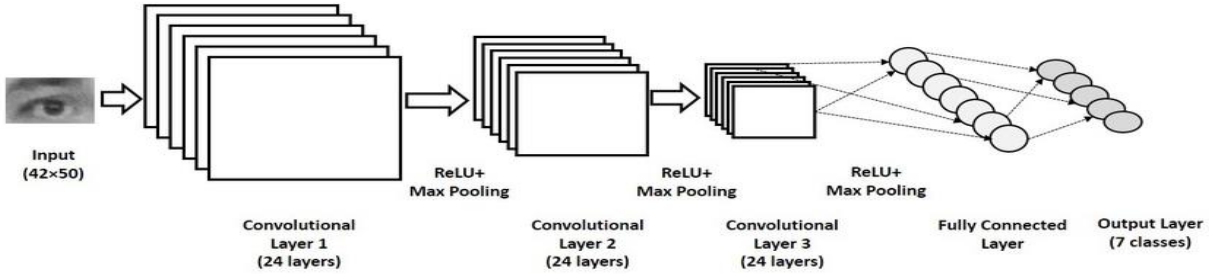
Computer Vision
Assignment – 04 Report
Aditya Yaji (A20426486)

Abstract:

This is the report of Assignment-04 for CS512. The task was to build a Convolutional Neutral Network for classification of handwritten image for digit recognition and classification. Implementation involves GPU Framework **Keras**. The report consists of detailed implementation of system which recognize and classify images of numbers in the **MNIST** dataset as either even or odd.

Convolutional Neural Network with Keras:



CNN architecture

Keras is higher level library which operates over TensorFlow or Theano and it is intended to stream-line the process of building deep learning networks. It was developed with a focus on enabling fast experimentation. Being able to go from idea to result with the least possible delay is key to doing good research. Keras allows for easy and fast prototyping through user friendliness, modularity and extensibility. Supports both convolutional and recurrent networks as well as combinations of the two. Runs seamlessly on CPU and GPU.

The code data structure of Keras is a Model, a way to organize layers. The simplest type of model is the Sequential model, a linear stack of layers.

Keras supports multiple backends engines and does not lock you into ecosystem. Available backends are :

1. The TensorFlow (From Google).
2. The CNTK backend (From Microsoft).
3. The Theano backend.

TensorFlow backend:

TensorFlow is an open source software library for high performance numerical computations. The high-level Keras API provides building blocks to create and train deep learning models. Its flexible architecture allows for the easy deployment of computation across a variety of platforms (CPUs, GPUs, TPUs), and from desktops to clusters of servers to mobile and edge devices. TensorFlow computations are expressed as stateful dataflow graphs.

CNTK backend:

Now, CNTK is known as The Microsoft Cognitive Toolkit. It empowers to harness the intelligence within massive datasets through deep learning by providing uncompromised scaling, speed, and accuracy with commercial-grade quality and compatibility with the programming languages and algorithms.

Theano backend:

Theano is a Python library that allows you to define, optimize, and evaluate mathematical expressions involving multi-dimensional arrays efficiently. It allows tight integration with NumPy (numpy.ndarray).

Deliverables 1: Custom CNN

Convolutional Neural Network is built using Keras. MINST dataset is used to train and evaluate the network. **M**odified **N**ational **I**nstitute of **S**tandards and **T**echnology dataset is a large database of handwritten digits that is commonly used for training various image processing systems. It was created by "re-mixing" the samples from NIST's (National Institute of Standards and Technology) original datasets. The MINST database contains 60,000 training images and 10,000 testing images. Half of the training set and half of the test set is taken from NIST's training database, while other half of both training and test set is taken from NIST's testing dataset. NIST's training set is taken from American Census Bureau employees, testing set is taken from American High School students.

- We are using Sequential Model. The Sequential model is a linear stack of layers. Sequential model is created by passing a list of layer instances to the constructor.
- Here we are creating 2 Convolutional layers and a pooling layer. In the 1st layer, 32 filters of kernel size 5x5 is applied. In the 2nd layer, 64 filters of kernel size 5x5 is used.

- Pooling in both layers is down sampled by a factor of 2.
- Applying dropout at the rate of 40%.
- Then compute cross entropy loss.

Below are the important functions briefly explained:

- ***add()*** : This method is used to add layers to the network.

- ***Conv2D****(filters, kernel_size, strides=(1, 1), padding='valid', data_format=None, dilation_rate=(1, 1), activation=None, use_bias=True, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=None, bias_regularizer=None, activity_regularizer=None, kernel_constraint=None, bias_constraint=None)*

where,

- **filters** : integer, dimensionality of the output space i.e the number of output filters.
- **kernel_size** : An integer or tuple/ list of 2 integers, specifying the height and weight of the 2D convolution window.
- **strides** : An integer or tuple/ list of 2 integers, specifying the strides of the convolution along the height and width.
- **padding** : value can be 'valid' or 'same'. 'same' is inconsistent across backends with strides!=1.
- **data_format** : 'channels_last' or 'channels_first'. The ordering of dimensions in the inputs.
- **dilation_rate** : an integer or tuple/list of 2 integers, specifying the dilation rate to use for dilated convolution. specifying any dilation_rate!=1 is incompatible with specifying any stride value != 1.
- **activation** : specify activation function to use. If you don't specify anything, no activation is applied.
- **use_bias**: Boolean, whether the layer uses a bias vector.
- **kernel_initializer**: Initializer for the kernel weights matrix.
- **bias_initializer**: Initializer for the bias vector.
- **activity_regularizer**: Regularizer function applied to the output of the layer (its "activation").
- **kernel_constraint**: Constraint function applied to the kernel matrix.
- **bias_constraint**: Constraint function applied to the bias vector.

Input:

4D tensor with shape: (batch,channels,rows,cols) if data_format is ’channels_first’or 4D tensor with shape: (batch,rows,cols,channels) if data_format is ‘channels_last’.

Output:

4D tensor with shape: (batch,filters,new_rows,new_cols) if data_format is ’channels_first’or 4D tensor with shape: (batch,new_rows,new_cols,filters) if data_format is ‘channels_last’. rows and cols values might have changed due to padding.

- ***MaxPooling2D****(pool_size=(2, 2), strides=****None****, padding='valid', data_format=****None****)*

where,

1. **pool_size :** integer or tuple of 2 integers, factors by which to downscale (vertical, horizontal). (2, 2) will halve the input in both spatial dimension. If only one integer is specified, the same window length will be used for both dimensions.
2. **strides**: Integer, tuple of 2 integers, or None. Strides values. If None, it will default to pool_size.
3. **Padding** : ‘valid’ or ‘same’.
4. **data_format** : ‘channels_last’ or ‘channels_first’. The ordering of dimensions in the inputs.

Input:

If **data_format= ‘channels_last’** : 4D tensor with shape (batch_size ,rows,cols,channels).
If **data_format= ‘channels_first’** : 4D tensor with shape (batch_size ,channels,rows,cols).

Output:

If **data_format= ‘channels_last’** : 4D tensor with shape (batch_size , pooled_rows, pooled_cols, channels).
If **data_format= ‘channels_first’** : 4D tensor with shape (batch_siz1qe ,channels, pooled_rows, pooled_cols).

- ***Dropout****(rate, noise_shape=****None****, seed=****None****)*

where,

1. **rate**: float between 0 and 1. Fraction of the input units to drop.
2. **noise_shape**: 1D integer tensor representing the shape of the binary dropout mask that will be multiplied with the input.
3. **seed**: A Python integer to use as random seed.

- ***Dense****(units, activation=****None****, use_bias=****True****, kernel_initializer='glorot_uniform', bias_initializer='zeros', kernel_regularizer=****None****, bias_regularizer=****None****, activity_regularizer=****None****, kernel_constraint=****None****, bias_constraint=****None****)*

where,

1. **units**: Positive integer, dimensionality of the output space.
2. **activation** : specify activation function to use. If you don't specify anything, no activation is applied.
3. **use_bias**: Boolean, whether the layer uses a bias vector.
4. **kernel_initializer**: Initializer for the kernel weights matrix.
5. **bias_initializer**: Initializer for the bias vector.
6. **activity_regularizer**: regularizer function applied to the output of the layer (its "activation").
7. **kernel_constraint**: Constraint function applied to the kernel matrix.
8. **bias_constraint**: Constraint function applied to the bias vector.

Input:

nD tensor with shape : (batch_size, ..., input_dim).

Output:

nD tensor with shape: (batch_size, ..., units).

Once the model is build, it is compiled using compile function.

- *compile(optimizer, loss , metrics, loss_weights, sample_weight_mode, weighted_metrics, target_tensors)*

where,

- **optimizer**: String (name of optimizer) or optimizer instance.
- **loss**: String (name of objective function) or objective function.
- **metrics**: List of metrics to be evaluated by the model during training and testing. Typically, you will use metrics =['accuracy'].
- **loss_weights**: Optional list or dictionary specifying scalar coefficients (Python floats) to weight the loss contributions of different model outputs.
- **sample_weight_mode**: If you need to do timestep-wise sample weighting (2D weights), set this to 'temporal'. None defaults to sample wise weights.
- **weighted_metrics**: List of metrics to be evaluated and weighted by sample_weight or class_weight during training and testing.
- **target_tensors**: By default, Keras will create placeholders for the model's target, which will be fed with the target data during training.

➢ ***optimizers***: This is one of the two arguments required for compiling a keras model.

- sgd is used Stochastic gradient descent optimizer. Includes support for momentum, learning rate decay, and Nesterov momentum.

optimizers.SGD(lr, momentum, decay, nesterov)

where,

- **lr** : Learning rate which is a float value >=0.
- **momentum :** Parameter that accelerates SGD in the relevant direction and dampens oscillations. It is a float value >=0.
- **decay :** Learning rate decay over each update. Float value >=0.
- **nesterov :** Boolean value, which indicates whether to apply Nesterov momentum or not.
- ***loss*** : This is one of the two arguments required for compiling a Keras model. Available loss functions are : mean_squared_error, mean_absolute_error, mean_absolute_percentage_error , mean_squared_logarithmic_error, squared_hinge, hinge, logcosh, categorical_crossentropy.

- *fit(x=**None**, y=**None**, batch_size=**None**, epochs=1, verbose=1, callbacks=**None**, validation_split=0.0, validation_data=**None**, shuffle=**True**, class_weight=**None**, sample_weight=**None**, initial_epoch=0, steps_per_epoch=**None**, validation_steps=**None**)*

where,

1. x : Numpy array of training data.
2. y : Numpy array of target data.
3. **batch_size**: Integer, Number of samples per gradient update. If unspecified, batch_size will default to 32.
4. **epochs**: Integer. Number of epochs to train the model. An epoch is an iteration over the entire x and y data provided.
5. **verbose**: Integer. 0, 1, or 2. Verbosity mode. 0 = silent, 1 = progress bar, 2 = one line per epoch.
6. **callbacks**: List of callback instance.
7. **validation_split**: Float between 0 and 1. Fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.
8. **shuffle**: Boolean (whether to shuffle the training data before each epoch) or str (for 'batch').
9. **validation_data**: tuple (x_val,y_val) or tuple (x_val,y_val,val_sample_weights) on which to evaluate the loss and any model metrics at the end of each epoch. The model will not be trained on this data. validation_data will override validation_split.
10. **class_weight**: Optional dictionary mapping class indices (integers) to a weight (float) value, used for weighting the loss function.
11. **sample_weight**: Optional Numpy array of weights for the training samples, used for weighting the loss function.
12. **initial_epoch**: Integer. Epoch at which to start training (useful for resuming a previous training run).
13. **steps_per_epoch**: Integer. Total number of steps (batches of samples) before declaring one epoch finished and starting the next epoch.
14. **validation_steps**: Only relevant if steps_per_epoch is specified.

Output :

fit function returns history object. History attributes is a dictionary recording training loss values and metrics values at successive epochs, as well as validation loss values and validation metrics values.

Recall and Precision metrics has been removed from Keras 2.0. We can still wrap the TensorFlow metrics into keras.metrics

- The **recall** function creates two local variables, true_positives and false_negatives, that are used to compute the recall. This value is ultimately returned as recall, an idempotent operation that simply divides true_positives by the sum of true_positives and false_negatives.

  tensorflow.metrics.recall(labels,predictions,weights=None,metrics_collections=None,updates_collections=None,name=None)

  where,

  - **labels:** The ground truth values, a Tensor whose dimensions must match predictions.
  - **predictions :** The predicted values, a tensor of arbitrary dimensions.
  - **weights :** optional Tensor whose rank is either 0 or same rank as labels.
  - **metrics_collections:** An optional list of collections that recall should be added to.
  - **updates_collections:** An optional list of collections that update_op should be added to.
  - **name:** An optional variable_scope name.

- **precision** function creates two local variables, true_positives and false_negatives, that are used to compute the precision. This value is ultimately returned as recall, an idempotent operation that simply divides true_positives by the sum of true_positives and false_negatives.

  tensorflow.metrics.precision(labels,predictions,weights=None,metrics_collections=None,updates_collections=None,name=None)

  where,

  - **labels:** The ground truth values, a Tensor whose dimensions must match predictions.
  - **predictions :** The predicted values, a tensor of arbitrary dimensions.

- **weights :** optional Tensor whose rank is either 0 or same rank as labels.
- **metrics_collections:** An optional list of collections that recall should be added to.
- **updates_collections:** An optional list of collections that update_op should be added to.
- **name:** An optional variable_scope name.

- **save**: This function is used to save the model created. Keras models are saved into one HDF5 file which will contain the architecture of the model, weights of the model, the training configuration (loss, optimizers), the state of optimizers.

Illustration*:*

model.save(filepath)

model.save_weights(filepath)

where,

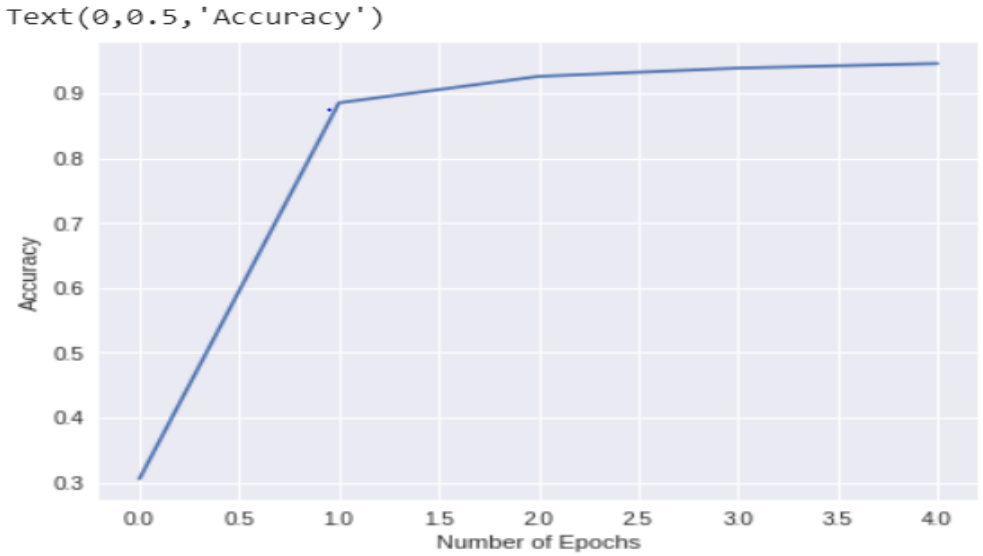
filepath : is the location where the model gets saved.

Here, we are saving the model as **deliverable_1.h5** and their corresponding weights as **my_model weights_deliverable_1.h5** to the google drive. Later, in use, we are loading the same model and their weights from the drive to recognize and classify the image.

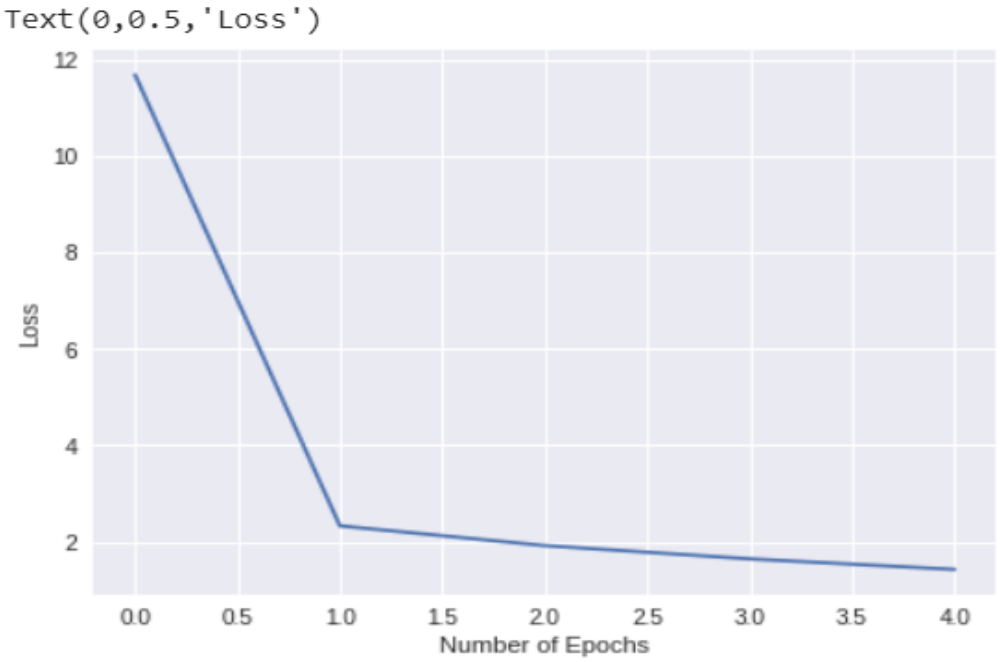
Outcome of deliverable – 01:

```
Train on 55000 samples, validate on 10000 samples
Epoch 1/5
55000/55000 [==============================] - 12s 219us/step - loss: 11.6888 - categorical_accuracy: 0.3055 - precision: 0.3664 - recall: 0.2397 - val_loss: 2.5328 - val_categorical_accuracy: 0.8868
Epoch 2/5
55000/55000 [==============================] - 12s 213us/step - loss: 2.3303 - categorical_accuracy: 0.8858 - precision: 0.9301 - recall: 0.8379 - val_loss: 2.0553 - val_categorical_accuracy: 0.9281
Epoch 3/5
55000/55000 [==============================] - 12s 213us/step - loss: 1.9208 - categorical_accuracy: 0.9265 - precision: 0.9527 - recall: 0.9021 - val_loss: 1.7008 - val_categorical_accuracy: 0.9601
Epoch 4/5
55000/55000 [==============================] - 12s 213us/step - loss: 1.6453 - categorical_accuracy: 0.9394 - precision: 0.9597 - recall: 0.9187 - val_loss: 1.4805 - val_categorical_accuracy: 0.9573
Epoch 5/5
55000/55000 [==============================] - 12s 214us/step - loss: 1.4259 - categorical_accuracy: 0.9464 - precision: 0.9641 - recall: 0.9281 - val_loss: 1.3010 - val_categorical_accuracy: 0.9578
```

Accuracy graph of deliverable – 01:



Loss graph of deliverable – 01:



Deliverable – 02 : Parameter tuning

Here, we will change the parameters and create different models to evaluate the performance of each changed model. This deliverable helps to compare the results and draw a conclusion among the models. Here, we are changing the following parameters for each variation of models.

- Network architecture (by changing the no. of layers or reorganizing the layers)
- Receptive fields and strides.

- Optimizer and loss function.
- Epochs, dropout rate, learning rate, size of filters, batch size and so on.

Comparsion of Results:

Open excel for results comparison

Thus, from the model comparison obtained from the excel sheet, we can conclude that, **Variation2** has good value for accuracy, precision and recall metrics with low loss rate.

Deliverable – 03 : Building an application which is used to classify the numbers

Deliverable 3 is built by using the model developed in deliverable 1. Here, we are using the saved model and its corresponding weights of deliverable – 01 that is, **deliverable_1.h5 and my_model_weights_deliverable_1.h5**.
The program works as follows:
The program execution will prompt for the location of the image. The image should be an handwritten digit saved locally.
If 'q' or 'esc' is pressed, program will terminate.
Else, following functionality will be executed.

- Program accepts an input handwritten image of digit.
- Image is preprocessed by using OpenCV
  1. Image is resized to 28x28.
  2. Transform gray scale image to binary image using GaussianBlur() and Threshold() function.

  - **GaussianBlur**() :The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The equation of a Gaussian function in one dimension is,

$$G(x,y)=\frac{1}{2\pi\sigma^2}e^{-\left(x^2+y^2\right)/\left(2\sigma^2\right)}$$

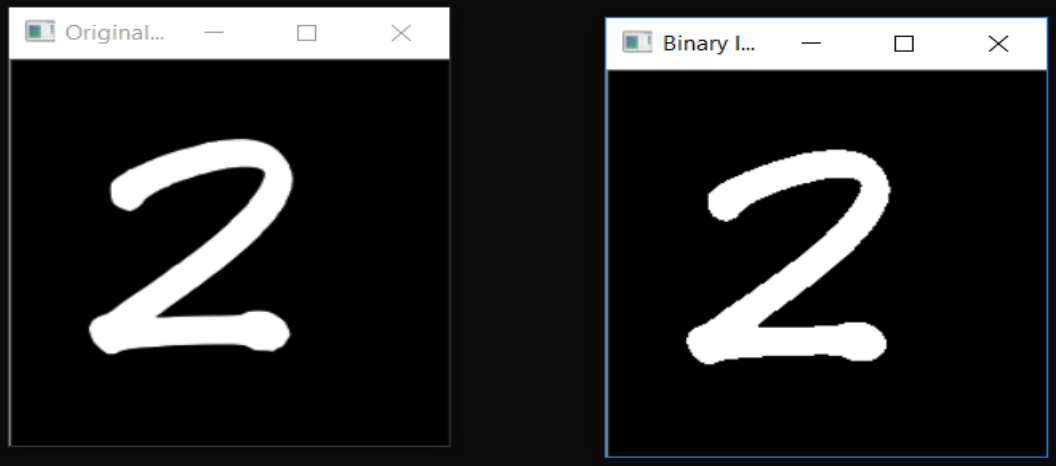
GaussianBlur() is widely used because it reduces image noise.

- threshold() : Thresholding is the method where the threshold value is calculated for smaller regions and therefore, there will be different threshold values for different regions.

3. Both binary image and original image is displayed in two separate windows.
4. Now, by using the CNN model developed in deliverable 1, we will classify the above processed image as even or odd.
5. We should perform prediction on the image using the model and then identify the digit in it.
6. Once we have the digit, applying modulus operation to find whether the digit is even or odd. If the digit is completely divisible by 2 then it is even number else, it is odd number.

Outcome of deliverable – 03:

```
(tfenv) C:\Users\adityayaji\Desktop\Assignments\CV\AS4\src>python cnn_test.py
Using TensorFlow backend.
2018-11-01 17:00:56.183279: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports instructions that
2018-11-01 17:00:56.659720: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1411] Found device 0 with properties:
name: GeForce 940MX major: 5 minor: 0 memoryClockRate(GHz): 1.2415
pciBusID: 0000:01:00.0
totalMemory: 4.00GiB freeMemory: 3.35GiB
2018-11-01 17:00:56.671071: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1490] Adding visible gpu devices: 0
2018-11-01 17:00:58.113897: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] Device interconnect StreamExecut
2018-11-01 17:00:58.131392: I tensorflow/core/common_runtime/gpu/gpu_device.cc:977]      0
2018-11-01 17:00:58.134579: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0:   N
2018-11-01 17:00:58.138544: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1103] Created TensorFlow device (/job
 GeForce 940MX, pci bus id: 0000:01:00.0, compute capability: 5.0)
enter path of the Image :C:\Users\adityayaji\Desktop\Assignments\CV\AS4\data\digit_3.png
Prediction
[2]
Image has an even number
```



```
(tfenv) C:\Users\adityayaji\Desktop\CV_AS_04>python cnn_test.py
Using TensorFlow backend.
2018-11-01 00:51:34.258117: I tensorflow/core/platform/cpu_feature_guard.cc:141] Your CPU supports in
2018-11-01 00:51:34.879015: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1411] Found device 0 w
name: GeForce 940MX major: 5 minor: 0 memoryClockRate(GHz): 1.2415
pciBusID: 0000:01:00.0
totalMemory: 4.00GiB freeMemory: 3.35GiB
2018-11-01 00:51:34.895260: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1490] Adding visible g
2018-11-01 00:51:47.922624: I tensorflow/core/common_runtime/gpu/gpu_device.cc:971] Device interconne
2018-11-01 00:51:47.936279: I tensorflow/core/common_runtime/gpu/gpu_device.cc:977]      0
2018-11-01 00:51:47.938917: I tensorflow/core/common_runtime/gpu/gpu_device.cc:990] 0:   N
2018-11-01 00:51:47.942218: I tensorflow/core/common_runtime/gpu/gpu_device.cc:1103] Created TensorFl
 GeForce 940MX, pci bus id: 0000:01:00.0, compute capability: 5.0)
enter path of the ImageC:\Users\adityayaji\Desktop\CV_AS_04\Images\digit_1.png
Prediction
[3]
Image has a odd number
```