**Downloading Images**:

We have read the web link from the excel file by column and sent the request in the server and got the image and saved it in the folder as a jpg format. If there have no image then the system omitted the link and tried the next one. This process continued until the end of the websites in the column.

**Cropping Face:**

We have write a function to detect the multiple faces from image and after cropping the data we resize the face image and reshape this in one dimensional data before saving the data in the csv format. We saved this one dimensional data in a csv format to reduce the uses of the memory. We have used frontal face detector from to detect the faces and crop the faces in a fixed length.

We have resize face image using their height and width.

**Extracting Pixel Information:**

We have read the images from the folder and after identified faces we crop the faces then resize them as 100 x 100 and transform as an array. After seeing the faces we changed the color as grayscale and then reshape the pixel info as 1 dimensional array. Then we save this array in a csv file. This process will continue until the end of the images in the folder.

**Creating Model:**

After reading data, we need to reshape the data (only the pixel information data of grayscale image) set as follows (length of data x 100 x 100 x 1). We can change this 100 x 100 by any dimensions but need to keep in mind that depending on this dimension we will test our data in future and that’s why we need to convert test data in the same dimension. Last 1 means the image is in the Grayscale. If the size of image is lower than our model will be too fast.

We need to separate our label data and need to categorize into their classes. Then we need to transform both data sets as an array. Then need to split the data into train and test.

The model type that we will be using is Sequential. Sequential is the easiest way to build a model in Keras. It allows building a model layer by layer. We use the add() function to add layers to our model. Our first layers are Conv2D layers. These are convolution layers that will deal with our input images, which are seen as 2-dimensional matrices.

We have use 64 in the first layer, 32 in the second layer, 16 in the third, 8 in the fourth and 4 in the fifth layer are the number of nodes in each layer. This number can be adjusted to be higher or lower, depending on the size of the dataset. Kernel size of 3 means we will have a 3x3 filter matrix. Activation is the activation function for the layer. The activation function we have use for our 5 Convolution2D layers is the ReLU, or Rectified Linear Activation. Our first layer takes in an input shape of each input image, 100 x 100 x 1.

Pooling layers are used to down sample the image. Image contain a lot of pixel values and it is typically easy for the network to learn the features if the image size is being reduced. Pooling layers help in reducing the number of parameters required and hence, this reduces the computation required. Pooling also helps in avoiding overfitting. The MaxPooling2D layer consists of pooling size of 3×3 and stride 2. In between the Convolution2D layers and the dense layer, there is a Flatten layer which serves as a connection between the convolution and dense layers. So, we have connected our convolution and dense layers.

Dense is the layer type we have use in for our output layer. Dense is a standard layer type that is used in many cases for neural networks. Softmax makes the output sum up to 1 so the output can be interpreted as probability that’s why we have softmax. The model will then make its prediction based on which option has the highest probability.

The optimizer controls the learning rate. We have use adadelta as our optimizer although there have another one called adam. Adam is generally a good optimizer to use for many cases. The optimizer adjusts the learning rate throughout training. The learning rate determines how fast the optimal weights for the model are calculated. A smaller learning rate may lead to more accurate weights (up to a certain point), but the time it takes to compute the weights will be longer. We have use categorical\_crossentropy for our loss function. This is the most common choice for classification. A lower score indicates that the model is performing better.

**Size of Tensors:**

Size of the Output Tensor (Image) of a Conv Layer

Let’s define

O = Size (width) of output image.  
I = Size (width) of input image.  
K = Size (width) of kernels used in the Conv Layer.  
N = Number of kernels.  
S = Stride of the convolution operation.  
P = Padding.

The size (O) of the output image is given by,

The number of channels in the output image is equal to the number of kernels N.

## Size of Output Tensor (Image) of a MaxPool Layer:

Let’s define

O = Size (width) of output image.  
I = Size (width) of input image.  
S = Stride of the convolution operation.  
P = Pool size.

The size (O) of the output image is given by,

Note that this can be obtained using the formula for the convolution layer by making padding equal to zero and keeping same as the kernel size. But unlike the convolution layer, the number of channels in the Maxpool layer’s output is unchanged.

**Reusable Model Weights:**

JSON is a simple file format for describing data hierarchically. Keras provides the ability to describe any model using JSON format with a *to\_json()* function. This can be saved to file and later loaded via the *model\_from\_json()* function that will create a new model from the JSON specification. The weights are saved directly from the model using the *save\_weights()* function and later loaded using the symmetrical *load\_weights()* function.

The model and weight data is loaded from the saved files and a new model is created. It is important to compile the loaded model before it is used. This is so that predictions made using the model can use the appropriate efficient computation from the Keras backend.