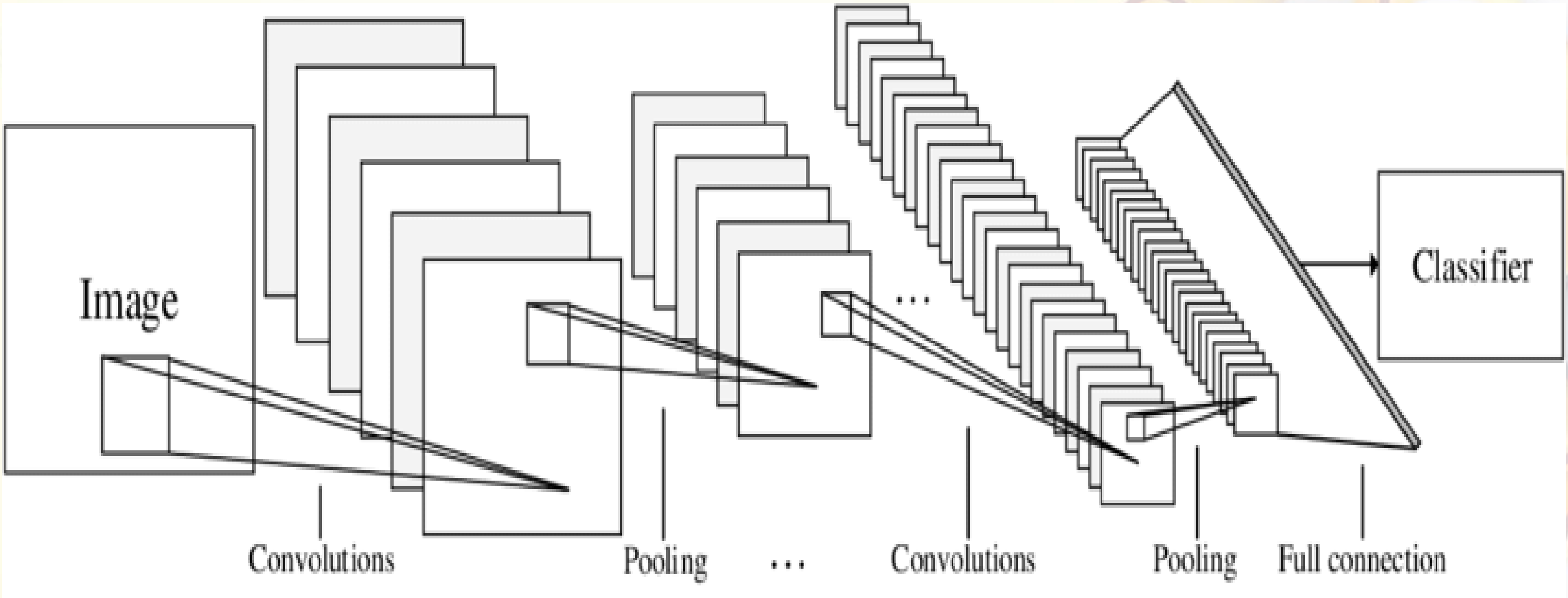
Topics to be covered :-

- Introduction to the Problem
- Importance and Need
- Objective and Scope
- Working Principle and Methodology
- Datasets
- Software and Hardware Requirements
- Conclusion
- References

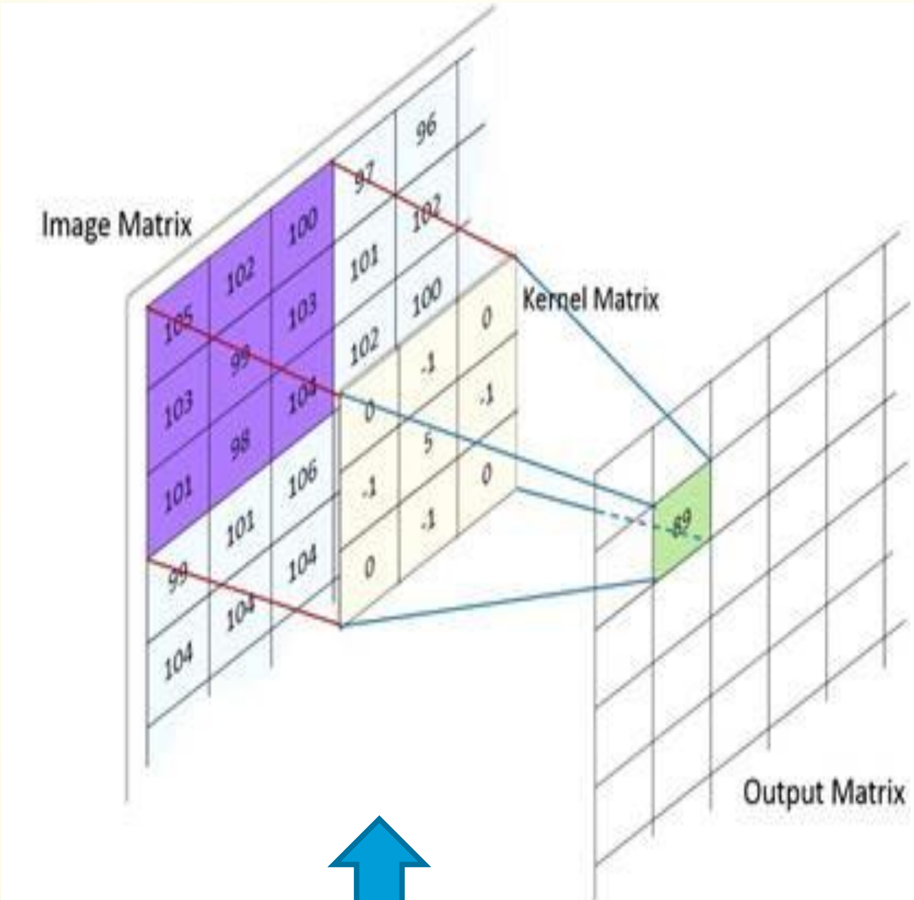
Why to recognise Faces :-

- The motivation behind choosing this topic specifically lies in the huge investments large corporations do in feedbacks and surveys but fail to get equitable response on their investments.
- Facial Expression Recognition through facial gestures is a technology that aims to improve product and services performance by monitoring customer facial expressions to certain products or service staff by their evaluation.

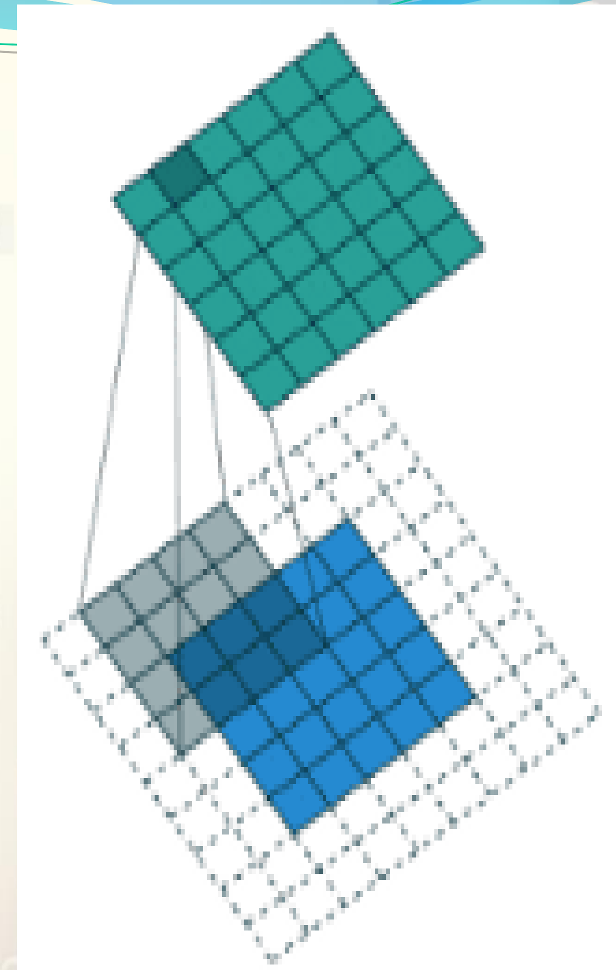
Recap to theory (explained earlier):



LeNet Architecture (Broad View)

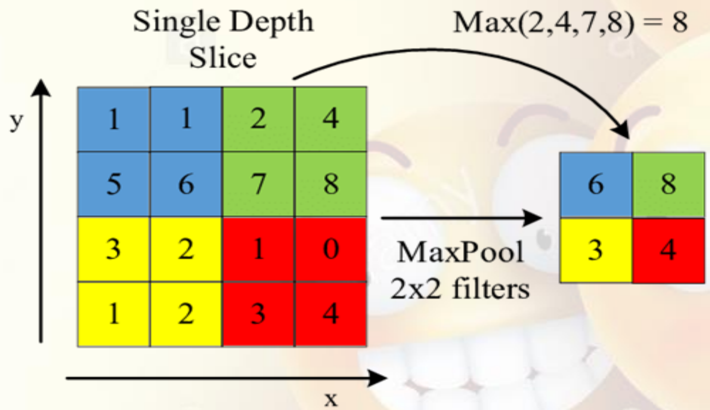


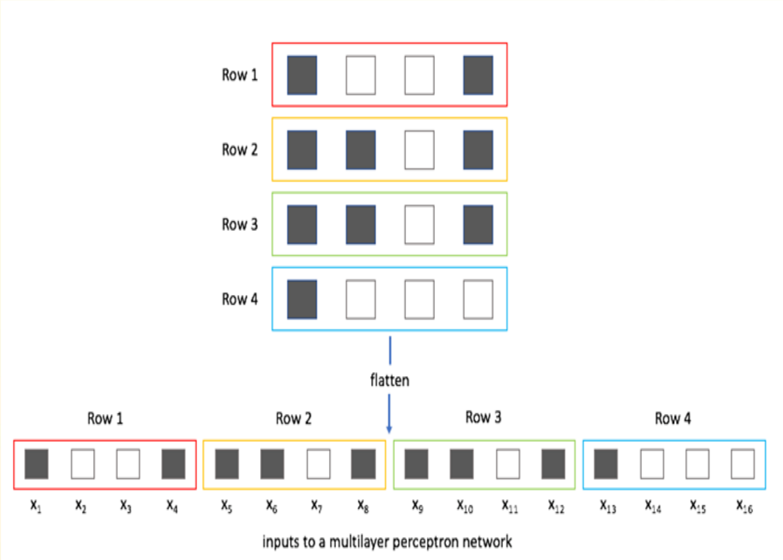
Step 1. Convolution



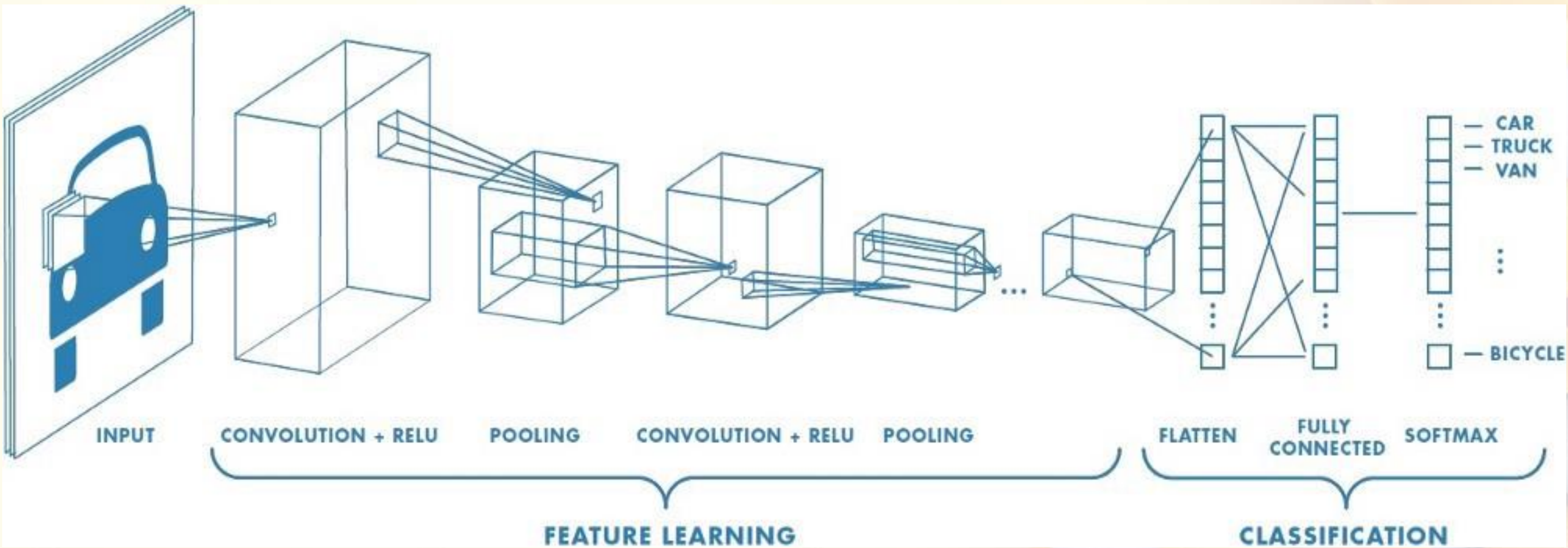
Step 2. Image Padding

Step 3. Max Pooling





Step 4. Image Flattening



Step 5. Feature Learning and Classification

Implementation of Convolution Neural Network

We have used Python Programming Language for implementation purpose.

Major Steps are:

- Data Preprocessing
- Feature Extraction
- Training and Validation
- Testing

1. Data Preprocessing

- The dataset from a Kaggle Facial Expression Recognition Challenge
- (FER2013) is used for the training and testing.
- The fer2013.csv consists of three columns namely emotion, pixels and purpose.

```
data = pd.read_csv('./fer2013.csv')
```

- The column in pixel first of all is stored in a list format.
- The face objects stored are reshaped and resized to the mentioned size of 48 X 48.

```
width, height = 48, 48

datapoints = data['pixels'].tolist()

#getting features for training
X = []
for xseq in datapoints:
    xx = [int(xp) for xp in xseq.split(' ')]
    xx = np.asarray(xx).reshape(width, height)
    X.append(xx.astype('float32'))
print(X)
```

- The respective emotion labels and their respective pixel values are stored in objects.

```
X = np.asarray(X)
print(X)
X = np.expand_dims(X, -1)
print(X)
#getting labels for training
y = pd.get_dummies(data['emotion']).values

#storing them using numpy
np.save('fdataX', X)
np.save('flabels', y)
```

2. Feature Extraction using Haar Cascade Classifier

- Haar Cascade classifier is an effective object detection approach which was proposed by **Paul Viola and Michael Jones** in their paper, "**Rapid Object Detection using a Boosted Cascade of Simple Features**" in 2001.

- Cascade function is trained from a lot of images both positive and negative. Based on the training it is then used to detect the objects in the other images.

- They are huge individual .xml files with a lot of feature sets and each xml corresponds to a very specific type of use case and in our case we have used frontal-face haar cascade detector to detect front faces.

Haarcascade Classifier (Frontal Face)

Fig. 3

3. Training the model

Step 1: **Load the files generated in preprocessing step**

```
x = np.load('./fdataX.npy')
y = np.load('./flabels.npy')

x -= np.mean(x, axis=0)
x /= np.std(x, axis=0)

#for xx in range(10):
#    plt.figure(xx)
#    plt.imshow(x[xx].reshape((48, 48)), interpolation='none', cmap='gray')
#plt.show()

#splitting into training, validation and testing data
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.1, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.1, random_state=41)

#saving the test samples to be used later
np.save('modXtest', X_test)
np.save('modytest', y_test)
```

Designing Convolution Layer

The 2D convolution is a fairly simple operation at heart:

We 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 summingup the results into a single output pixel.

Step 2: Designing the Convolution Neural Network

```
#desinging the CNN
model = Sequential()

model.add(Conv2D(num_features, kernel_size=(3, 3), activation='relu', input_shape=(width, height, 1), data_format='channels_last', kernel_regularizer=l2(0.01)))
model.add(Conv2D(num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))
```

Step 3: Adding the layer in convolution neural network

```
model.add(Conv2D(2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(Conv2D(2*num_features, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
model.add(Dropout(0.5))
```

Step 4: Flattening the layer and getting the maximum probable expression

```
model.add(Flatten())

model.add(Dense(2*2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*2*num_features, activation='relu'))
model.add(Dropout(0.4))
model.add(Dense(2*num_features, activation='relu'))
model.add(Dropout(0.5))

model.add(Dense(num_labels, activation='softmax'))
```

Step 5: Compile and fit the model

```python
#Compliling the model with adam optimizer and categorical crossentropy loss
model.compile(loss=categorical_crossentropy,
              optimizer=Adam(lr=0.001, beta_1=0.9, beta_2=0.999, epsilon=1e-7),
              metrics=['accuracy'])

#training the model
model.fit(np.array(X_train), np.array(y_train),
          batch_size=batch_size,
          epochs=epochs,
          verbose=1,
          validation_data=(np.array(X_valid), np.array(y_valid)),
          shuffle=True)

#saving the  model to be used later
model_json = model.to_json()
with open("model.json", "w") as json_file:
    json_file.write(model_json)
model.save_weights("model.h5")
print("Saved model to disk")
```

Through these steps, .json file and .h5 files are generated, containing the weights of neural networks

4. Testing

Step 6: Load the .json and .h5 file

Loading .json file

```
json_file = open('model.json', 'r')
loaded_model_json = json_file.read()
json_file.close()
loaded_model = model_from_json(loaded_model_json)
# Load weights into new model
loaded_model.load_weights("model.h5")
print("Loaded model from disk")
```

Loading .h5 model

.json file data

sers > S Rani > Desktop > Facial Expression Recognition > {} model.json > ...

{"class_name": "Sequential", "keras_version": "2.2.4", "config": {"layers": [{"class_name": "Conv2D", "config": {"kernel_initializer":
{"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_1",
"kernel_constraint": null, "bias_regularizer": null, "bias_constraint": null, "dtype": "float32", "activation": "relu", "trainable": true,
"data_format": "channels_last", "filters": 64, "padding": "valid", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer":
{"class_name": "L1L2", "config": {"l2": 0.009999999776482582, "l1": 0.0}}, "bias_initializer": {"class_name": "Zeros", "config": {}},
"batch_input_shape": [null, 48, 48, 1], "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}}, {"class_name": "Conv2D",
"config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.
0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_2", "bias_regularizer": null, "bias_constraint": null, "activation": "relu", "trainable":
true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 64,
"bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}},
{"class_name": "BatchNormalization", "config": {"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}},
"moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name": "batch_normalization_1", "epsilon": 0.001, "trainable": true,
"moving_variance_initializer": {"class_name": "Ones", "config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true,
"axis": -1, "gamma_constraint": null, "gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.99, "center": true}}, {"class_name":
"MaxPooling2D", "config": {"name": "max_pooling2d_1", "trainable": true, "data_format": "channels_last", "pool_size": [2, 2], "padding":
"valid", "strides": [2, 2]}}, {"class_name": "Dropout", "config": {"rate": 0.5, "noise_shape": null, "trainable": true, "seed": null, "name":
"dropout_1"}}, {"class_name": "Conv2D", "config": {"kernel_constraint": null, "kernel_initializer": {"class_name": "VarianceScaling", "config":
{"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}}, "name": "conv2d_3", "bias_regularizer": null, "bias_constraint":
null, "activation": "relu", "trainable": true, "data_format": "channels_last", "padding": "same", "strides": [1, 1], "dilation_rate": [1, 1],
"kernel_regularizer": null, "filters": 128, "bias_initializer": {"class_name": "Zeros", "config": {}}, "use_bias": true,
"activity_regularizer": null, "kernel_size": [3, 3]}}, {"class_name": "BatchNormalization", "config": {"beta_constraint": null,
"gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros", "config": {}}, "name":
"batch_normalization_2", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones", "config": {}},
"beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null, "gamma_regularizer": null,
"beta_regularizer": null, "momentum": 0.99, "center": true}}, {"class_name": "Conv2D", "config": {"kernel_constraint": null,
"kernel_initializer": {"class_name": "VarianceScaling", "config": {"distribution": "uniform", "scale": 1.0, "seed": null, "mode": "fan_avg"}},
"name": "conv2d_4", "bias_regularizer": null, "bias_constraint": null, "activation": "relu", "trainable": true, "data_format": "channels_last",
"padding": "same", "strides": [1, 1], "dilation_rate": [1, 1], "kernel_regularizer": null, "filters": 128, "bias_initializer": {"class_name":
"Zeros", "config": {}}, "use_bias": true, "activity_regularizer": null, "kernel_size": [3, 3]}}, {"class_name": "BatchNormalization", "config":
{"beta_constraint": null, "gamma_initializer": {"class_name": "Ones", "config": {}}, "moving_mean_initializer": {"class_name": "Zeros",
"config": {}}, "name": "batch_normalization_3", "epsilon": 0.001, "trainable": true, "moving_variance_initializer": {"class_name": "Ones",
"config": {}}, "beta_initializer": {"class_name": "Zeros", "config": {}}, "scale": true, "axis": -1, "gamma_constraint": null,
"gamma_regularizer": null, "beta_regularizer": null, "momentum": 0.99, "center": true}}, {"class_name": "MaxPooling2D", "config": {"name":

Ln 1, Col 1 Spaces: 4 UTF-8 CRLF

Step 7: Accuracy Calculation :

Load the test files

```
truey=[]
predy=[]
x = np.load('./modXtest.npy')
y = np.load('./modytest.npy')

yhat= loaded_model.predict(x)
yh = yhat.tolist()
yt = y.tolist()
count = 0

for i in range(len(y)):
    yy = max(yh[i])
    yyt = max(yt[i])
    predy.append(yh[i].index(yy))
    truey.append(yt[i].index(yyt))
    if(yh[i].index(yy)== yt[i].index(yyt)):
        count+=1

acc = (count/len(y))*100
```

Accuracy Calculation

Step 8: Saving the values for confusion matrix analysis

```python
#saving values for confusion matrix and analysis
np.save('truey', truey)
np.save('predy', predy)
print("Predicted and true label values saved")
print("Accuracy on test set :"+str(acc)+"%")
```

Confusion Matrix

- Also known as error matrix, it is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.
- Each row of the matrix represents the instances in a predicted class, while each column represents the instances in an actual class

Creating Confusion Matrix

```
import itertools
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix

y_true = np.load('./truey.npy')
y_pred = np.load('./predy.npy')
cm = confusion_matrix(y_true, y_pred)
labels = ['Angry', 'Disgust', 'Fear', 'Happy', 'Sad', 'Surprise', 'Neutral']
title='Confusion matrix'
print(cm)
```

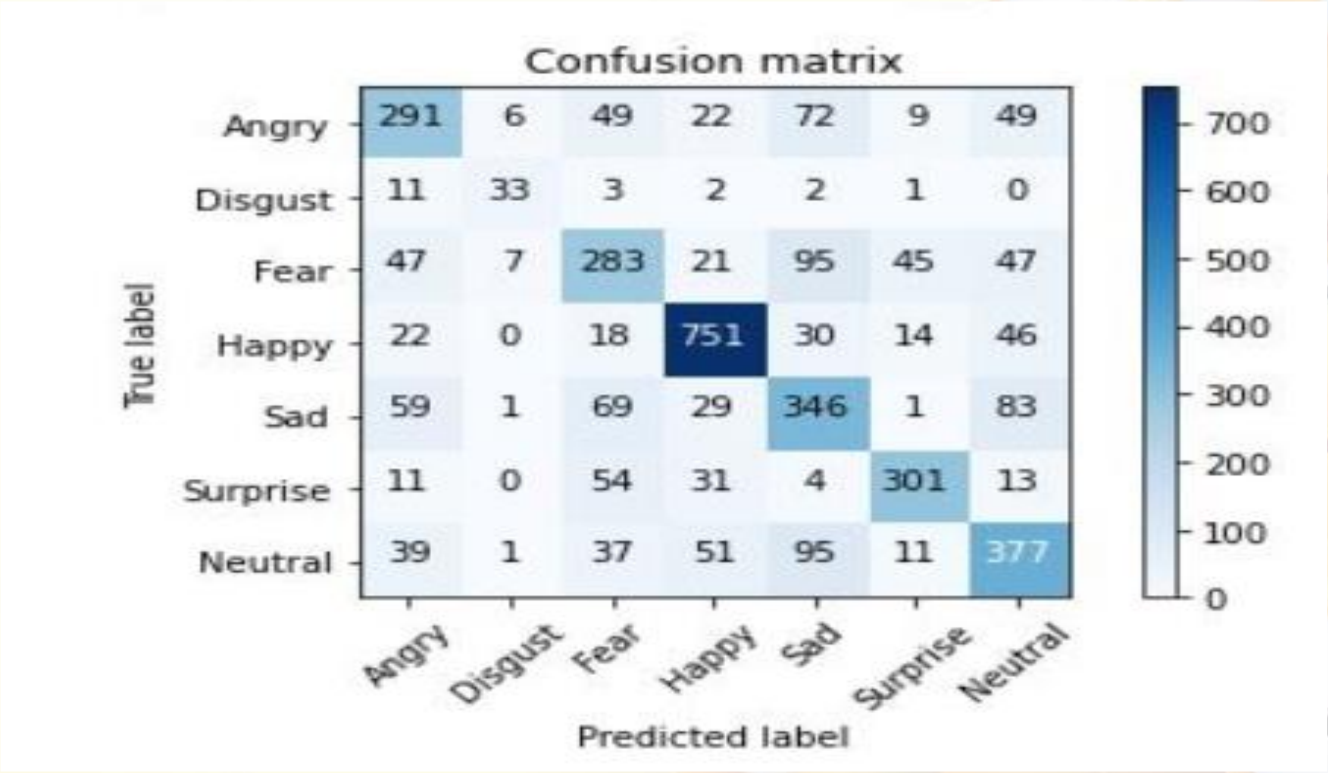
```
In [4]: runfile('E:/Facial Expression Recognition/confm
Recognition')
[[291   6  49  22  72   9  49]
 [ 11  33   3   2   2   1   0]
 [ 47   7 283  21  95  45  47]
 [ 22   0  18 751  30  14  46]
 [ 59   1  69  29 346   1  83]
 [ 11   0  54  31   4 301  13]
 [ 39   1  37  51  95  11 377]]
```

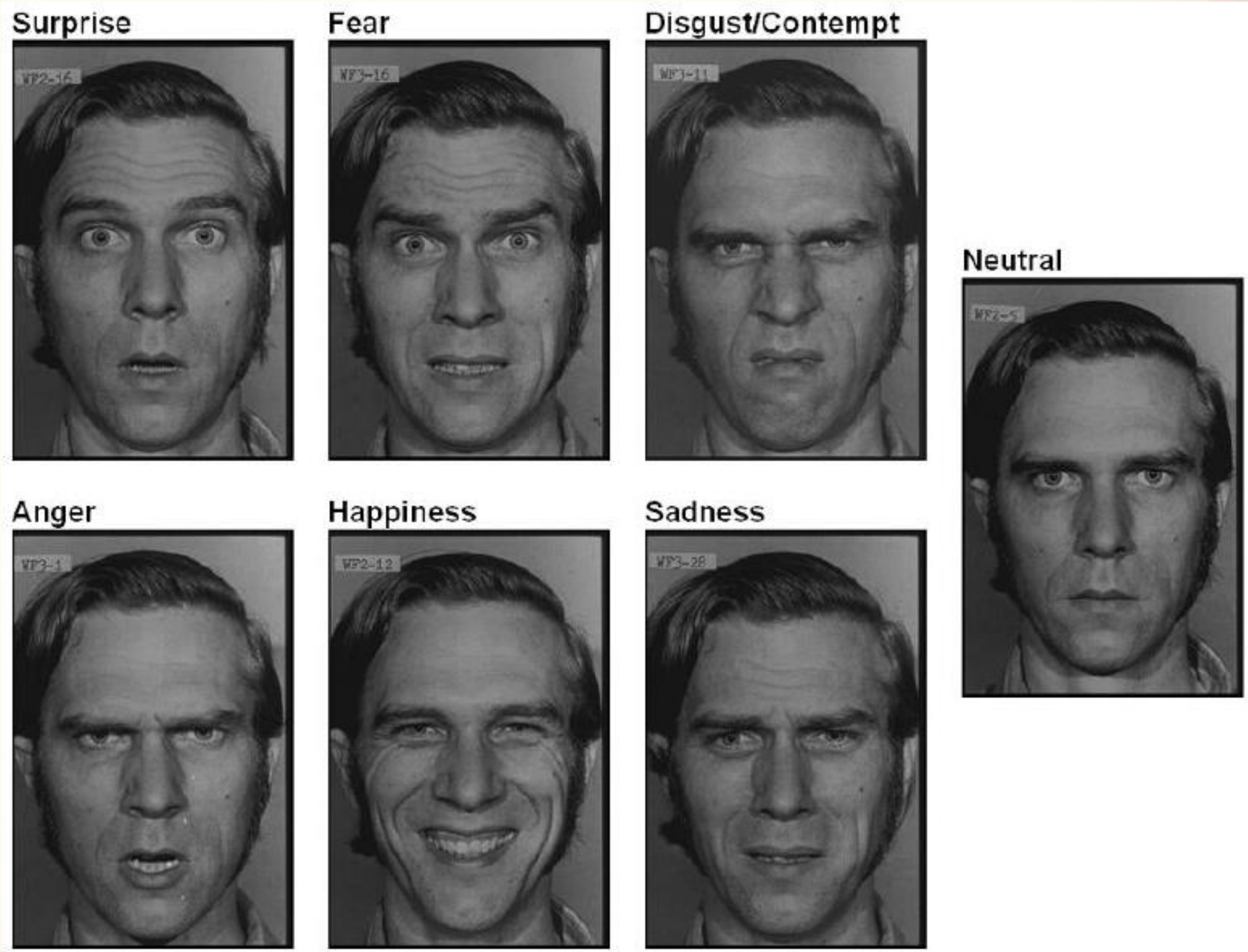
Confusion Matrix

Plotting the confusion Matrix Using matplotlib for better visualisation

```
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title(title)
plt.colorbar()
tick_marks = np.arange(len(labels))
plt.xticks(tick_marks, labels, rotation=45)
plt.yticks(tick_marks, labels)
fmt = 'd'
thresh = cm.max() / 2.
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, format(cm[i, j], fmt),
             horizontalalignment="center",
             color="white" if cm[i, j] > thresh else "black")

plt.ylabel('True label')
plt.xlabel('Predicted label')
plt.tight_layout()
plt.show()
```


Confusion matrix
True label
Predicted label
Angry
Disgust
Fear
Happy
Sad
Surprise
Neutral
291 6 49 22 72 9 49
11 33 3 2 2 1 0
47 7 283 21 95 45 47
22 0 18 751 30 14 46
59 1 69 29 346 1 83
11 0 54 31 4 301 13
39 1 37 51 95 11 377
700
600
500
400
300
200
100
0


Surprise
Fear
Disgust/Contempt
Neutral
Anger
Happiness
Sadness

Recognize expression using Picture

```python
detection_model_path = 'haarcascade_frontalface_default.xml'
image_path = 'test_for_image.jpg'

face_detection = cv2.CascadeClassifier(detection_model_path)

emotion_classifier = load_model("model.hdf5")

emotions = ['angry', 'disgust', 'scared', 'happy', 'sad', 'surprised', 'neutral']

color_frame = cv2.imread(image_path)
gray_frame = cv2.imread(image_path, 0)


cv2.imshow('Input test image', color_frame)
cv2.waitKey(1000)
cv2.destroyAllWindows()


detected_faces = face_detection.detectMultiScale(color_frame, scaleFactor=1.1, minNeighbors=5,
                                        minSize=(30,30), flags=cv2.CASCADE_SCALE_IMAGE)
print('Number of faces detected : ', len(detected_faces))

```

```
    if len(detected_faces)>0:

        detected_faces = sorted(detected_faces, reverse=True, key=lambda x: (x[2]-x[0])*(x[3]-x[1]))[0] #
        (fx, fy, fw, fh) = detected_faces

        im = gray_frame[fy:fy+fh, fx:fx+fw]
        im = cv2.resize(im, (48,48))  # the model is trained on 48*48 pixel image
        im = im.astype("float")/255.0
        im = img_to_array(im)
        im = np.expand_dims(im, axis=0)

        preds = emotion_classifier.predict(im)[0]
        emotion_probability = np.max(preds)
        label = emotions[preds.argmax()]

        cv2.putText(color_frame, label, (fx, fy-10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
        cv2.rectangle(color_frame, (fx, fy), (fx + fw, fy + fh),(0, 0, 255), 2)

    cv2.imshow('Input test image', color_frame)
    cv2.imwrite('output_'+image_path.split('/')[-1], color_frame)
    cv2.waitKey(10000)
    cv2.destroyAllWindows()

    import matplotlib.image as mpimg
    img = mpimg.imread('output_for_image.jpg')
    imgplot = plt.imshow(img)
    plt.show()
```

Clicking Image from user and recognize expression

```
videoCaptureObject = cv2.VideoCapture(0)
result=True
while(result):
    ret,frame=videoCaptureObject.read()
    cv2.imwrite("test_for_camera.jpg",frame)
    result=False
videoCaptureObject.release()
cv2.destroyAllWindows()


detection_model_path = 'haarcascade_frontalface_default.xml'
image_path = 'test_for_camera.jpg'

face_detection = cv2.CascadeClassifier(detection_model_path)

emotion_classifier = load_model("model.hdf5")

emotions = ['angry', 'disgust', 'scared', 'happy', 'sad', 'surprised', 'neutral']

color_frame = cv2.imread(image_path)
gray_frame = cv2.imread(image_path, 0)


cv2.imshow('Input test image', color_frame)
cv2.waitKey(10000)
cv2.destroyAllWindows()


detected_faces = face_detection.detectMultiScale(color_frame, scaleFactor=1.1, minNeighbors=5,
                                        minSize=(30,30), flags=cv2.CASCADE_SCALE_IMAGE)
print('Number of faces detected : ', len(detected_faces))
```

Change from previous code

```python
if len(detected_faces)>0:

    detected_faces = sorted(detected_faces, reverse=True, key=lambda x: (x[2]-x[0])*(x[3]-x[1]))[0] #
    (fx, fy, fw, fh) = detected_faces

    im = gray_frame[fy:fy+fh, fx:fx+fw]
    im = cv2.resize(im, (48,48))  # the model is trained on 48*48 pixel image
    im = im.astype("float")/255.0
    im = img_to_array(im)
    im = np.expand_dims(im, axis=0)

    preds = emotion_classifier.predict(im)[0]
    emotion_probability = np.max(preds)
    label = emotions[preds.argmax()]

    cv2.putText(color_frame, label, (fx, fy-10), cv2.FONT_HERSHEY_SIMPLEX, 0.45, (0, 0, 255), 2)
    cv2.rectangle(color_frame, (fx, fy), (fx + fw, fy + fh),(0, 0, 255), 2)

cv2.imshow('Input test image', color_frame)
cv2.imwrite('output_'+image_path.split('/')[-1], color_frame)
cv2.waitKey(10000)
cv2.destroyAllWindows()

import matplotlib.image as mpimg
img = mpimg.imread('output_for_camera.jpg')
imgplot = plt.imshow(img)
plt.show()
```

REFERENCES

- *Shan, C., Gong, S., & McOwan, P. W. (2005, September). Robust facial expression recognition using local binary patterns. In Image Processing, 2005. ICIP 2005. IEEE International Conference on (Vol. 2, pp. II-370). IEEE.*

- *Chibelushi, C. C., & Bourel, F. (2003). Facial expression recognition: A brief tutorial overview. CVonline: On-Line Compendium of Computer Vision, 9.*

- *"Convolutional Neural Networks (LeNet) – DeepLearning 0.1 documentation". DeepLearning 0.1. LISA Lab. Retrieved 31 August 2013.*

THANK YOU!