JBM Group, Gurgaon

Name: Gaurav Gosain (Email – gauraveil@gmail.com, Phone – 8800 683 581)

Assignment Allotment Date: 3rd Feb 2021
Assignment Submission Date: 4th Feb 2021

Problem Statement: **Develop an AI model to detect damaged parts from gray scale images based on available images of Good and Bad Parts.**

Problem Approach Flowchart / Pipeline

Tools Used: Spyder Conda Python, Jupyter Notebook, TensorFlow, Keras, Python Image Library, MATPLOTLIB, Numpy, Convolutional Neural Networks

Step 1: Preprocessing the Images

- The images received were comprised of 64 different parts with varying defects – mostly noticeable visibly (slug hole drill miss, warp and shortening).
- For each part 7-10 Good Images and 2-5 Bad Images were available
- The file format was ".tif" and Image Size was 2448 x 2050 Pixels.

1) **Directory Relocation:** First all Images are extracted to be placed in 2 separate folders locally – "Good" and "Bad" clubbing different parts in the same folder. **Note: Ideally a separate machine learning network should be deployed upfront to identify parts and bin them and then defects should be detected for individual parts to avoid false negatives. However, in current scenario data was very less.** This is also done for TensorFlow to identify classes efficiently.
   Code_File = "Base_Code.py"

2) **Resizing and Formatting:** TensorFLow works well with similar sized images and with ".jpeg",".png" formats. Hence each image is resized to approximately quarter of original size and squaring them (Note rectangular images can also work but for efficiency square images are used). **Resulting Filesize = 512 x 512 pixels; Format = ".jpeg"; Renamed: "Good" / "Bad" Tags to avoid Keras naming error with Float32 values.**
   Code_File = "Rename_Images.py"

3) **Removing Corrupt & Duplicate Images:** Some images had defects marked in Yellow for illustrations and hence they were removed to avoid skewing the data. Some other images with corrupt ".tif" headers were also removed as they caused errors in resizing.

Step 2: Developing Convolutional Neural Network

- Final Good Images = 326 ; Final Bad Images = 59
- Image Size = 512 x 512
- Image Type = ".jpeg"

Ideally defects are associated with part features and surface defects. Since these are not uniform for each part and vary – **a feature identifying Convolutional Neural Network was required.**

Note: **Literature Survey reveals Nearest Neighbors, Decision Tree Algorithms could also be used, however these would hold good if large dataset is available for each damaged part separately.**

Setup TensorFlow in Jupyter Notebook: a CPU based TensorFLow environment was used. A Sequential Layering Model was used.

Initial Model Conditions		
Parameter	**Value**	**Description**
Batch_size	32	Default – not changed
Validation_split	0.2	80% - 20% Validation Split
Total Convolutional Layers	3	Used as a starting point to test accuracy on cost of time complexity
Total Dense Layers	2	Downstream of Convolutional and Output Layer
CNN_Filters	64	Number of Convolutional Neurons at each layer
Filter_window	(3,3)	Filter Matrix moving window to be summed up over each image
Pool_size	(2,2)	MaxPooling matrix size
Activation	Relu	Rectified Linear Unit – for better convergence vs sigmoidal
Optimizer	Adam	Default – good for stochastic gradient descent
Loss Function	SparseCategoricalCrossEntropy	Looks at difference in probability distributions of output neurons vs desired
Metric	Accuracy	How often network guesses the correct answer.
Epochs	5 to 10	Repetitions of Guesses

Using TensorFlow Convoluted Neural Network to Identify Defected Parts

In [1]:
```
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Activation, Flatten, Conv
```

Define Parameters for Preprocessing

In [2]:
```
batch_size = 32
img_height = 512
img_width = 512
data_dir = r"C:\Users\Gaurav Gosain\Desktop\Important Docs\JBM Assignment\Clu
```

Import Data from Local Computer

In [3]:
```
train_ds = tf.keras.preprocessing.image_dataset_from_directory(
data_dir, labels='inferred', label_mode='int', color_mode='grayscale',image_s
validation_split=0.2,subset="training",batch_size=batch_size)
```

```
Found 385 files belonging to 2 classes.
Using 308 files for training.
```

Create Validation Data from Local Computer

In [4]:
```
val_ds = tf.keras.preprocessing.image_dataset_from_directory(
data_dir, labels='inferred', label_mode='int', color_mode='grayscale',image_s
validation_split=0.2,subset="validation",batch_size=batch_size)
```

```
Found 385 files belonging to 2 classes.
Using 77 files for validation.
```

In [5]:
```
class_names = train_ds.class_names
print(class_names)
```

```
['Bad', 'Good']
```

Deploy a Convolution Neural Network with 64 Filter Zone and a 3 x 3 Pixel Moving Window

In [6]:
```
num_classes = 2
CNN_Filters = 64
filter_window = (3,3)
pool_size = (2,2)
from tensorflow.keras import layers

model = tf.keras.Sequential([
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(CNN_Filters, filter_window, activation='relu'),
  layers.MaxPooling2D(pool_size = pool_size),
  layers.Conv2D(CNN_Filters, filter_window, activation='relu'),
  layers.MaxPooling2D(pool_size = pool_size),
  layers.Conv2D(CNN_Filters, filter_window, activation='relu'),
  layers.MaxPooling2D(pool_size = pool_size),

  layers.Flatten(),
  layers.Dense(CNN_Filters, activation='relu'),

  layers.Dense(num_classes)
])
```

Compile the CNN Model

In [7]:
```
model.compile(
  optimizer='adam',
  loss=tf.losses.SparseCategoricalCrossentropy(from_logits=True),
  metrics=['accuracy'])
```

Fit the Model with 5 Epochs

In [8]:
```
epochs = 5
model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs
)
```

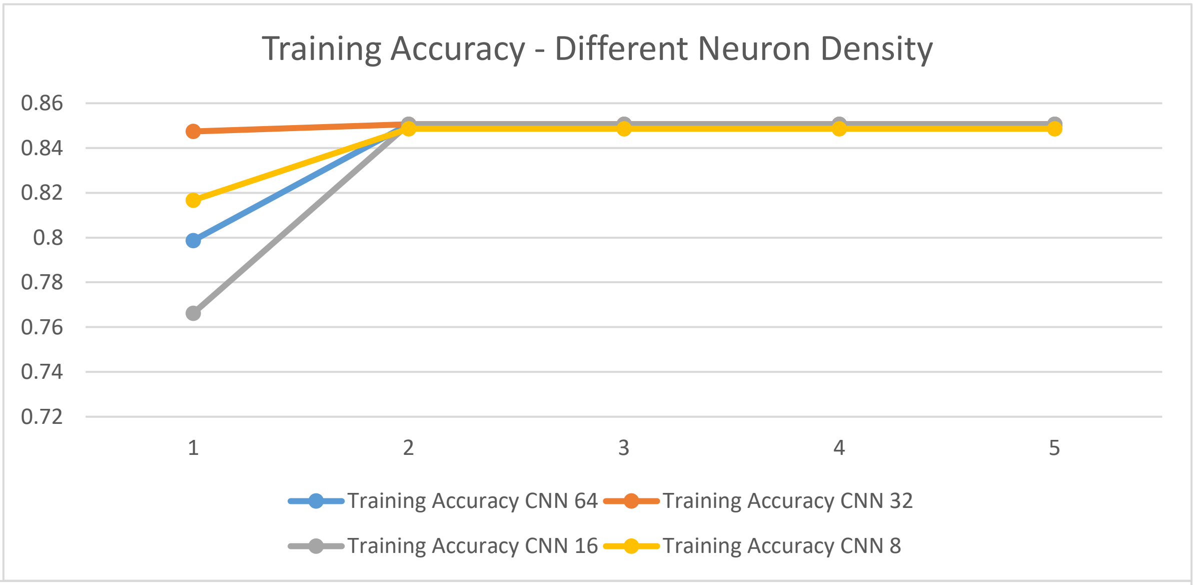
```
Epoch 1/5
10/10 [==============================] - 112s 11s/step - loss: 0.6815 - acc
uracy: 0.8182 - val_loss: 0.4496 - val_accuracy: 0.8312
Epoch 2/5
10/10 [==============================] - 123s 12s/step - loss: 0.4132 - acc
uracy: 0.8506 - val_loss: 0.4742 - val_accuracy: 0.8312
Epoch 3/5
10/10 [==============================] - 117s 12s/step - loss: 0.3698 - acc
uracy: 0.8506 - val_loss: 0.5983 - val_accuracy: 0.8312
Epoch 4/5
10/10 [==============================] - 116s 12s/step - loss: 0.3312 - acc
uracy: 0.8506 - val_loss: 0.4935 - val_accuracy: 0.8312
Epoch 5/5
10/10 [==============================] - 115s 11s/step - loss: 0.2262 - acc
uracy: 0.8506 - val_loss: 0.4773 - val_accuracy: 0.8312
```

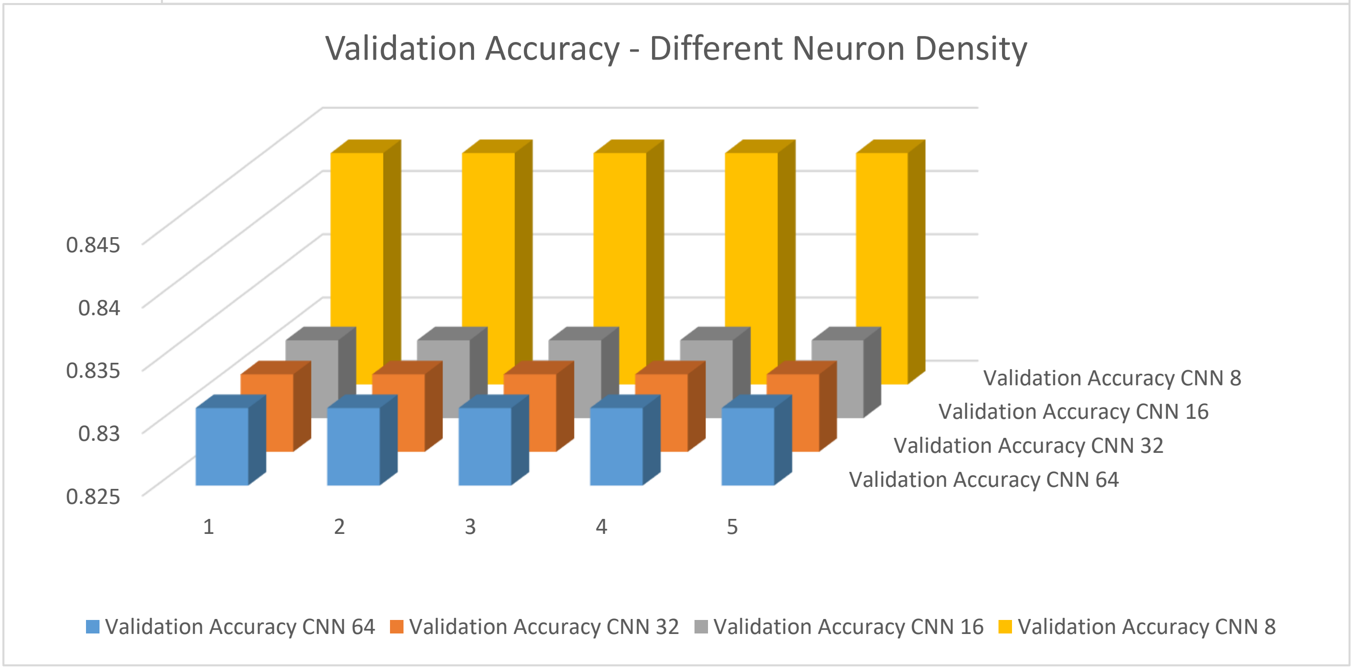
Out[8]:
```
<tensorflow.python.keras.callbacks.History at 0x1dfff415a60>
```

Step 3: Model Sensitivity Analysis: With 3 CNN layering – 83% accuracy is achieved!

However, 64 Neuron x 3 Layers Network is time consuming. Also, **there is slight hint of overfitting as validation accuracy reaches a Global Minima very early – signs that the network is memorizing the training images.**

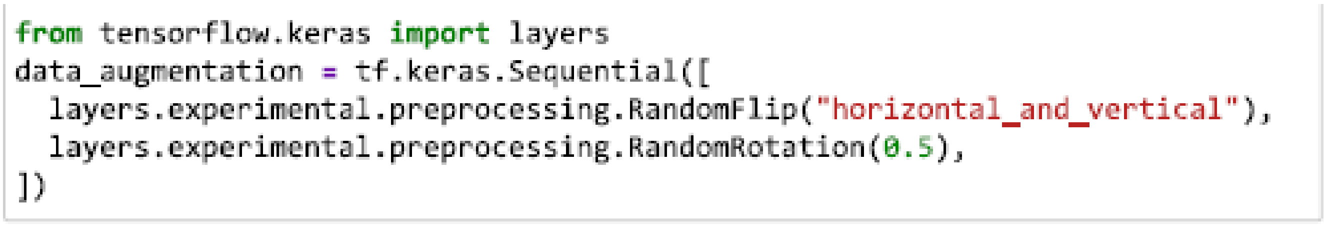
1) **Reducing the Number of Neurons –** this is done to avoid overfitting but also prevent underfitting and hence number of neurons are reduced from 64 -> 32 -> 16 - > 8





Result: It is clear that CNN network with 8 neurons density is sufficient to match desired accuracy of 83%

2) **Applying Data Augmentation**: since data is less, to achieve better accuracy, data augmentation was used. This process would also **future proof the algorithm from identifying same part with rotated and flipped images.**

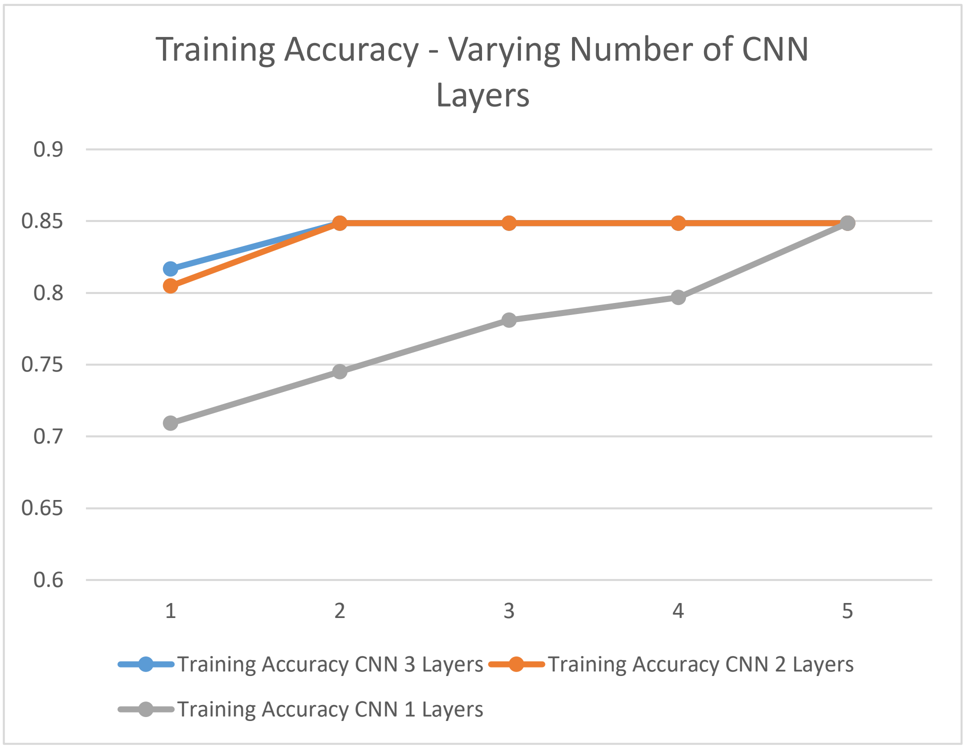


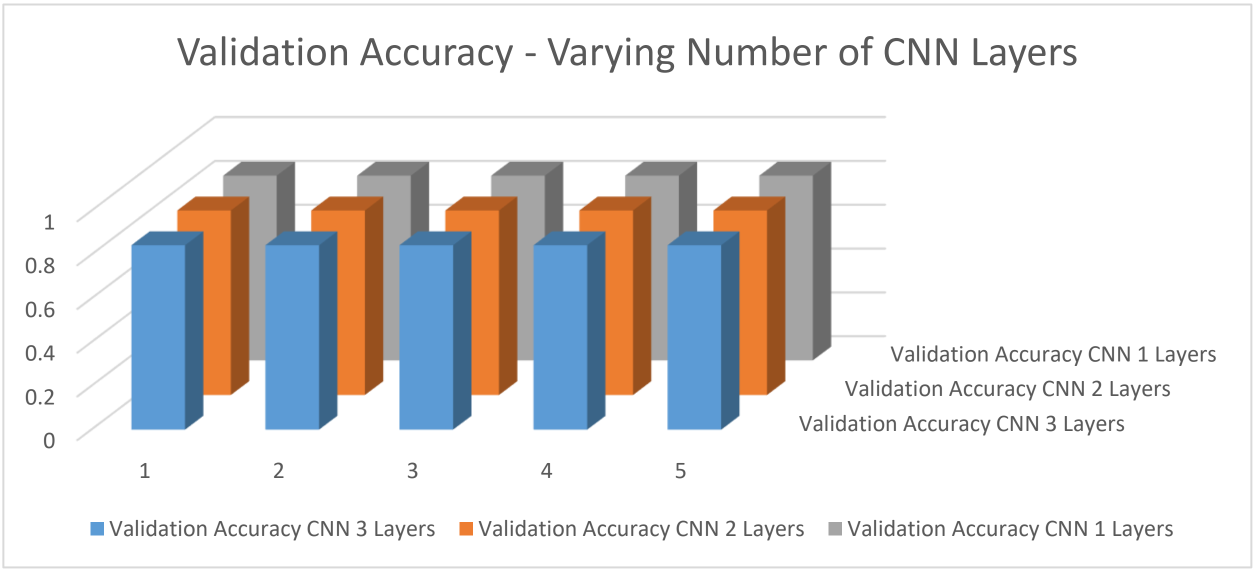
```
from tensorflow.keras import layers
data_augmentation = tf.keras.Sequential([
  layers.experimental.preprocessing.RandomFlip("horizontal_and_vertical"),
  layers.experimental.preprocessing.RandomRotation(0.5),
])
```

3) **Applying Dropout and Kernel_Regularization:** both dropouts (50% Probability) and L1 Kernel_Regularization were applied to Convolutional Layers to reduce the chances of overfitting.

- Final Optimized Dropout = 50% after each CNN Layer
- Kernel_Regularization = L1 regularizer

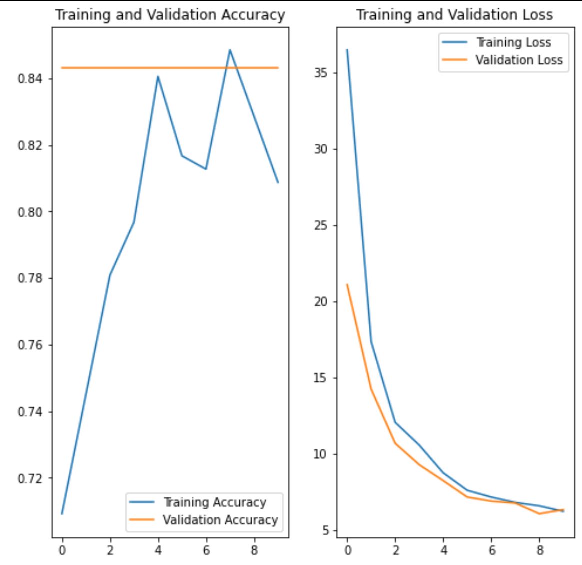
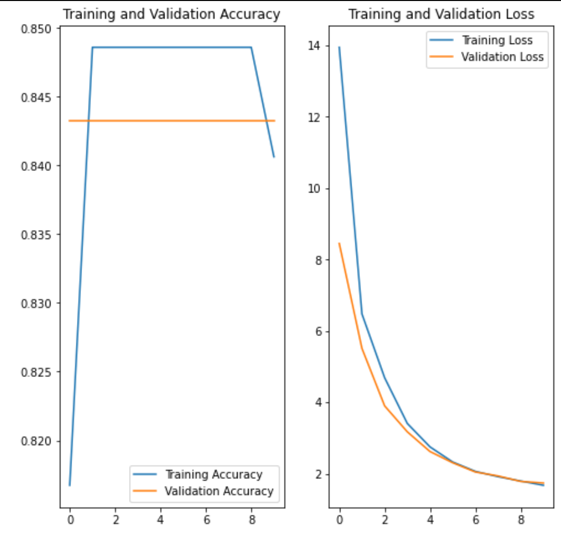
4) **Optimizing Number of Convolutional Layers:** after optimizing the number of neurons per layer, the number of layers were reduced from 3 -> 2 -> 1 to optimize the model further.





Result: It is clear that CNN network with 1 Layer is sufficient to match desired accuracy of 83%

5) **Varying Validity Ratio:** to test robustness of model, validity ratio was increased from 0.2 -> 0.3 -> 0.35. Validation and Training Losses were plotted to find Minima.

Validity Ratio = 0.3; 8 Neurons/ Layer; 1 CNN Layer	**Validity Ratio = 0.35; 8 Neurons/ Layer; 1 CNN Layer**
	

Final Result

- **An efficiency of ~84% is achieved with Convolutional Neural Network with 1 Layer followed by 2 Dense Layers.**
- **Number of 8 Neurons per layer is sufficient to achieve desired accuracy**
- **For better results more images per part are required and a machine learning algorithm needs to be placed over this neural network to bin the parts.**