AIDI-1002-01-AI ALGORITHM I

Human Facial/Expression Recognition

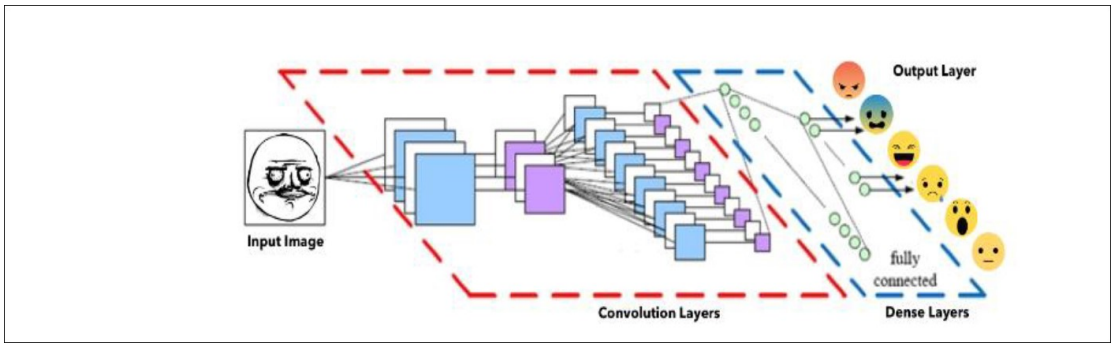
Model Architecture:

Deep learning is a common computer vision approach.

Convolutional Neural Network (CNN) layers were used as building blocks for our model architecture.

A convolutional neural network's usual architecture includes an input layer, some convolutional layers, some dense layers (also known as fully-connected layers), and an output layer.

These are linearly stacked layers ordered in sequence. In Keras, the model is created as Sequential() and more layers are added to build architecture.



Input:

Since the input layer's dimensions are pre-determined and fixed, the image must be pre-processed before being fed into it.

After successfully loading the training, validation, and test datasets. Since there are only grayscale images instead of color images stored in RGB format, there are 6 tags, so one channel is used.

For the training dataset, the sample size was 28709, whereas for the validation and test datasets, the sample size was 3589. Overall, we had a fairly large data set.

Convolutional:

The numpy array is sent to the Convolution2D layer, where the number of filters is specified as one of the hyperparameters.

With randomly generated weights, the set of filters (also known as the kernel) is unique.

To construct a feature map, each filter, (3, 3) receptive field, slides across the source image with shared weights.

For example, edge and pattern detection, convolution provides feature maps that represent how pixel values are enhanced.

Filter 1 is applied across the entire image to build a feature map.

Other filters are applied one by one, resulting in a series of feature maps.

Dense:

The dense layer, also known as completely connected layers) is modelled after the way neurons transmit impulses in the brain.

It accepts a large number of input features and transforms them using trainable weights and layers.

Each layer's weight had to be adjusted previously.
By adjusting hyper-parameters like as learning rate and network density, we can regulate the training speed and complexity of the design.

As additional data is fed into the network, it is able to make progressive modifications until errors are minimised.

In other words, the more layers/nodes we add to the network, the better it will be in picking up signals.

As appealing as it may appear, the model is increasingly prone to overfitting the training data.

Output:

Our facial expressions are typically far more complex, containing a variety of emotions that might be used to effectively characterise a specific expression.

To begin, we created a basic CNN with an input, three convolution layers, one dense layer, and one output layer. The simplistic model, as it turned out, performed miserably. The poor accuracy of 0.1500 indicated that it was simply a matter of picking one of the six emotions at random.

Face expressions with nuanced features were missed by the simplistic net design. Only one thing could be the case.

Deep learning comes into play here. Because of the pattern complexity of facial expressions, it's important to use a more complicated architecture to detect tiny signals.

Here, we approached this project first, clearly delineating our main goal of classifying a given grayscale image into one of six labels.

The image classification problem naturally leads us to the well-known Convolutional Neural Network (CNN).

After investigating how CNNs work for grayscale images in general, we decided to start with 3 successive convolutional layers with the same fill pattern as the normal activation function for the "relu" convolutional layer, then the max pooling layer. Then we added more and more convolutional layers using various captured features on the underlying structure.

Finally, the folded layer is first flattened and then goes through the two denser layers to reach the output layer.

Here, the softmax activation function is used for multi-class classification (6 classes in total).

In [*]:
```
# Final Model Architecture:
from keras import layers
from keras import models
from keras import optimizers

modelN = models.Sequential()
modelN.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu',
                         input_shape=(48, 48, 1)))
modelN.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
modelN.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu'))
modelN.add(layers.MaxPooling2D(pool_size=(2, 2)))

modelN.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
modelN.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
modelN.add(layers.Conv2D(64, (3, 3), padding='same', activation='relu'))
modelN.add(layers.MaxPooling2D(pool_size=(2, 2)))

modelN.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
modelN.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
modelN.add(layers.Conv2D(128, (3, 3), padding='same', activation='relu'))
modelN.add(layers.MaxPooling2D(pool_size=(2, 2)))

modelN.add(layers.Flatten())  # this converts our 3D feature maps to 1D feature vectors
modelN.add(layers.Dense(64, activation='relu'))
modelN.add(layers.Dense(64, activation='relu'))
modelN.add(layers.Dense(6, activation='softmax'))

# optimizer:
modelN.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print ('Training....')

modelF = modelN.fit(X_train, y_train,
          validation_data=(X_val, y_val), shuffle=True, verbose=1)
```

```
Training....
536/898 [================>.............] - ETA: 3:11 - loss: 1.7665 - accuracy: 0.2470
```