Drishti.AI Intern Project

Submitted by - Paluvadi Surya Vamsi

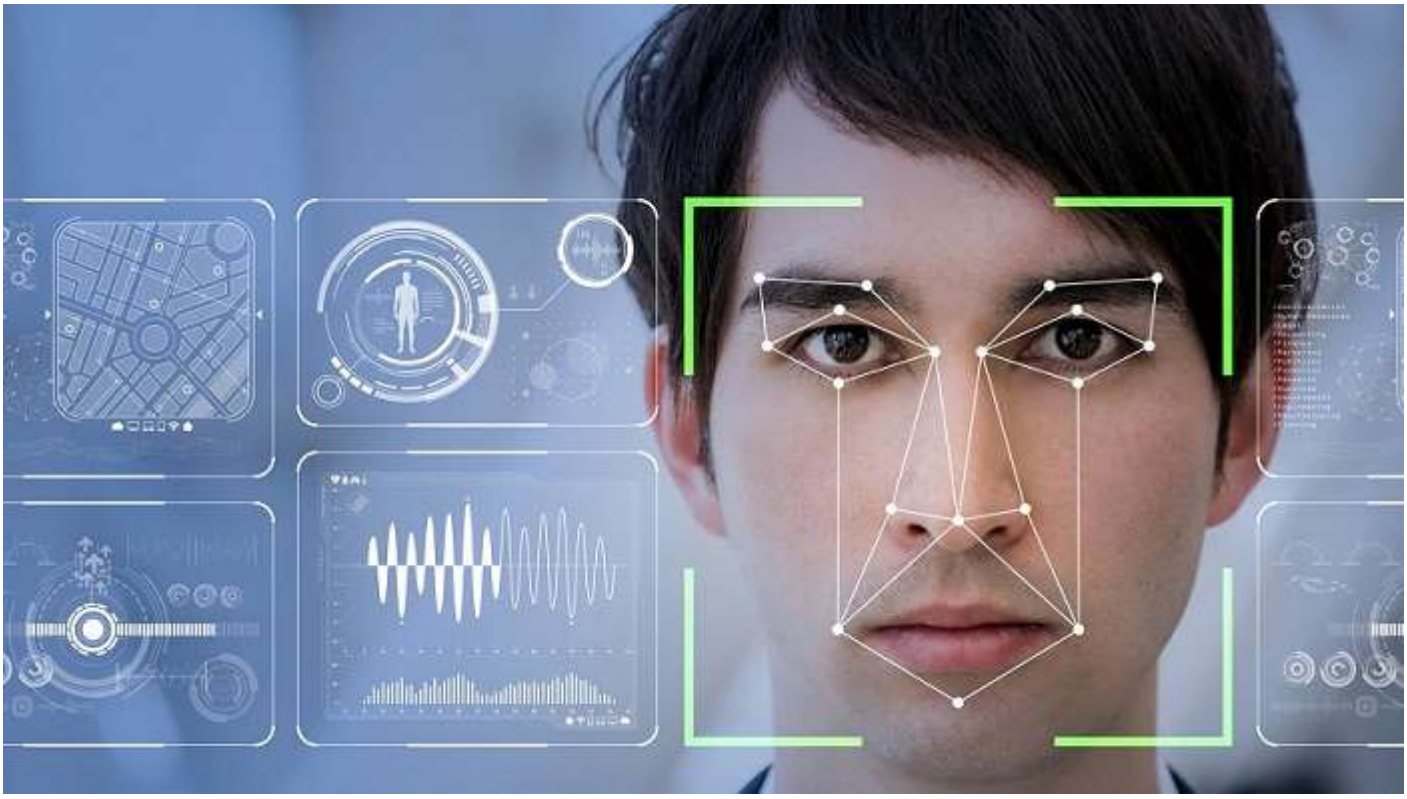
- Contact No : +91-81100 20188
- Email ID: paluvadisurya@gmail.com

AIM

- Create a Deep Learning model to estimate character in Image

Provided Information

- 4 Types of character images are subdivided into a folder for training the model
- 4 Types of labelled characteristic images are provided in a folder for testing them with the model



Importing Basic Standard Libraries

In [1]:
```
import warnings
warnings.filterwarnings('ignore') # To avoid annoying update notices from libr
aries
import numpy as np # For strong mathematical operations
import pandas as pd # For dataframe analysis & creating dummy variables
import matplotlib.pyplot as plt # for visualization imports
import seaborn as sns # For beautiful visualizations
# To make all plt plots visible inside the command code without using plt.show
()
%matplotlib inline
```

01-- Looking at Data

Training Data

In [2]:
```
# Libraries for importing the files from our directories
import glob
import PIL # To extract Image Data
from PIL import Image
```

In [3]:
```
directory = "Train/"

# Let's look at the folders available inside the training directory
glob.glob(directory+'*/')
```

Out[3]:
```
['Train\\A\\', 'Train\\B\\', 'Train\\C\\', 'Train\\D\\']
```

In [4]:
```
# Store all the directories in a variable for future use

char_dir = glob.glob(directory+'*/') # Character Directory

print('#'*75)
print("{} Types of labelled categorical images are available in our training d
ataset".format(len(char_dir)))
print('#'*75)
```

```
###########################################################################
4 Types of labelled categorical images are available in our training dataset
###########################################################################
```

In [5]:
```
# Lets look at the no of image of the images available inside the character di
rectories
images_dir = []
print('-'*50)
for i in char_dir:
    images_dir = glob.glob(i+'/*jpg')
    print("{} Images of character {} available in our directory".format(len(im
ages_dir),i.split('\\')[-2]))
    print('-'*50)
```

```
--------------------------------------------------
902 Images of character A available in our directory
--------------------------------------------------
1441 Images of character B available in our directory
--------------------------------------------------
1331 Images of character C available in our directory
--------------------------------------------------
612 Images of character D available in our directory
--------------------------------------------------
```

Testing Data

In [6]:
```
directory = "Test/"

# Let's look at the folders available inside the training directory
glob.glob(directory+'*/')
```

Out[6]:
```
['Test\\A\\', 'Test\\B\\', 'Test\\C\\', 'Test\\D\\']
```

In [7]:
```
# Store all the directories in a variable for future use

test_dir = glob.glob(directory+'*/') # Character Directory

print('#'*75)
print("{} Types of labelled categorical images are available in our testing da
taset".format(len(char_dir)))
print('#'*75)
```

```
###########################################################################
4 Types of labelled categorical images are available in our testing dataset
###########################################################################
```

In [8]:
```
# Lets look at the no of image of the images available inside the character di
rectories
images_dir = []
print('-'*50)
for i in test_dir:
    images_dir = glob.glob(i+'/*jpg')
    print("{} Images of character {} available for testing".format(len(images_
dir),i.split('\\')[-2]))
    print('-'*50)
```

```
--------------------------------------------------
11 Images of character A available for testing
--------------------------------------------------
11 Images of character B available for testing
--------------------------------------------------
11 Images of character C available for testing
--------------------------------------------------
11 Images of character D available for testing
--------------------------------------------------
```

In [9]:
```
# Let's look at some images inside the directories
random_image = PIL.Image.open('Train/A/pic_0011.jpg')
print(random_image.size)
random_image
```

```
(288, 416)
```

Out[9]:

In [10]:
```
random_image1 = PIL.Image.open('Train/B/pic_0011.jpg')
print(random_image1.size)
random_image1
```

```
(340, 256)
```

Out[10]:

In [11]:
```
random_image2 = PIL.Image.open('Train/C/pic_0011.jpg')
print(random_image2.size)
random_image2
```

```
(544, 432)
```

Out[11]:

02 -- Resizing Images

We can clearly say that images are not symmetrical in resolution

- Let's resize all the images to make machine understand easily

Procedure for resizing

- Create a definition to obtain the inputs from user to resize into a specific size
  - Steps for creating a function to resize all training & testing data
    - Resize all our images into a square resolution
    - Apply best resampling method while resizing the images to make the pixels smooth
    - Convert resized images into an array using numpy function
    - Obtain image label & image data, store them in a variable

`Image.resize`(*size, resample=0*)

Returns a resized copy of this image.

Parameters:

- **size** – The requested size in pixels, as a 2-tuple: (width, height).
- **resample** – An optional resampling filter. This can be one of `PIL.Image.NEAREST` (use nearest neighbour), `PIL.Image.BILINEAR` (linear interpolation), `PIL.Image.BICUBIC` (cubic spline interpolation), or `PIL.Image.LANCZOS` (a high-quality downsampling filter). If omitted, or if the image has mode "1" or "P", it is set `PIL.Image.NEAREST`.

Returns: An `Image` object.

In [12]:
```
# Location of Our Training Character Images
train_loc = "Train/"

# Looping through each directory roots for searching *jpg files (We have given
JPG files)
# Convert Images into Array & Resize the images at same time
def Train_img_Resize(directory, length, width):
    train_label = []
    train_image = []
    project_direc = glob.glob(train_loc+"*/")
    for direc in project_direc:
        # Extract all the JPG extensions from the directories using Glob funct
ion
        image_filename = glob.glob(direc+"/*jpg")
        for file in image_filename:
            #Resize and converting images into an array
            # Length and width can be changes & will see by iterating them lat
er
            img = np.array(Image.open(file).resize([length,width],resample = P
IL.Image.BILINEAR))
            train_image.append(img)
            # Extracting the label from the name of the folder that obtained b
y glob function earlier
            label = direc.split("\\")[-2] # Splitting is clearly observed in t
he top lines of code
            train_label.append(label)

    #Convert the images and labels to numpy arrays
    train_label = np.array(train_label)
    train_image = np.array(train_image)
    return train_image, train_label
```

In [13]:
```
train_image, train_label = Train_img_Resize(directory= train_loc, length = 150
, width = 150)
```

Let's do the same resizing for the testing data also

In [14]:
```
# Location of Our Training Character Images
test_loc = "Test/"
def Test_img_Resize(directory, length, width):
    pre_defined_label = []
    test_image = []
    project_direc = glob.glob(test_loc+"*/")
    for direc in project_direc:
        image_filename = glob.glob(direc+"/*jpg")
        for file in image_filename:
            img = np.array(Image.open(file).resize([length,width],resample = P
IL.Image.BILINEAR))
            test_image.append(img)
            label = direc.split("\\")[-2]
            pre_defined_label.append(label)
    pre_defined_label = np.array(pre_defined_label)
    test_image = np.array(test_image)
    return test_image, pre_defined_label
```

In [15]:
```
test_image, pre_defined_label = Test_img_Resize(directory=test_loc, length=150
, width=150)
```

In [16]:
```
# Let's look at our data in numericals which are converted from Images

train_image.shape, train_label.shape, test_image.shape, pre_defined_label.shap
e
```

Out[16]:
```
((4286, 150, 150, 3), (4286,), (44, 150, 150, 3), (44,))
```

The above shape defines

(4286, 150, 150, 3)

- 4286 ----------> No of Images
- 150 ----------> No of horizontal pixels
- 150 ----------> No of vertical pixels
- 3 ----------> No of color layers (RGB)

03 -- Data Preparation

In [17]:
```
import sklearn # For machine learning data pre-processing
from sklearn.utils import class_weight
from sklearn.model_selection import train_test_split
# To validate our model by splitting the training data into training & vadilat
ion data
from sklearn.metrics import confusion_matrix, accuracy_score
```

In [18]:
```
X = train_image
y = train_label
```

In [19]:
```
X.shape, y.shape
```

Out[19]:
```
((4286, 150, 150, 3), (4286,))
```

In [20]:
```
# Let's normalize the training & testing Image Data dividing it by 255

X = X/255

X_test = test_image/255
```

In [21]:
```
# Let's look our label data
label_df = pd.DataFrame(data=train_label, columns=['Charachter_ID'])
label_df['Charachter_ID'].value_counts()
```

Out[21]:
```
B    1441
C    1331
A     902
D     612
Name: Charachter_ID, dtype: int64
```

In [22]:
```
# Splitting data into train & validation data

X_train , X_validate, y_train, y_validate = train_test_split(X,y, test_size =
0.2, random_state = 25)
```

In [23]:
```
X_test.shape, pre_defined_label.shape
```

Out[23]:
```
((44, 150, 150, 3), (44,))
```

In [24]:
```
X_train.shape, X_validate.shape, y_train.shape, y_validate.shape
```

Out[24]:
```
((3428, 150, 150, 3), (858, 150, 150, 3), (3428,), (858,))
```

- Before training our data to a model let's obtain a balanced weights of each character

In [25]:
```
class_weights = class_weight.compute_class_weight(class_weight='balanced', y =
y_train, classes= np.unique(train_label))
```

In [26]:
```
class_weights
```

Out[26]:
```
array([1.14572193, 0.74912587, 0.80925401, 1.7966457 ])
```

In [27]:
```
# We can see that the values of train_labels are in categorical form
# Lets encode these values into 0 & 1 to make our machine understand the class
es easily

y_train_onehot = pd.get_dummies(y_train).values
y_validate_onehot = pd.get_dummies(y_validate).values
pre_defined_label_onehot = pd.get_dummies(pre_defined_label).values
```

Label Encoding

Food Name	Categorical #	Calories
Apple	1	95
Chicken	2	231
Broccoli	3	50

→

One Hot Encoding

Apple	Chicken	Broccoli	Calories
1	0	0	95
0	1	0	231
0	0	1	50

In [28]:
```
# Creating a reverse lookup for the encoded data for future purpose
reverse_lookup_train = pd.get_dummies(y_train).columns.values.tolist()
reverse_lookup_test = pd.get_dummies(pre_defined_label).columns.values.tolist
()
```

In [29]:
```
reverse_lookup_test
```

Out[29]:
```
['A', 'B', 'C', 'D']
```

In [30]:
```
reverse_lookup_train
```

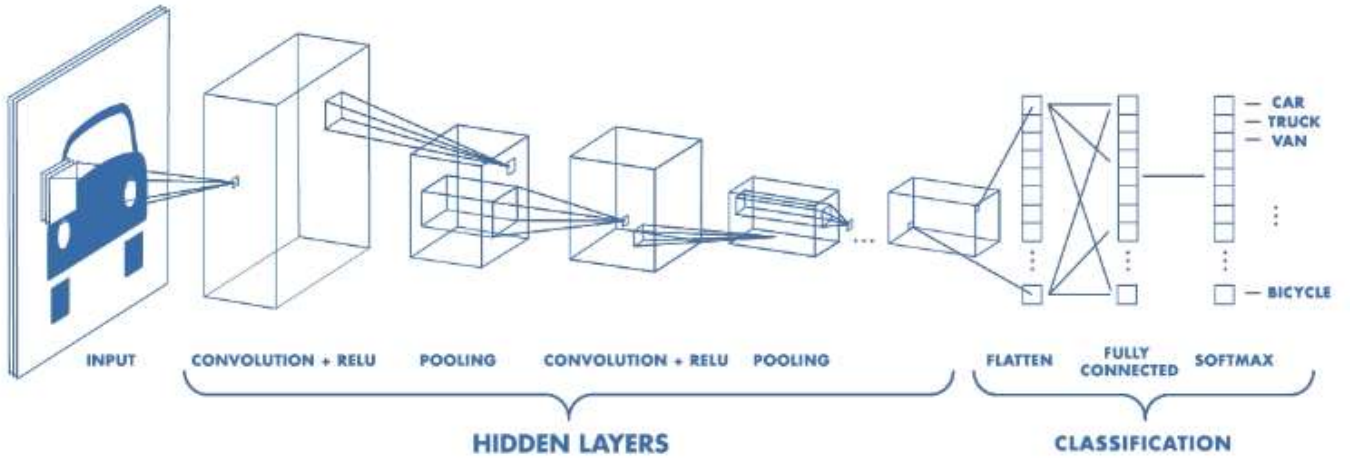
Out[30]:
```
['A', 'B', 'C', 'D']
```

---

04 -- Deep Learning Model Building

In [31]:
```
import keras # Deep Learning Library
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
```

```
Using TensorFlow backend.
```



In [32]:
```
# Let's create a model using a function with flexible inputs

def Model(X_train,y_train):
    #The input and output shape of our model
    input_shape = X_train[0].shape
    output_shape = len(y_train[0])

    #The CNN model
    model = Sequential()
    model.add(Conv2D(32, kernel_size=(5, 5), strides=(1, 1),activation='relu',
input_shape=input_shape))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
    model.add(Conv2D(64, (5, 5), 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(Dense(64, activation='relu'))
    model.add(Dense(output_shape, activation='softmax'))
    model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['a
ccuracy'])
    return model
```

In [33]:
```
#Instantiate Deep Learning Model with our X_train & Y_onehot coded data

DL_Model = Model(X_train, y_train_onehot)
```

```
WARNING:tensorflow:From C:\Users\Gold\Anaconda3\lib\site-packages\tensorflow
\python\framework\op_def_library.py:263: colocate_with (from tensorflow.pytho
n.framework.ops) is deprecated and will be removed in a future version.
Instructions for updating:
Colocations handled automatically by placer.
```

In [34]:
```
# Summary of our newly created model withe no.of layers
DL_Model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 146, 146, 32)      2432
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 73, 73, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 69, 69, 64)        51264
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 34, 34, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 32, 32, 64)        36928
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 16, 16, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 16384)             0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               2097280
_________________________________________________________________
dense_2 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_3 (Dense)              (None, 4)                 260
=================================================================
Total params: 2,196,420
Trainable params: 2,196,420
Non-trainable params: 0
_________________________________________________________________
```

In [35]:
```
# Let's fit the model with our training data & test the accuracy of model usin
g validation data

DL_Model.fit(X_train,
             y_train_onehot,
             epochs=20,
             verbose=1,
             batch_size=64,
             validation_data=[X_validate,y_validate_onehot])
```

```
WARNING:tensorflow:From C:\Users\Gold\Anaconda3\lib\site-packages\tensorflow
\python\ops\math_ops.py:3066: to_int32 (from tensorflow.python.ops.math_ops)
is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.cast instead.
Train on 3428 samples, validate on 858 samples
Epoch 1/20
3428/3428 [==============================] - 102s 30ms/step - loss: 1.2593 -
acc: 0.4414 - val_loss: 1.0111 - val_acc: 0.5699
Epoch 2/20
3428/3428 [==============================] - 101s 30ms/step - loss: 0.8262 -
acc: 0.6689 - val_loss: 0.7550 - val_acc: 0.7179
Epoch 3/20
3428/3428 [==============================] - 103s 30ms/step - loss: 0.5654 -
acc: 0.7827 - val_loss: 0.5775 - val_acc: 0.7786
Epoch 4/20
3428/3428 [==============================] - 103s 30ms/step - loss: 0.4407 -
acc: 0.8285 - val_loss: 0.4581 - val_acc: 0.8392
Epoch 5/20
3428/3428 [==============================] - 102s 30ms/step - loss: 0.3332 -
acc: 0.8766 - val_loss: 0.4205 - val_acc: 0.8636
Epoch 6/20
3428/3428 [==============================] - 99s 29ms/step - loss: 0.2509 - a
cc: 0.9113 - val_loss: 0.4035 - val_acc: 0.8683
Epoch 7/20
3428/3428 [==============================] - 100s 29ms/step - loss: 0.1604 -
acc: 0.9446 - val_loss: 0.5407 - val_acc: 0.8357
Epoch 8/20
3428/3428 [==============================] - 102s 30ms/step - loss: 0.1344 -
acc: 0.9516 - val_loss: 0.5370 - val_acc: 0.8718
Epoch 9/20
3428/3428 [==============================] - 101s 29ms/step - loss: 0.0650 -
acc: 0.9790 - val_loss: 0.5161 - val_acc: 0.8811
Epoch 10/20
3428/3428 [==============================] - 102s 30ms/step - loss: 0.0818 -
acc: 0.9726 - val_loss: 0.5977 - val_acc: 0.8543
Epoch 11/20
3428/3428 [==============================] - 97s 28ms/step - loss: 0.0341 - a
cc: 0.9898 - val_loss: 0.6839 - val_acc: 0.8730
Epoch 12/20
3428/3428 [==============================] - 104s 30ms/step - loss: 0.0506 -
acc: 0.9837 - val_loss: 0.5763 - val_acc: 0.8753
Epoch 13/20
3428/3428 [==============================] - 103s 30ms/step - loss: 0.0347 -
acc: 0.9883 - val_loss: 0.6197 - val_acc: 0.8904
Epoch 14/20
3428/3428 [==============================] - 97s 28ms/step - loss: 0.0147 - a
cc: 0.9953 - val_loss: 0.7069 - val_acc: 0.8800
Epoch 15/20
3428/3428 [==============================] - 100s 29ms/step - loss: 0.0036 -
acc: 1.0000 - val_loss: 0.8029 - val_acc: 0.8893
Epoch 16/20
3428/3428 [==============================] - 103s 30ms/step - loss: 5.8146e-0
4 - acc: 1.0000 - val_loss: 0.8042 - val_acc: 0.8928
Epoch 17/20
3428/3428 [==============================] - 105s 31ms/step - loss: 2.7219e-0
4 - acc: 1.0000 - val_loss: 0.8163 - val_acc: 0.8916
```

```
Epoch 18/20
3428/3428 [==============================] - 95s 28ms/step - loss: 2.1237e-04
- acc: 1.0000 - val_loss: 0.8264 - val_acc: 0.8928
Epoch 19/20
3428/3428 [==============================] - 110s 32ms/step - loss: 1.7086e-0
4 - acc: 1.0000 - val_loss: 0.8325 - val_acc: 0.8951
Epoch 20/20
3428/3428 [==============================] - 108s 32ms/step - loss: 1.4554e-0
4 - acc: 1.0000 - val_loss: 0.8387 - val_acc: 0.8951
```

Out[35]: <keras.callbacks.History at 0x1a6e3c29e48>

05 -- Evaluating the Model

In [36]:
```
#Get the predictions from our model
predicted_label = DL_Model.predict(X_test)
predicted_label
```

Out[36]:
```
array([[9.99999762e-01, 2.39944285e-07, 9.59414592e-09, 1.00864055e-12],
       [9.99377072e-01, 2.93432240e-04, 3.23661050e-04, 5.84979398e-06],
       [8.84858309e-06, 3.41771694e-10, 9.99991179e-01, 3.59232364e-16],
       [4.89431098e-02, 1.43347157e-03, 1.07413740e-03, 9.48549211e-01],
       [9.99997020e-01, 1.13009758e-11, 2.96806252e-06, 9.47105658e-11],
       [9.99903560e-01, 2.66544430e-05, 6.96950156e-05, 6.58089642e-12],
       [1.51101689e-04, 8.34493116e-02, 9.16399539e-01, 2.02169040e-15],
       [1.60743170e-08, 9.99973297e-01, 2.66535044e-05, 2.90458884e-15],
       [4.92067486e-02, 7.94592500e-02, 8.71334016e-01, 1.10005790e-16],
       [1.00000000e+00, 1.59491636e-12, 3.25955547e-08, 5.26524007e-12],
       [1.00000000e+00, 2.41108864e-13, 1.34112041e-10, 1.26589239e-18],
       [1.10533182e-02, 9.88946736e-01, 2.84184477e-16, 3.35666915e-17],
       [1.64154301e-10, 1.00000000e+00, 2.75534498e-14, 3.10453960e-16],
       [4.49765153e-04, 9.99547303e-01, 2.89735135e-06, 1.77642523e-08],
       [1.17828690e-01, 8.82171273e-01, 1.22876571e-07, 1.62098033e-08],
       [3.86072401e-07, 9.99999642e-01, 1.56022306e-09, 7.85908408e-26],
       [2.10982512e-12, 9.99678254e-01, 3.21768923e-04, 4.45590078e-18],
       [5.80252509e-15, 1.00000000e+00, 2.58707182e-16, 2.25252795e-16],
       [6.16570063e-27, 1.00000000e+00, 2.39690811e-23, 1.60315439e-38],
       [1.34333489e-10, 1.00000000e+00, 1.91196565e-16, 1.19493127e-31],
       [1.29674190e-12, 1.00000000e+00, 1.00172304e-09, 6.44382848e-19],
       [6.61104192e-21, 1.00000000e+00, 1.25527250e-14, 5.23016513e-28],
       [1.64239037e-19, 1.63100755e-09, 1.00000000e+00, 6.49626464e-11],
       [1.24943362e-08, 1.06199056e-10, 1.00000000e+00, 1.66345063e-11],
       [4.64043769e-05, 6.32426111e-09, 9.99953508e-01, 9.58227773e-08],
       [3.05492757e-03, 9.21095192e-01, 7.58499131e-02, 2.99683407e-08],
       [3.56851444e-02, 1.86799332e-01, 7.77515531e-01, 3.33792473e-11],
       [9.71527925e-07, 9.02492161e-08, 9.99998808e-01, 1.48951415e-07],
       [1.42217278e-08, 4.67552219e-08, 9.99999881e-01, 2.35647097e-17],
       [1.39792927e-10, 2.28104646e-08, 1.00000000e+00, 2.05443003e-13],
       [6.45888987e-09, 7.74625005e-05, 9.99922514e-01, 4.44648336e-08],
       [2.70044162e-16, 4.79184803e-10, 1.00000000e+00, 3.83023335e-23],
       [3.53496824e-03, 2.44545639e-02, 9.59391356e-01, 1.26192197e-02],
       [9.99686122e-01, 3.02258995e-04, 8.05365062e-07, 1.08797722e-05],
       [4.50917437e-09, 3.84601648e-08, 7.88498369e-08, 9.99999881e-01],
       [7.09993453e-10, 5.20243972e-08, 1.60491495e-10, 1.00000000e+00],
       [1.22402563e-14, 3.78062138e-13, 2.26060193e-09, 1.00000000e+00],
       [5.52678732e-14, 1.42274437e-17, 2.59444582e-15, 1.00000000e+00],
       [7.22587754e-11, 4.66306369e-11, 3.91158683e-04, 9.99608815e-01],
       [2.17352691e-09, 1.77043579e-12, 5.99259791e-12, 1.00000000e+00],
       [2.12483865e-04, 9.70237136e-01, 1.37968918e-05, 2.95366459e-02],
       [5.91602603e-21, 3.96445301e-17, 3.74327609e-12, 1.00000000e+00],
       [6.71814080e-12, 7.33339056e-10, 1.23212494e-15, 1.00000000e+00],
       [6.11958373e-03, 5.18333545e-05, 7.31527672e-10, 9.93828595e-01]],
      dtype=float32)
```

In [37]:
```
#Convert the predictions to a one hot encoded vector
final = np.zeros_like(predicted_label)
final
```

Out[37]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]], dtype=float32)
```

In [38]:
```
final[np.arange(len(predicted_label)), predicted_label.argmax(1)] = 1
final
```

Out[38]:
```
array([[1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 0., 1.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [1., 0., 0., 0.],
       [1., 0., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 1., 0., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [0., 0., 1., 0.],
       [1., 0., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 1., 0., 0.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.],
       [0., 0., 0., 1.]], dtype=float32)
```

In [39]:
```
#Compare the predictions to the test labels to get the Test Classification Acc
uracy
test_acc = accuracy_score(pre_defined_label_onehot,final)
print(test_acc)
```

```
0.8181818181818182
```

In [40]:
```
# let us look at some random images along with thier classes using basic visua
lizations

rows = 11
cols = 4

# subplot return the figure object and axes object
# we can use the axes object to plot specific figures at various locations

fig, axes = plt.subplots(nrows=rows, ncols=cols, figsize = (18,70))

axes = axes.ravel() # to flaten the axes values

for i in np.arange(0, rows*cols): # create evenly spaces variables
    # Select a random number from 0 to len(training)
    num = np.random.randint(0, len(train_image))
    pred =  DL_Model.predict_classes(X_test[i:i+1])[0]
    # read and display an image with the selected index
    axes[i].imshow(X_test[i,:])
    axes[i].axis('off')
    axes[i].set_title('True Character = {} \n Predicted Character = {} '.forma
t(reverse_lookup_train[pred],reverse_lookup_test[np.argmax(pre_defined_label_o
nehot[i])] ), fontsize = 12)

plt.subplots_adjust(hspace=0.3)
```

True Character = A
Predicted Character = A

True Character = A
Predicted Character = A

True Character = C
Predicted Character = A

True Character = D
Predicted Character = A

True Character = A
Predicted Character = A

True Character = A
Predicted Character = A

True Character = C
Predicted Character = A

True Character = B
Predicted Character = A

True Character = C
Predicted Character = A

True Character = A
Predicted Character = A

True Character = A
Predicted Character = A

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = B
Predicted Character = B

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = B
Predicted Character = C

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = C
Predicted Character = C

True Character = A
Predicted Character = D

True Character = D
Predicted Character = D

True Character = D
Predicted Character = D

True Character = D
Predicted Character = D

True Character = D
Predicted Character = D

True Character = D
Predicted Character = D

True Character = D
Predicted Character = D

In [41]:
```
score = DL_Model.evaluate(X_test, pre_defined_label_onehot, batch_size=32)
```

```
44/44 [==============================] - 0s 9ms/step
```

In [42]:
```
print("The Model is {} % accurate with the testing data".format(round(score[1]*100)))
```

```
The Model is 82.0 % accurate with the testing data
```

06 -- Conclusion

- Keras is one of the best library used vastly for the classification of Images
- Convolutional Neural Networks Method helped us alot to achieve a better model by adding different layers of Pooling, Flatten & Densing Laters
- The less no.of batch size denotes that model is trained properly & slowly
- Normalizing the independent variables will provide us best & faster results.
- Shuffling data will assure us that machine understands series of images perfectly.
- Alter the number of epochs untill we have achieved best validation accuracy & less amount of loss.
- Using Categorical Cross Entropy along side with one hot encoding provied us the best accuracy results.
- The same results can be obtained even with the Sparse_Categorical Entropy
- We have successfully obtained a model with 82% accuracy overtthe testing data set.