Summer Intern at ISRO - Summary

Recently I had the oppurtunity to collaborate with Indian Space Research Organisation (ISRO) on a new proje
learning and deep learning to automate the process of defect data classfication. It was wonderful to work w
the insightful experience I had as an intern. I am grateful for being a part of their team and working alongsid

> ### What is Deep Learning?

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humar
computer model learns to perform classification tasks directly from images, text, or sound.

Deep learning is making a big impact in many areas of human life for solving complex problems. Deep learn
the learning dynamics of neurons in human brain. As the scope of AI is expanding from general intelligence
bodily intelligence etc., the scope of deep learning is also expanding rapidly.

> ### Why Deep Learning?

Till now the classification of defect data was done manually which involved laborious human effort and acc
efficiency of the fab. Now with the proposed automation, the software itself will feed the information coming
human involvement. This speeds up the overall operation of wafer processing and inspection.

Explaining the code

We first start our program by importing some of the libraries we would require.

- **glob**: The glob module finds all the pathnames matching a specified pattern according to the rules us
returned in arbitrary order.
- **os**: The OS module in Python provides a way of using operating system dependent functionality. The f
allows you to interface with the underlying operating system that Python is running on – be that Wind
- **numpy**: Numpy is a general-purpose array-processing package. It provides a high-performance multid
working with these arrays. It is the fundamental package for scientific computing with Python.
- **PIL**: Python Imaging Library is a library for the Python programming language that adds support for o
different image file formats.
- **matplotlib**: Matplotlib is a Python 2D plotting library which produces publication quality figures in a va
environments across platforms. It is used to display an array as an image.

```
#Importing Libraries

import glob
import os
import numpy as np
from PIL import Image
from matplotlib import image
import matplotlib.pyplot as plt
```

There are in total of 83 different classes of data and training images corresponding to their classes are store
image pixel values, because of which we have to walk through every class of folder, read the image and then
Since we're treating the data as 2D images of 150x150 pixels, we need to shape it accordingly.

To label the images we give each and every folder a specific integer to denote it. All the images in a particula number given to the folder. This way all the images in a particular folder gets the same label which is intuitiv read, resized into 150x150 pixels and stored in a linear ***numpy*** array.

The labels are stored is in a list.

```
DIR = "/Users/utkarshjain/Documents/SCL/Defect Data/Defect Labeled Data"
os.chdir(DIR)

directory_list = []
dirList = os.listdir("./")
for root, dirs, files in os.walk(DIR):
    for name in dirs:
        directory_list.append(os.path.join(root, name))

image_size = 150
num_classes = 0
num_images = 0
train_list = np.empty([0])
label_list = []


for dirs in directory_list:
    files = glob.glob (dirs+'/*.*')

    num_images = num_images+ len(files)

    temp = [num_classes]*len(files)
    num_classes = num_classes + 1
    label_list = np.append(label_list, temp)
    temp = []

    for file in files:
        im = Image.open(file).convert("L")
        resized = im.resize((image_size,image_size))
        resized = np.asarray(resized)
        train_list = np.append(train_list,resized)
```

The ***numpy*** array containing the training images is reshaped into a 3D array where each plane represent one

```
train_list = train_list.reshape((num_images,image_size,image_size))
train_list = np.asarray(train_list)
```

The label list is then converted into a ***numpy*** array.

```
label_list = label_list.astype(int)
label_list = np.asarray(label_list)
label_list = label_list.reshape(len(label_list),1)
```

We rescale the images into range of 0-1 since it boosts the ***CNN*** image clasifier performance. If we didn't sc of our distributions of feature values would likely be different for each feature, and thus the learning rate wo that would differ (proportionally speaking) from one another. We might be over compensating a correction i undercompensating in another.

We then convert the label data into one-hot-encoded categorical format, which we'll talk about in a second:

```
import tensorflow as tf
train_list /=255
label_list = tf.keras.utils.to_categorical(label_list, 83)
```

The training images are therefore a tensor of shape [***num_images***, 150,150] - ***num_images*** instances of 150x

The label data is encoded as "one_hot" when we loaded it above. Think of one_hot as a binary representatio each handwriting sample was intended to represent. Mathematically one_hot represents a dimension for ev is set to the value 0, except for the "correct" one which is set to 1. For example, the label vector representing 0, 0] (remember we start counting at 0.) It's just a format that's optimized for how the labels are applied duri

So the training label data is a tensor of shape [***num_images***, 83] - ***num_images*** train images each associated whether or not the image represents a given number from 0-82.

Next, we will import model_selection from scikit-learn, and use the function `train_test_split( )` to spl ***Validation Set***. By specifying the test_size as 0.2, we aim to put 20% of the data into our validation set, and tl

> Train Set is the set on which the ***CNN*** trains itself. The trained model is then run on the validation set to the model hyperparameters.

Depending on the data format Keras is set up for, this may be 1x150x150 or 150x150x1 (the "1" indicates a s grayscale. If we were dealing with color images, it would be 3 instead of 1 since we'd have red, green, and bl

```
from tensorflow.keras import backend as K


from sklearn.model_selection import train_test_split
xTrain, xValidation, yTrain, yValidation = train_test_split(train_list, label_list, te


if K.image_data_format()=='channels_first':
    xTrain = xTrain.reshape(xTrain.shape[0],1,image_size,image_size)
    xValidation = xValidation.reshape(xValidation.shape[0],1,image_size,image_size)
    input_shape = (1,image_size,image_size)
else:
    xTrain = xTrain.reshape(xTrain.shape[0],image_size,image_size,1)
    xValidation = xValidation.reshape(xValidation.shape[0],image_size,image_size,1)
    input_shape = (image_size,image_size,1)
```

Next, we will import some libraries which are essential for CNN to work.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Conv2D, MaxPooling2D, Flatten
from tensorflow.keras.optimizers import RMSprop
```

Now for the meat of the problem. Setting up a convolutional neural network involves more layers. Not all of t without pooling and dropout, but those extra steps help avoid overfitting and help things run faster.

We'll start with a 2D convolution of the image - it's set up to take 32 windows, or "filters", of each image, eacl

We then run a second convolution on top of that with 64 3x3 windows - this topology is just what comes rec Again you want to re-use previous research whenever possible while tuning CNN's, as it is hard to do.

Next we apply a MaxPooling2D layer that takes the maximum of each 2x2 result to distill the results down in

A dropout filter is then applied to prevent overfitting.

Next we flatten the 2D layer we have at this stage into a 1D layer. So at this point we can just pretend we hav

... and feed that into a hidden, flat layer of 128 units.

We then apply dropout again to further prevent overfitting.

And finally, we feed that into our final 10 units where softmax is applied to choose our category of 0-82.

```
model = Sequential()
model.add(Conv2D(32, kernel_size=(3, 3),
```

```
                  activation='relu',
                  input_shape=input_shape))

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

model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Dropout(0.25))

model.add(Flatten())

model.add(Dense(128, activation='relu'))

model.add(Dropout(0.5))

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

Let's double check the model description:

```
model.summary()
```

We are still doing multiple categorization, so categorical_crossentropy is still the right loss function to use. W

```
model.compile(loss='categorical_crossentropy', optimizer='rmsprop',metrics=['accuracy
```

Now that our model is ready and compiled, we can fit in our train and validation set and let the CNN train our

Warning

This could take hours to run, and your computer's CPU will be maxed out during that time! Don't run the next computer for a long time. It will print progress as each epoch is run, but each epoch can take around 20 min

```
history = model.fit(xTrain, yTrain,
                    batch_size=2,
                    epochs=5,
                    verbose=2,
                    validation_data=(xValidation, yValidation))
```

This final evaluation gives us the information about the loss and accuracy of the model.

```
score = model.evaluate(xValidation, yValidation, verbose=0)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```