with the use of the Dropout layer, which randomly sets some of the input units to zero during training.

Following the three convolutional layers, the output feature maps are converted into a 1D vector using a Flatten layer. Two fully linked layers (Dense) of size 128 and len(folders) are then applied to the output of the flattener. The SoftMax activation function is utilized in the output layer, while the relu activation function is employed in the first completely linked layer. The probabilities of each class's output are normalized by the SoftMax activation function so that they add up to one.

This code generates a CNN model for image recognition that consists of three convolutional layers, a max-pooling layer, a dropout layer, and two fully connected layers with relu and SoftMax activation functions.

Adam optimizer uses 0.0001 learning rate. Adam's adaptive learning rate optimization approach adjusts learning rate using a weighted moving average of the gradient and its square during training. It optimizes deep learning models well, making it popular.

The categorical cross entropy loss function addresses multiclass difficulties. It reduces variance between observed and predicted class probabilities as its training metric.

Accuracy determines performance in model training and validation. It measures the percentage of samples correctly categorized.

This code creates a CNN model with an Adam optimizer, categorical cross-entropy loss function, and accuracy measure for image recognition and multi-class classification.

The training set generates augmented photos from the training directory. The validation set data generator creates batches of non-augmented photos from the validation directory. The training and validation set batches determine the steps per epoch and validation steps parameters. The full dataset is seen once every epoch during training. The model will observe the training dataset 12 times, set by the epochs option. The model's weights are updated every epoch depending on the loss function's gradient. Fit() produces a History object with training and validation losses and accuracies for each epoch. This data can be used to assess model performance and identify overfitting or underfitting during training.

This program trains the CNN model for 12 epochs utilizing training and validation data generators for multi-class classification tasks like image recognition.

The consequence.history is a dictionary of training and validation loss and accuracy for each epoch. We visualize training and validation accuracy using the result.history object's 'accuracy' and 'val_accuracy' keys.Line graphs of training and validation accuracy are created using plt.plot(). Plot titles and labels are set using plt.title(), plt.ylabel(), and plt.xlabel(). The plot legend shows training and validation accuracy using plt.legend(). The plt.grid() method grids the plot for easier reading.

keys "loss" and "val_loss" from output.A history object can plot validation and training loss.

The plt.plot() method draws a line graph showing the difference between training and validation loss. The plt.title(), plt.ylabel(), and plt.xlabel() functions change the plot's title and labels. The plt.legend() method displays the plot's legend, which shows training and validation loss. Plt.grid() makes the plot more legible.

This code plots the CNN model's training and validation loss over the epochs, complete with titles and labels, so we can assess the model's performance and decide if it overfitted or underfitted.

Model.evaluate tests the learned model with the test_set generator.(). No evaluation progress indicator will appear since the verbose argument is 0.

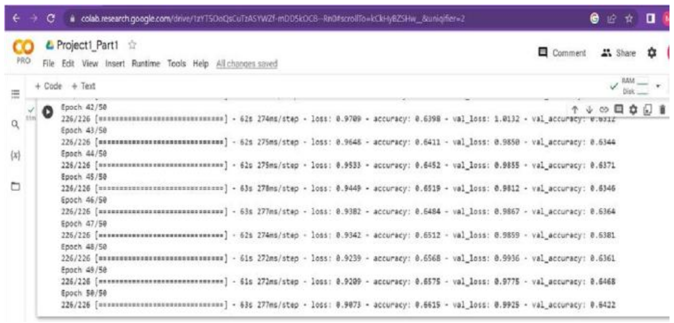
The method returns the model's loss and accuracy on the test set in loss and accuracy. These variables can display or print assessment results. sklearn.metrics' confusion_matrix() for validation set predictions.

The val_set.classes argument contains the validation set's class labels as integers.

The trained Model predicts the validation set class labels using the predict() function and the val_set generator object from the training set. On the final axis, the class with the highest probability is selected using the argmax() method. The validation set generator object's class_indices dictionary turns class names into integer indices.

Print the confusion_matrix() matrix from the actual and projected class labels.(). Class confusion matrices show right and incorrect classifications.

V RESULTS

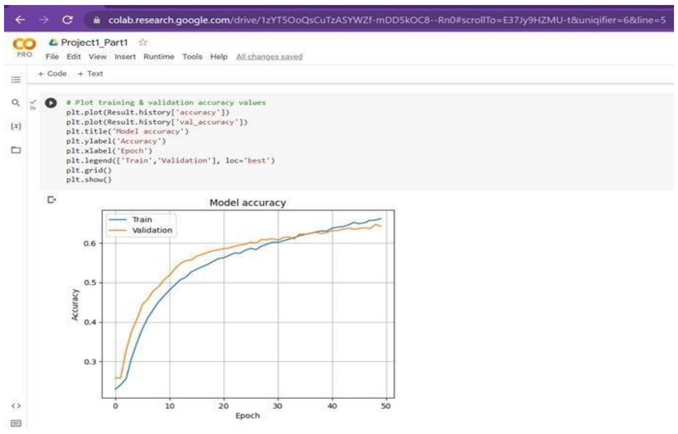

colab.research.google.com/drive/1zYT5OoQsCuTzASYWZf-mDD5kOCB--Rn0#scrollTo=kCkHyBZSHw_&uniqifier=2
Project1_Part1
File Edit View Insert Runtime Tools Help All changes saved
Comment Share
+ Code + Text
RAM Disk

```
Epoch 42/50
226/226 [==============================] - 62s 274ms/step - loss: 0.9709 - accuracy: 0.6398 - val_loss: 1.0132 - val_accuracy: 0.6312
Epoch 43/50
226/226 [==============================] - 62s 275ms/step - loss: 0.9648 - accuracy: 0.6411 - val_loss: 0.9850 - val_accuracy: 0.6344
Epoch 44/50
226/226 [==============================] - 62s 275ms/step - loss: 0.9533 - accuracy: 0.6452 - val_loss: 0.9855 - val_accuracy: 0.6371
Epoch 45/50
226/226 [==============================] - 63s 278ms/step - loss: 0.9449 - accuracy: 0.6519 - val_loss: 0.9812 - val_accuracy: 0.6346
Epoch 46/50
226/226 [==============================] - 63s 277ms/step - loss: 0.9382 - accuracy: 0.6484 - val_loss: 0.9867 - val_accuracy: 0.6364
Epoch 47/50
226/226 [==============================] - 62s 274ms/step - loss: 0.9342 - accuracy: 0.6512 - val_loss: 0.9859 - val_accuracy: 0.6381
Epoch 48/50
226/226 [==============================] - 61s 272ms/step - loss: 0.9239 - accuracy: 0.6568 - val_loss: 0.9936 - val_accuracy: 0.6361
Epoch 49/50
226/226 [==============================] - 61s 272ms/step - loss: 0.9209 - accuracy: 0.6575 - val_loss: 0.9775 - val_accuracy: 0.6468
Epoch 50/50
226/226 [==============================] - 63s 277ms/step - loss: 0.9073 - accuracy: 0.6615 - val_loss: 0.9925 - val_accuracy: 0.6422
```

Plot training & validation accuracy values:

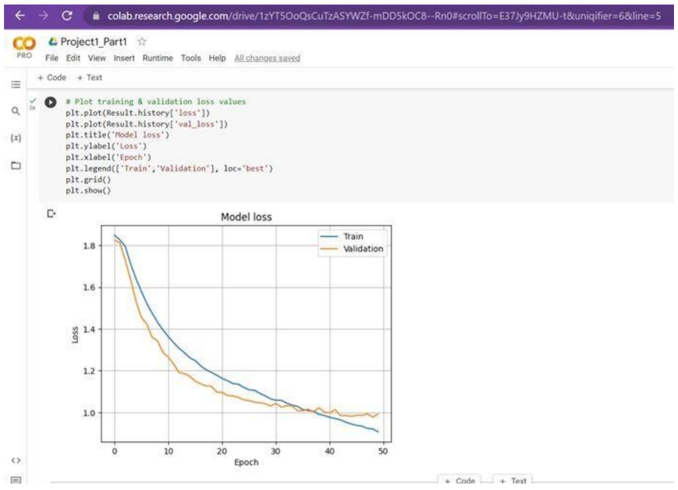
Project 1 Part1: **Emotion Data Recognition**

The accuracy of the data set by training the model:

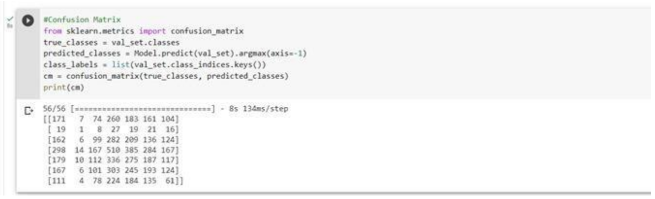

colab.research.google.com/drive/1zYT5OoQsCuTzASYWZf-mDD5kOC8--Rn0#scrollTo=E37Jy9HZMU-t&uniqifier=6&line=5
Project1_Part1
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Model accuracy
Train
Validation
Accuracy
Epoch
0.3
0.4
0.5
0.6
0
10
20
30
40
50

```
# Plot training & validation accuracy values
plt.plot(Result.history['accuracy'])
plt.plot(Result.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()
```

Plot training & validation loss values:


colab.research.google.com/drive/1zYT5OoQsCuTzASYWZf-mDD5kOC8--Rn0#scrollTo=E37Jy9HZMU-t&uniqifier=6&line=5
Project1_Part1
File Edit View Insert Runtime Tools Help All changes saved
+ Code + Text
Model loss
Train
Validation
Loss
Epoch
1.0
1.2
1.4
1.6
1.8
0
10
20
30
40
50
+ Code + Text

```
# Plot training & validation loss values
plt.plot(Result.history['loss'])
plt.plot(Result.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train','Validation'], loc='best')
plt.grid()
plt.show()
```

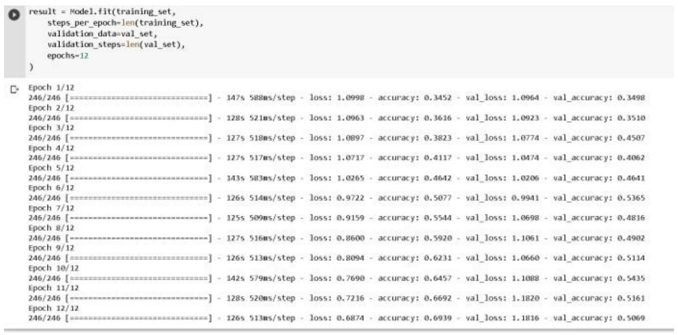


```
#Confusion Matrix
from sklearn.metrics import confusion_matrix
true_classes = val_set.classes
predicted_classes = Model.predict(val_set).argmax(axis=-1)
class_labels = list(val_set.class_indices.keys())
cm = confusion_matrix(true_classes, predicted_classes)
print(cm)
```

```
56/56 [==============================] - 8s 134ms/step
[[171   7  74 260 183 161 104]
 [ 19   1   8  27  19  21  16]
 [162   6  99 282 209 136 124]
 [298  14 167 510 385 284 167]
 [179  10 112 336 275 187 117]
 [167   6 101 303 245 193 124]
 [111   4  78 224 184 135  61]]
```

Project 1 Part2: **Don’t touch your face**

The accuracy of the data set by training the model:

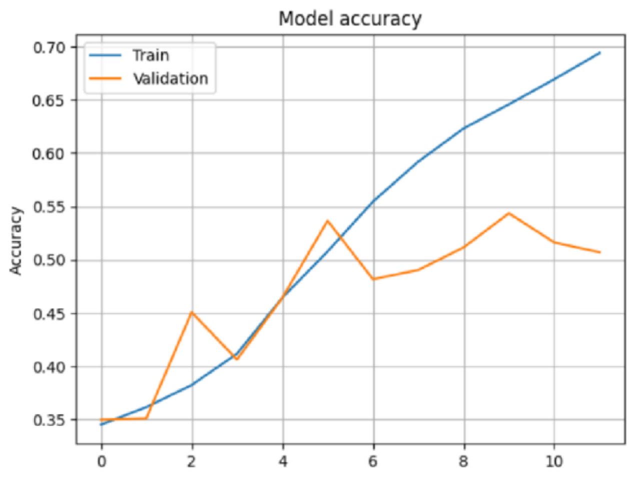


```
result = Model.fit(training_set,
    steps_per_epoch=len(training_set),
    validation_data=val_set,
    validation_steps=len(val_set),
    epochs=12
)
```

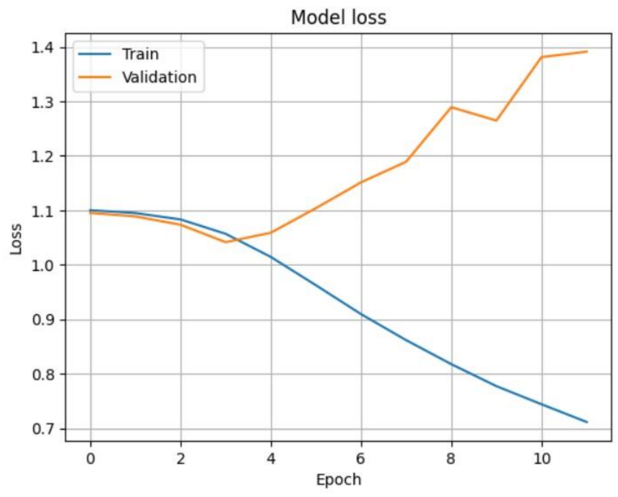
```
Epoch 1/12
246/246 [==============================] - 147s 588ms/step - loss: 1.0998 - accuracy: 0.3452 - val_loss: 1.0964 - val_accuracy: 0.3498
Epoch 2/12
246/246 [==============================] - 128s 521ms/step - loss: 1.0963 - accuracy: 0.3616 - val_loss: 1.0923 - val_accuracy: 0.3510
Epoch 3/12
246/246 [==============================] - 127s 518ms/step - loss: 1.0897 - accuracy: 0.3823 - val_loss: 1.0774 - val_accuracy: 0.4507
Epoch 4/12
246/246 [==============================] - 127s 517ms/step - loss: 1.0717 - accuracy: 0.4117 - val_loss: 1.0474 - val_accuracy: 0.4062
Epoch 5/12
246/246 [==============================] - 143s 583ms/step - loss: 1.0265 - accuracy: 0.4642 - val_loss: 1.0206 - val_accuracy: 0.4641
Epoch 6/12
246/246 [==============================] - 126s 514ms/step - loss: 0.9722 - accuracy: 0.5077 - val_loss: 0.9941 - val_accuracy: 0.5365
Epoch 7/12
246/246 [==============================] - 125s 509ms/step - loss: 0.9159 - accuracy: 0.5544 - val_loss: 1.0698 - val_accuracy: 0.4816
Epoch 8/12
246/246 [==============================] - 127s 516ms/step - loss: 0.8600 - accuracy: 0.5920 - val_loss: 1.1061 - val_accuracy: 0.4902
Epoch 9/12
246/246 [==============================] - 126s 513ms/step - loss: 0.8094 - accuracy: 0.6231 - val_loss: 1.0660 - val_accuracy: 0.5114
Epoch 10/12
246/246 [==============================] - 142s 579ms/step - loss: 0.7690 - accuracy: 0.6457 - val_loss: 1.1088 - val_accuracy: 0.5435
Epoch 11/12
246/246 [==============================] - 128s 520ms/step - loss: 0.7216 - accuracy: 0.6692 - val_loss: 1.1820 - val_accuracy: 0.5161
Epoch 12/12
246/246 [==============================] - 126s 513ms/step - loss: 0.6874 - accuracy: 0.6939 - val_loss: 1.1816 - val_accuracy: 0.5069
```

Confusion Matrix:

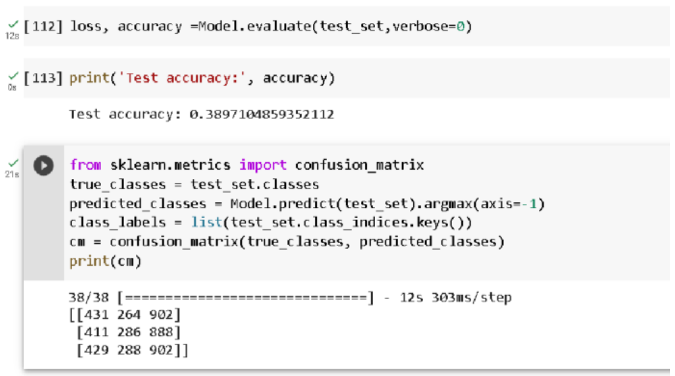
Plot training & validation accuracy values:



Plot training & validation loss values:



Confusion Matrix:



```
[112] loss, accuracy =Model.evaluate(test_set,verbose=0)

[113] print('Test accuracy:', accuracy)

      Test accuracy: 0.3897104859352112

      from sklearn.metrics import confusion_matrix
      true_classes = test_set.classes
      predicted_classes = Model.predict(test_set).argmax(axis=-1)
      class_labels = list(test_set.class_indices.keys())
      cm = confusion_matrix(true_classes, predicted_classes)
      print(cm)

      38/38 [==============================] - 12s 303ms/step
      [[431 264 902]
       [411 286 888]
       [429 288 902]]
```

VI. ROLES AND RESPONSIBILITIES

VinushaGarlapati:

Importing required libraries, preprocessing of data like image shaping and conversion from image to array and designing the CNN model for Part-2. Collecting the data for Documentation and major part in document editing.

Teja Sree Kilari :

In the first part of the project, I am responsible for importing the necessary libraries, downloading the data set from the Kaggle website and transferring it to the drive, as well as extracting, processing, and presenting the images and classes contained in the data set. After defining the CNN model for the dataset a a training model for the dataset using 50 Epochs is designed, the accuracy of the CNN model is 66.15.The model loss and model accuracy graphs have been coded for project 1 Part 2.Studied are a different IEEE papers, Kaggle programs and used google for documentation purposes.

Venkata Sai Prahlad :

In the context of this project, it is my duty to create the code that will be used for the confusion matrix in both part 1 and part 2.The information that is gathered from the code outputs is utilized in the process of constructing the document using the IEEE format.Google is used for analysis on the other data, and open source is being used to create the document.

REFERENCES

[1] Yann LeCun, Yoshua Bengio, Geoffery Hinton, “Deep Learning”, Nature, Volume 521, pp. 436-444, Macmillan Publishers, May 2015.

[2] Norhidayu binti Abdul Hamid, Nilam Nur Binti Amir Sjarif, “Handwritten Recognition Using SVM, KNN and Neural Network”, www.arxiv.org/ftp/arxiv/papers/1702/1702.00723

[3] Cheng-Lin Liuↄ ,Kazuki Nakashima, Hiroshi Sako, Hiromichi Fujisawa, “Handwritten digit recognition: benchmarking of stateof-the-art techniques”, ELSEVIER, Pattern Recognition 36 (2003) 2271 – 2285).

[4] Ping kuang, Wei-na cao and Qiao wu, “Preview on Structures and Algorithms of Deep Learning”, 11th International Computer Conference on Wavelet Actiev Media Technology and Information Processing (ICCWAMTIP), IEEE, 2014. [
5] Mahmoud M. Abu Ghosh ; Ashraf Y. Maghari, “A Comparative Study on Handwriting Digit Recognition Using Neural Networks”, IEEE, 2017.

[6] Youssouf Chherawala, Partha Pratim Roy and Mohamed Cheriet, “Feature Set Evaluation for Offline Handwriting Recognition Systems: Application to the Recurrent Neural Network,” IEEE Transactions on Cybernetics, VOL. 46, NO. 12, DECEMBER 2016. [7] Alex Krizhevsky, “Convolutional Deep belief Networks on CIFAR-10”. Available: https://www.cs.toronto.edu/~kriz/convcifar10aug2010.pdf.

[8] Yehya Abouelnaga , Ola S. Ali , Hager Rady , Mohamed Moustafa, “ CIFAR-10: KNN-based ensemble of classifiers”, IEEE, March 2017.

[9] Caifeng Shan, Shaogang Gong, Peter W. McOwan,

“Facial expression recogniton based on Local binary patterns: A comprehensive study”, ELSEVIER, Image and Vision Computing 27, pp. 803-816, 2009.

[10] Li Deng, “A tutorial survey of architectures, algorithms, and applications of Deep Learning”, APSIPA Transactions on Signal and Information Processing (SIP), Volume 3, 2014