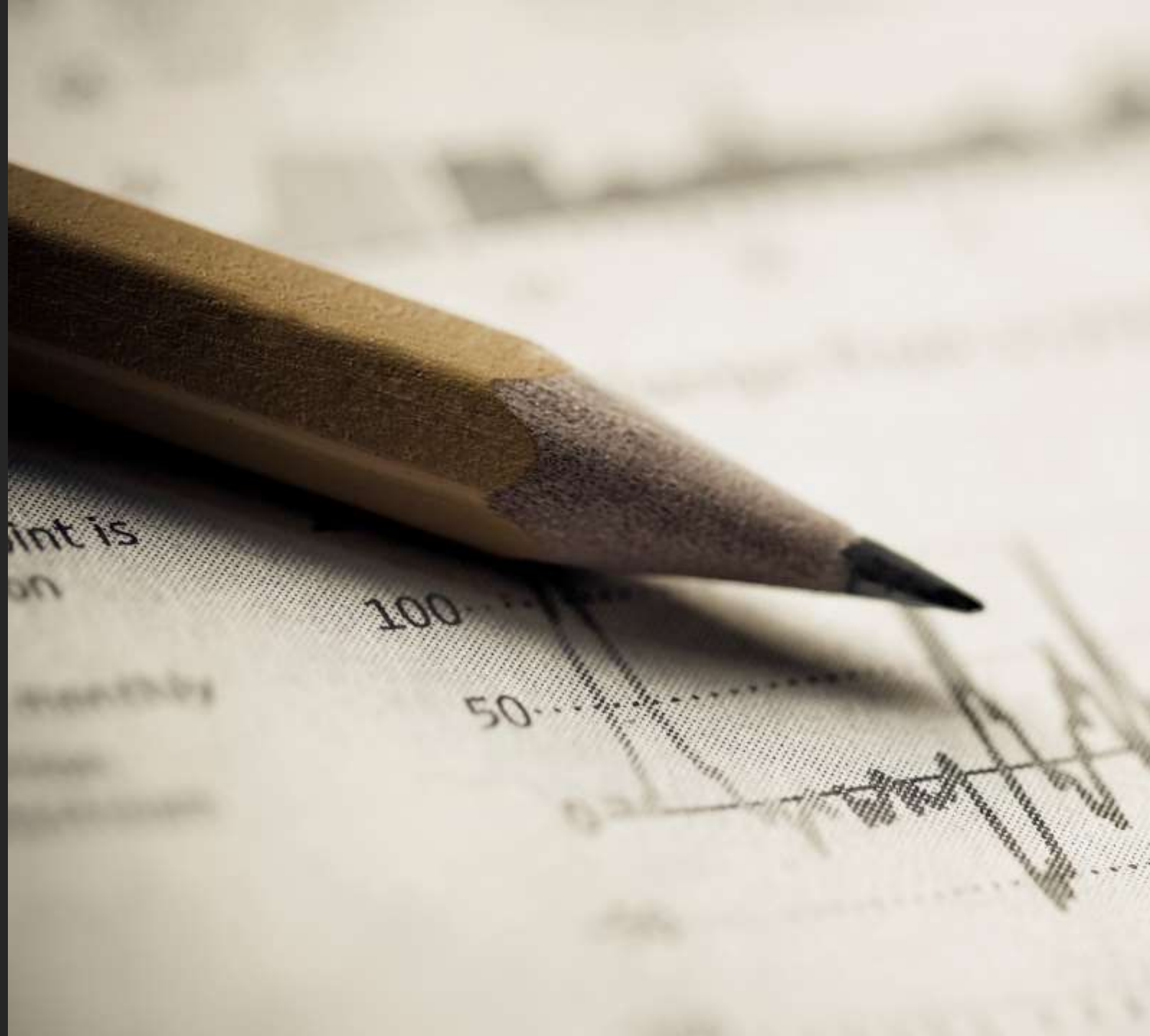
Efficient Facial Expression Recognition Algorithm Based on Hierarchical Deep Neural Network Structure



Why Facial Recognition?

FACIAL EXPRESSION RECOGNITION (FER) IS AN IMPORTANT TYPE OF VISUAL INFORMATION THAT CAN BE USED TO UNDERSTAND A HUMAN'S EMOTIONAL SITUATION. IN PARTICULAR, THE IMPORTANCE OF AI SYSTEMS HAS RECENTLY INCREASED DUE TO ADVANCEMENTS IN RESEARCH ON AI SYSTEMS APPLIED TO AI ROBOTS.

THEREFORE, THE MOST NECESSARY TECHNOLOGY IS A VISION SENSOR, AS VISION IS A LARGE PART OF HUMAN PERCEPTION IN MOST INTERACTIONS.

Facial Expression Detection Process

Data Set Description:

The Dataset that we have taken is CK+ dataset which is also called The Extended Cohn-Kanade Dataset consists of different types of emotion specified expressions. They are:

->Anger

->Contempt

->Disgust

→Fear

->Happy

->Sadness

->Surprise

Neural Networks:

In neural Networks we are using the concept of Convolutional Neural Network(CNN) has also been successfully used in computer vision applications. The CNN is a network that extracts feature maps by performing a convolutional operation with kernel on the original data. It is typically constructed of convolutional layers and pooling layers that extract feature maps expressing an image.

This CNN structure makes it possible to learn while preserving the shape characteristics of each component of the face image. In addition, maintaining the shape of the input and output data of each layer allows for the effective extraction of the facial expression features by considering the characteristics of the adjacent image.

Data Preprocessing:

-> The data such as images in each folder of the specified category is all stored in a list format.

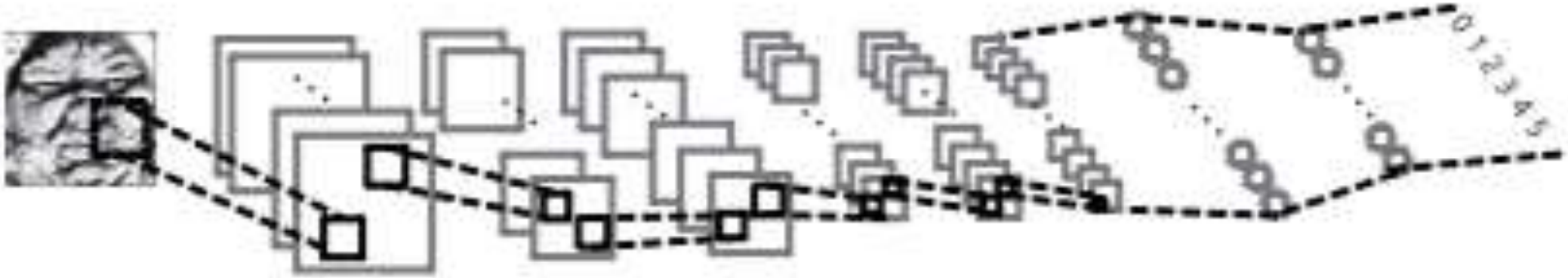
->Since computational complexity is high for computing pixel values in the range of (0-255), the data in pixel field is normalized to values between [0-1].

->The face objects stored are reshaped and resized to the mentioned size of 48 X 48.

->We use scikit-learn’s train_test_split() function to split the dataset into training and testing data. The test_size being 0.15 meaning, 15% of data is for validation while 85% of the data will be trained.

```
data_path = './ck/CK+48'
data_dir_list = os.listdir(data_path)

img_rows=256
img_cols=256
num_channel=1

num_epoch=10

img_data_list=[]


for dataset in data_dir_list:
    img_list=os.listdir(data_path+'/'+ dataset)
    print ('Loaded the images of dataset-'+'{}\n'.format(dataset))
    for img in img_list:
        input_img=cv2.imread(data_path + '/'+ dataset + '/'+ img )
        #input_img=cv2.cvtColor(input_img, cv2.COLOR_BGR2GRAY)
        input_img_resize=cv2.resize(input_img,(48,48))
        img_data_list.append(input_img_resize)

img_data = np.array(img_data_list)
img_data = img_data.astype('float32')
img_data = img_data/255
img_data.shape
```

```
Loaded the images of dataset-anger

Loaded the images of dataset-contempt

Loaded the images of dataset-disgust

Loaded the images of dataset-fear

Loaded the images of dataset-happy

Loaded the images of dataset-sadness

Loaded the images of dataset-surprise

(981, 48, 48, 3)
```



	Input LBP Image	convolution	pooling	convolution	pooling	convolution	pooling	Fully Connected	Fully Connected	
Image size	128×128	64×128×128	64×64×64	128×64×64	128×32×32	256×32×32	256×16×16	1024	500	output
kernel size		5×5	2×2	5×5	2×2	5×5	2×2			

The CNN - Structure

Convolutional 2D & MaxPooling

The 2D convolution is a simple operation at heart: you start with a kernel, which is simply a small matrix of weights. This kernel “slides” over the 2D input data, performing an elementwise multiplication with the part of the input it is currently on, and then summing up the results into a single output pixel.

Here we are learning a total of 6 filters for the first layer of the network and a total of 16 layers for the second layer of the network and again a total of 64 filters of learning in the network.

Then after we use Max Pooling to reduce the spatial dimensions of the output volume as we know that we will have to reduce the pool_size to half hence we give the tuple as (2,2).

Activation Function

We use rectified linear unit (ReLU) as the activation function between the convolutional layer and fully connected layer. This activation function is a step for converting the quantitative value of the feature map through the convolution operation into a nonlinear value. At the end of the network, six emotions are extracted as continuous values using the softmax function

The softmax function evaluates the exponential value of network and then gives the emotion score. After the process of error evaluation takes place.

```
model = Sequential()
model.add(Conv2D(6, (5, 5), input_shape=input_shape, padding='same', activation = 'r
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(16, (5, 5), padding='same', activation = 'relu'))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(128, activation = 'relu'))
model.add(Dropout(0.5))
model.add(Dense(7, activation = 'softmax'))
```

Optimizer, Loss function

Loss function used is categorical crossentropy

Loss function simply measures the absolute difference between our prediction and the actual value. The error is computed by the network through this process, and the error is reduced by using the cross entropy loss function

Using the cross entropy function, it is possible to flexibly respond to various probability distributions of the mode

Optimizer used is the Adam() optimizer. Adam stands for Adaptive Moment Estimation. Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter.

```
model.compile(loss='categorical_crossentropy', metrics=['accuracy'],optimizer='adam')
```

For Compiling the Model Using ‘loss’, ’metrics’, ’optimizer’

Validation

->Finally, fitting the model using the parameters

-X_train,

-y_train,

-batch_size(Which represents the subset of data using for updation of weight during its learning process) ,

-epochs=50(Which represents the number of iterations to be performed for the training of the data),

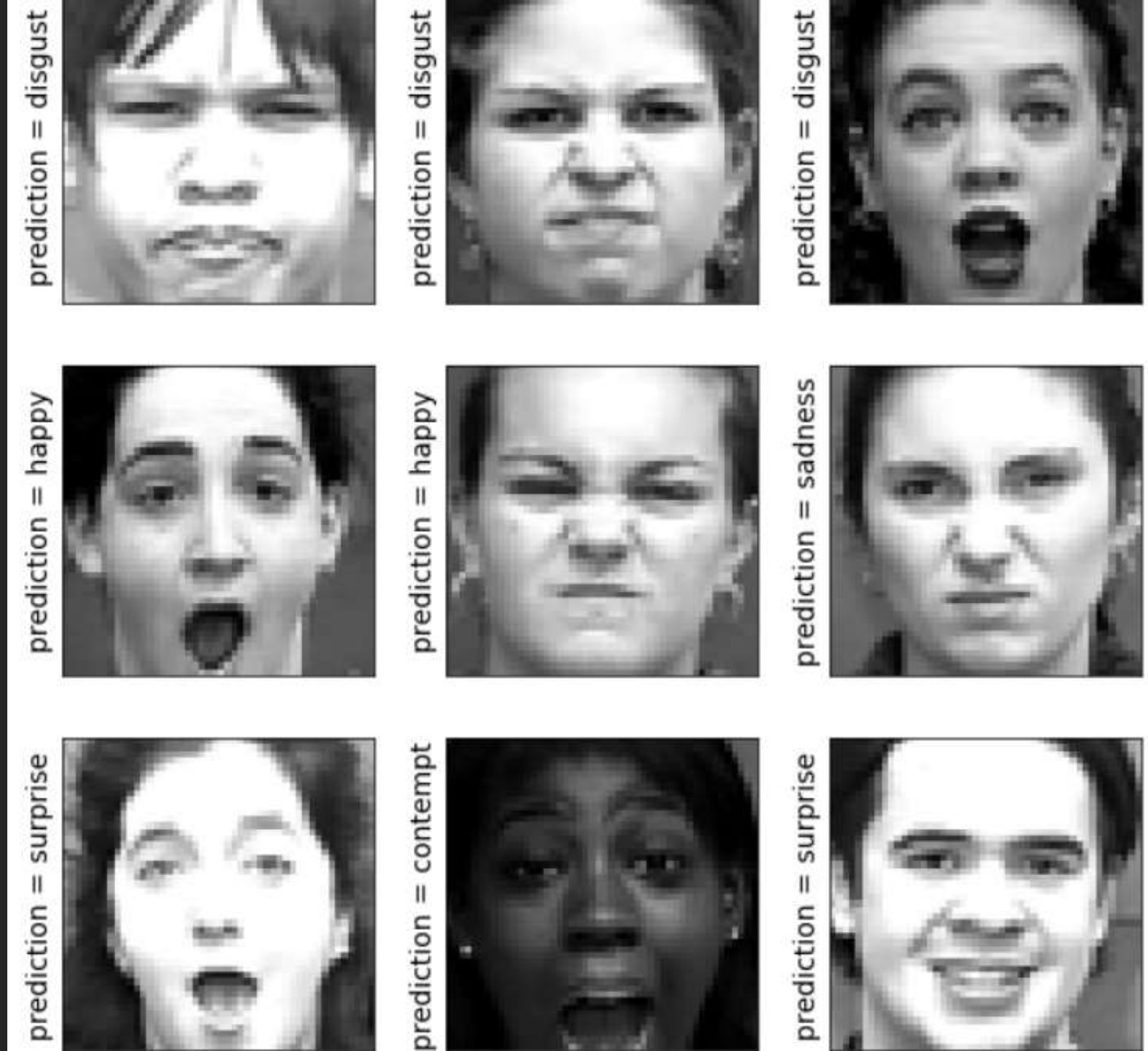
verbose(Which shows the progress bar for each epoch).

```
Train on 833 samples, validate on 148 samples
Epoch 1/50
833/833 [==============================] - 4s 5ms/step - loss: 1.7924 - accuracy: 0.3097 - val_loss: 1.5485 - val_accuracy:
0.5743
Epoch 2/50
833/833 [==============================] - 2s 3ms/step - loss: 1.1850 - accuracy: 0.5654 - val_loss: 0.8467 - val_accuracy:
0.7568
Epoch 3/50
833/833 [==============================] - 2s 3ms/step - loss: 0.8043 - accuracy: 0.6939 - val_loss: 0.6550 - val_accuracy:
0.8311
Epoch 4/50
833/833 [==============================] - 3s 4ms/step - loss: 0.6437 - accuracy: 0.7671 - val_loss: 0.4377 - val_accuracy:
0.8514
Epoch 5/50
833/833 [==============================] - 2s 3ms/step - loss: 0.4787 - accuracy: 0.8307 - val_loss: 0.3266 - val_accuracy:
0.9054
Epoch 6/50
```

Generated Output

Output values after prediction of the test data and Accuracy of the model.

For each image the expression is evaluated and represented with respective with the image.



97.972971200943

Conclusion

->We Finally created a model for the Facial Expression System using the Convolution Neural Network which can classify the given six types of facial expressions such as anger, disgust, surprise, sadness, happy, contempt.

->It gives a accuracy of 97.972971 and the Loss of 12.082313.

```
Test Loss: 12.082313068468293
Test accuracy: 97.972971200943
(1, 48, 48, 3)
```

THANK YOU…