CS512 – AS4 - Report

Akshay Jain

Department of Computer Science

Illinois Institute of Technology

April 7, 2022

Abstract

In this assignment, we have implemented Convolution Neural Network with different variations for classification problem using MNIST and CIFAR10 datasets. To do these operations, we have majorly used Keras and TensorFlow libraries in python on google colab notebook: https://colab.research.google.com/ free account.

1. Problem Statements

We have 4 problem statements for this assignment:

1. Binary Classification:
    - Load MNIST dataset and split it into train/validation/test subsets
    - Convert digit labels into odd/even labels
    - Construct a CNN network with two convolution layers with pooling, a dropout layer and two fully connected layers.
    - Select appropriate loss function, optimization algorithm and evaluation metric
    - Train the network and record the training and validation loss and accuracy
    - Plot the training and validation loss as a function of epochs.
    - Plot the accuracy as a function of epochs.
    - Report the loss and accuracy values of the final training step.

2. Hyperparameter Tuning: Evaluate different variations of the basic network as described below and measure performance. Compare the results and draw conclusions:
    - Changing the network architecture
    - Changing the receptive field and stride parameters
    - Changing optimizer and loss function
    - Changing various parameters (e.g., dropout, learning rate, number of filters, number of epochs)
    - Adding Batch and layer Normalization
    - Using different weight initializers
    - Evaluate the best validation model on the testing subset.

3. Inference: Write a program to use pretrained custom CNN. The program should do the following:
    - Accept as input an image of a handwritten digit. Assume each image contains one digit.
    - Using OpenCV do some basic image pre-processing to prepare the image for your CNN. Resize the image to fit your model’s image size requirement. Transform the

grayscale image to binary image (using GaussianBlur() and adaptiveThreshold(), or any other type of binary thresholding that performs well): Display the original and binary image into separate windows.
- Using your CNN classify the binary image (even/odd).

4. Multiclass Classification:
   - Download the CIFAR10 and load the pickled data into you program.
   - Build a convolution neural network with several convolutions, pooling and normalization layers. Flatten the output of the convolution layers and pass it to a single dense layer that will produce the output using SoftMax activation.
   - Test the performance of the model you built and tune hyper parameters as needed.
   - Add one or two inception blocks and test performance.
   - Remove the inception blocks and add one or two residual blocks instead. Test performance and compare to previous results.

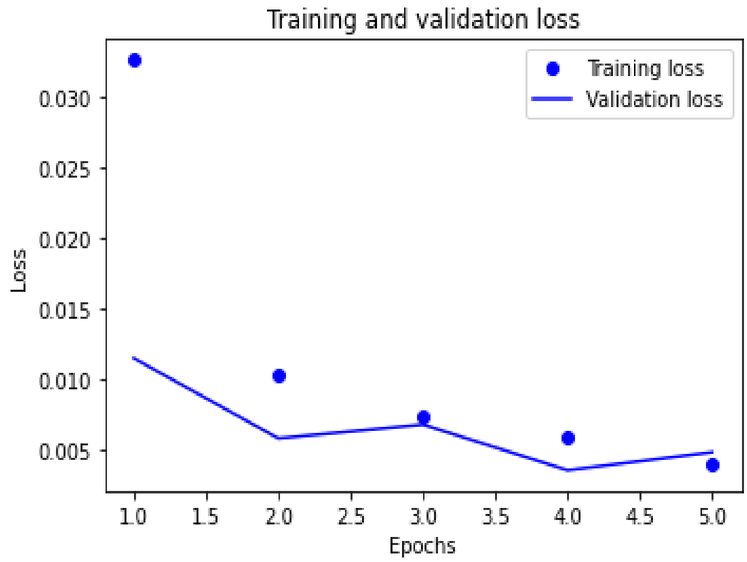
2. Proposed solution

1) For Binary Classification following are the proposed solutions:

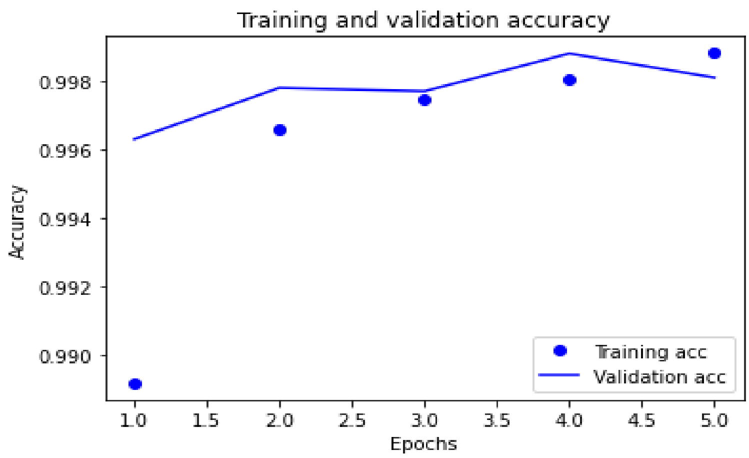
<u>"Summary of CNN model used for Binary classification"</u>

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_4 (Conv2D)           (None, 26, 26, 32)        320

 max_pooling2d_4 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_5 (Conv2D)           (None, 11, 11, 64)        18496

 max_pooling2d_5 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten_2 (Flatten)         (None, 1600)              0

 dropout_2 (Dropout)         (None, 1600)              0

 dense_4 (Dense)             (None, 64)                102464

 dense_5 (Dense)             (None, 1)                 65

=================================================================
Total params: 121,345
Trainable params: 121,345
Non-trainable params: 0
```

"Plot the training and validation loss as a function of epoch"



"Plot the training and validation accuracy as a function of epoch"



Final Step Loss and Accuracy of train and validation dataset:

- train_loss: 0.0040 - train_accuracy: 0.9988
- val_loss: 0.0048 - val_accuracy: 0.9981

2) For Hyper Parameter Tunning following are the proposed solutions with different variations in basic network as follows:

a) On changing the network architecture (added one more Conv2D layer with filters = 128)

<u>"Model Summary"</u>

```
Model: "sequential_4"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_10 (Conv2D)          (None, 26, 26, 32)        320

 max_pooling2d_9 (MaxPooling  (None, 13, 13, 32)       0
 2D)

 conv2d_11 (Conv2D)          (None, 11, 11, 64)        18496

 max_pooling2d_10 (MaxPoolin  (None, 5, 5, 64)         0
 g2D)

 conv2d_12 (Conv2D)          (None, 3, 3, 128)         73856

 max_pooling2d_11 (MaxPoolin  (None, 1, 1, 128)        0
 g2D)

 flatten_3 (Flatten)         (None, 128)               0

 dropout_3 (Dropout)         (None, 128)               0

 dense_6 (Dense)             (None, 64)                8256

 dense_7 (Dense)             (None, 1)                 65

=================================================================
Total params: 100,993
Trainable params: 100,993
Non-trainable params: 0
```

b) On changing stride parameters and optimizer/ loss values ***(clubbed part b and c of question 2)***:

<u>"Model Summary"</u>

```
Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_20 (Conv2D)          (None, 13, 13, 32)        320

 max_pooling2d_17 (MaxPoolin  (None, 6, 6, 32)         0
 g2D)

 conv2d_21 (Conv2D)          (None, 4, 4, 64)          18496

 max_pooling2d_18 (MaxPoolin  (None, 2, 2, 64)         0
 g2D)
```

```
 flatten_5 (Flatten)         (None, 256)               0

 dropout_5 (Dropout)         (None, 256)               0

 dense_10 (Dense)            (None, 64)                16448

 dense_11 (Dense)            (None, 1)                 65

=================================================================
Total params: 35,329
Trainable params: 35,329
Non-trainable params: 0
```

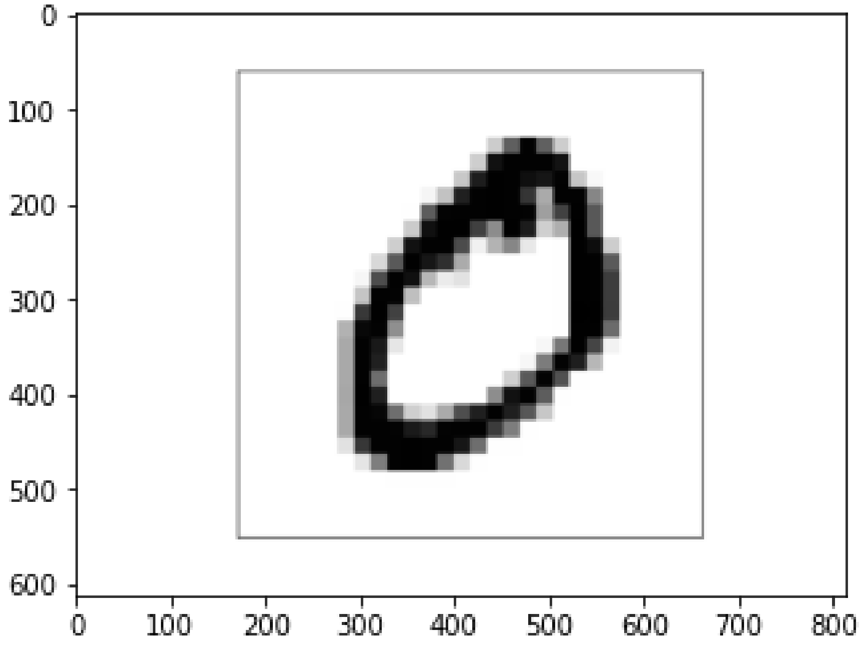
c) changing various parameters (here: number of epochs), adding batch normalization and usin g different weight initializers ***("clubbed part d, e and f of question 2"):***

"Model Summary"

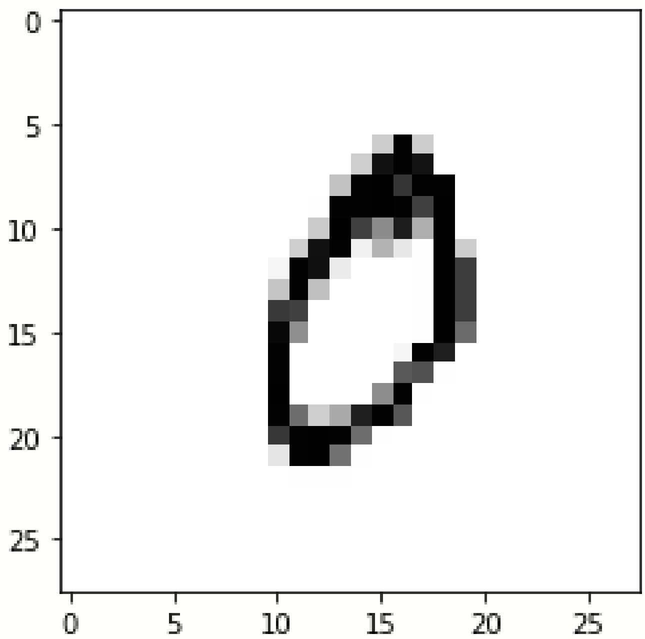
```
Model: "sequential_13"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_26 (Conv2D)          (None, 26, 26, 32)        320

 batch_normalization_4 (Batc  (None, 26, 26, 32)       128
 hNormalization)

 max_pooling2d_23 (MaxPoolin  (None, 13, 13, 32)       0
 g2D)

 conv2d_27 (Conv2D)          (None, 11, 11, 64)        18496

 batch_normalization_5 (Batc  (None, 11, 11, 64)       256
 hNormalization)

 max_pooling2d_24 (MaxPoolin  (None, 5, 5, 64)         0
 g2D)

 flatten_8 (Flatten)         (None, 1600)              0

 dropout_8 (Dropout)         (None, 1600)              0

 dense_16 (Dense)            (None, 64)                102464

 dense_17 (Dense)            (None, 1)                 65

=================================================================
Total params: 121,729
Trainable params: 121,537
Non-trainable params: 192
```

3) For Inference (using pretrained custom CNN model) following are the proposed solutions:

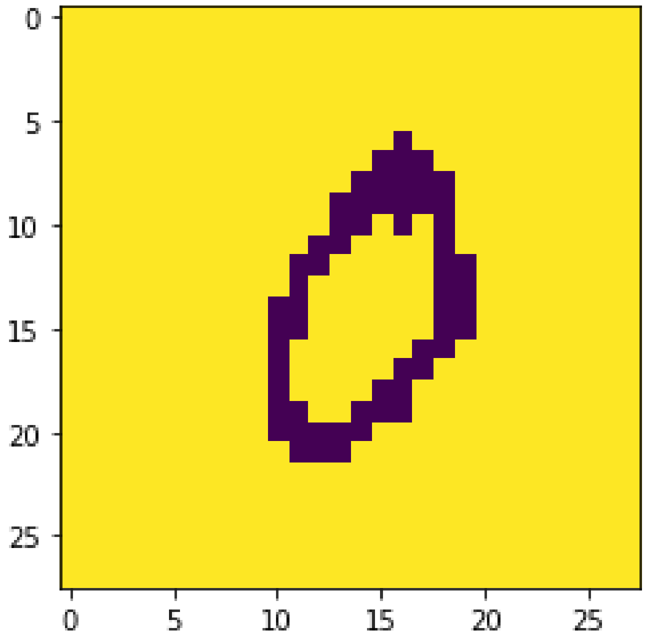
<u>"Input image: Handwritten digit"</u>



<u>"Resized Image"</u>



“Binary Image”



4) Multiclass classification using cifar10 dataset with various variations in the basic model by hyperparameter tuning, adding inception blocks and residual blocks:

a) Basic CNN

“Model Summary”

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_9 (Conv2D)           (None, 32, 32, 32)        896

 batch_normalization_3 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d_7 (MaxPooling  (None, 16, 16, 32)       0
 2D)

 conv2d_10 (Conv2D)          (None, 16, 16, 64)        18496

 batch_normalization_4 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_8 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 conv2d_11 (Conv2D)          (None, 8, 8, 128)         73856

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
```

```
 hNormalization)

 max_pooling2d_9 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 flatten_3 (Flatten)         (None, 2048)              0

 dense_3 (Dense)             (None, 10)                20490

=================================================================
Total params: 114,634
Trainable params: 114,186
Non-trainable params: 448
```

b) CNN model with Hyperparameter tuning (increase number of epochs to 15) and changing optimizer='adam':

"Model Summary"

```
Model: "sequential_5"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_15 (Conv2D)          (None, 32, 32, 32)        896

 batch_normalization_9 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d_13 (MaxPoolin  (None, 16, 16, 32)       0
 g2D)

 conv2d_16 (Conv2D)          (None, 16, 16, 64)        18496

 batch_normalization_10 (Bat  (None, 16, 16, 64)       256
 chNormalization)

 max_pooling2d_14 (MaxPoolin  (None, 8, 8, 64)         0
 g2D)

 conv2d_17 (Conv2D)          (None, 8, 8, 128)         73856

 batch_normalization_11 (Bat  (None, 8, 8, 128)        512
 chNormalization)

 max_pooling2d_15 (MaxPoolin  (None, 4, 4, 128)        0
 g2D)

 flatten_5 (Flatten)         (None, 2048)              0

 dense_5 (Dense)             (None, 10)                20490

=================================================================
Total params: 114,634
Trainable params: 114,186
Non-trainable params: 448
```

c) on adding 2 layers of inception blocks:

"Model Summary"

```
Model: "cnn_model_with_inception"
__________________________________________________________________________________________________
 Layer (type)                   Output Shape         Param #
Connected to
==================================================================================================
 input_2 (InputLayer)           [(None, 32, 32, 3)]  0            []

 conv2d_14 (Conv2D)             (None, 32, 32, 32)   896
['input_2[0][0]']

 batch_normalization_2 (BatchNo  (None, 32, 32, 32)  128
['conv2d_14[0][0]']
 rmalization)

 max_pooling2d_4 (MaxPooling2D)  (None, 16, 16, 32)  0
['batch_normalization_2[0][0]']

 conv2d_16 (Conv2D)             (None, 16, 16, 64)   2112
['max_pooling2d_4[0][0]']

 conv2d_18 (Conv2D)             (None, 16, 16, 64)   2112
['max_pooling2d_4[0][0]']

 max_pooling2d_5 (MaxPooling2D)  (None, 16, 16, 32)  0
['max_pooling2d_4[0][0]']

 conv2d_15 (Conv2D)             (None, 16, 16, 64)   2112
['max_pooling2d_4[0][0]']

 conv2d_17 (Conv2D)             (None, 16, 16, 64)   36928
['conv2d_16[0][0]']

 conv2d_19 (Conv2D)             (None, 16, 16, 64)   102464
['conv2d_18[0][0]']

 conv2d_20 (Conv2D)             (None, 16, 16, 64)   2112
['max_pooling2d_5[0][0]']

 tf.concat_2 (TFOpLambda)       (None, 16, 16, 256)  0
['conv2d_15[0][0]',

'conv2d_17[0][0]',

'conv2d_19[0][0]',

'conv2d_20[0][0]']

 conv2d_21 (Conv2D)             (None, 16, 16, 64)   147520
['tf.concat_2[0][0]']
```

```
 batch_normalization_3 (BatchNo  (None, 16, 16, 64)  256
['conv2d_21[0][0]']
 rmalization)

 max_pooling2d_6 (MaxPooling2D)  (None, 8, 8, 64)    0
['batch_normalization_3[0][0]']

 conv2d_23 (Conv2D)             (None, 8, 8, 64)     4160
['max_pooling2d_6[0][0]']

 conv2d_25 (Conv2D)             (None, 8, 8, 64)     4160
['max_pooling2d_6[0][0]']

 max_pooling2d_7 (MaxPooling2D)  (None, 8, 8, 32)    0
['max_pooling2d_4[0][0]']

 conv2d_22 (Conv2D)             (None, 8, 8, 64)     4160
['max_pooling2d_6[0][0]']

 conv2d_24 (Conv2D)             (None, 8, 8, 64)     36928
['conv2d_23[0][0]']

 conv2d_26 (Conv2D)             (None, 8, 8, 64)     102464
['conv2d_25[0][0]']

 conv2d_27 (Conv2D)             (None, 8, 8, 64)     2112
['max_pooling2d_7[0][0]']

 tf.concat_3 (TFOpLambda)       (None, 8, 8, 256)    0
['conv2d_22[0][0]',

'conv2d_24[0][0]',

'conv2d_26[0][0]',

'conv2d_27[0][0]']

 flatten_1 (Flatten)            (None, 16384)        0
['tf.concat_3[0][0]']

 dense_1 (Dense)                (None, 10)           163850
['flatten_1[0][0]']

========================================================================
===========================
Total params: 614,474
Trainable params: 614,282
Non-trainable params: 192
```

d) on adding 3 layers of residual block:

<u>"Model Definition"</u>

```
def cnn_model_with_residual_blocks(X=(32,32,3)):
  X = tf.keras.Input(shape=X)
  X1 = layers.Conv2D(32, (3, 3), activation='relu',padding='same', inpu
t_shape=(32, 32, 3))(X)
  X1 = layers.BatchNormalization()(X1)
  X1 = layers.MaxPooling2D((2, 2))(X1)

## residual layer 1 (Input X1 and Ouput Fx1)
  Fx = layers.Conv2D(filters= 32, kernel_size=(3, 3),padding='same')(X1
)
  Fx = layers.BatchNormalization(axis=3)(Fx)
  Fx = layers.Activation('relu')(Fx)
  Fx1 = layers.Add()([X1, Fx])

  X2 = layers.Conv2D(64, (3, 3), activation='relu',padding='same', inpu
t_shape=(32, 32, 3))(Fx1)
  X2 = layers.BatchNormalization()(X2)
  X2 = layers.MaxPooling2D((2, 2))(X2)

## residual layer 2 (Input X2 and Ouput Fx2)
  Fx = layers.Conv2D(filters= 64, kernel_size=(3, 3),padding='same')(X2
)
  Fx = layers.BatchNormalization(axis=3)(Fx)
  Fx = layers.Activation('relu')(Fx)
  Fx2 = layers.Add()([X2, Fx])

  X3 = layers.Conv2D(128, (3, 3), activation='relu',padding='same', inp
ut_shape=(32, 32, 3))(Fx2)
  X3 = layers.BatchNormalization()(X3)
  X3 = layers.MaxPooling2D((2, 2))(X3)

## residual layer 3 (Input X3 and Ouput Fx3)
  Fx = layers.Conv2D(filters= 128, kernel_size=(3, 3),padding='same')(X
3)
  Fx = layers.BatchNormalization(axis=3)(Fx)
  Fx = layers.Activation('relu')(Fx)
  Fx3 = layers.Add()([X3, Fx])

  Fx = layers.Activation('relu')(Fx3)
  Fx = layers.Flatten()(Fx)
  Fx = layers.Dense(10, activation='softmax')(Fx)

  model = Model(inputs = X, outputs= Fx, name='residual_block_cnn_model
')
```

```
  #compile model
  opt = SGD(learning_rate=0.001, momentum=0.9)
  model.compile(optimizer=opt, loss='categorical_crossentropy', metrics
=['accuracy'])

  return model
```

3. Implementation details:

- Used google colab (https://colab.research.google.com/) for implementing the programming questions.
- src folder has 2 '. ipynb' files:
  - Binary_Classification__Hyperparameter_Tuning_Inference.ipynb (1^{st} file)
  - Multiclass_Classification_CIFAR10.ipynb (2^{nd} file)

  So, the first file is for the implementation of Binary Classification, Hyper parameter Tuning and Inference questions and the 2^{nd} file is for Multiclass classification.
- **Note: for Hyperparameter Tuning of Binary Classification question 2**:
  - I have clubbed part b and c of question 2
  - also clubbed part d, e and f of question 2.

  Reason to club parts for hyperparameter tuning: As the accuracy of base model is very high for both train and validation set therefore, not much variation were observed while hyperparameter tuning.
- For inference question 3 code is reading input image ("mnist_sample_image.png") from gdrive.

Code to mount drive on colab:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

Note:

Few of the implementation details is added with Results and Conclusion points for easy understanding

5. Results comparison and Conclusion:

Below Table showing train and test accuracy and loss measures for different models of Binary Classifications:

Model Type	train_accuracy	test accuracy	train_loss	test loss
basic model	0.9988	0.9972	0.0040	0.0062
model_a	0.9983	0.9984	0.0051	0.0054
model_b	0.9022	0.9034	0.0000e+00	0.0000e+00
model_c	0.9983	0.9986	0.0047	0.0050

1) *For Binary Classification*

"Basic model"

Result:

Final Step Loss and Accuracy of train and test dataset:

- train_loss: 0.0048 - train_accuracy: 0.998
- val_loss: 0.0062 – test accuracy: 0.9972

Conclusion:

- Both train and validation set accuracies are pretty good along with less loss values.

2) *Hyper-parameter Tuning:*

"model_a"

a) On changing the network architecture (added one more Conv2D layer with filters = 128):

Result and Conclusion:

- As the accuracy of basic CNN model in 1st part – 99.72% on test dataset is itself very high there was very less scope of further improvements.
- On adding one more Conv2D layer filters=128 **accuracy of test dataset is going higher than train, not a good model.**

"model_b"

b) On changing stride parameters and optimizer/ loss values (clubbed part b and c of question 2):

Result and Conclusion:

- In this variation we have changed the stride parameter to (2,2) and optimizer to 'rmsprop' and loss= 'categorical_crossentropy':
  - train_loss: 0.0000e+00 - train_accuracy: 0.90
  - test loss: 0.0000e+00 - test accuracy: 0.90
- Conclusion: As the **accuracy drops, we can say that stride= (1,1), optimizer = 'adam' and loss= 'binary_crossentropy' was better.**

"model_c"

c) changing various parameters (here: number of epochs, adding batch normalization a nd using different weight initializers:
Note: (clubbed part d, e and f of question 2):

Result and Conclusion:

- In this variation we have reduced number of epochs from 5 to 3, added Batch normalization and used Xavier/GlorotUniform weight initializer:
  - train_loss: 0.0047 - train_accuracy: 0.9983
  - test loss: 0.0050 - test accuracy: 0.9986
- Conclusion:
  - As the accuracy of basic CNN model in 1st part – 99.72% on test dataset is itself very high there was very less scope of further improvements.
  - On changing model parameters **accuracy of test dataset is going higher than train, not a good model.**

3) *Inference:*

Table: accuracy and loss values for various models:

Model Type	train_accuracy	test accuracy	train_loss	test loss
basic model	0.9988	0.9972	0.0040	0.0062
model_a	0.9983	0.9984	0.0051	0.0054
model_b	0.9022	0.9034	0.0000e+00	0.0000e+00
model_c	0.9983	0.9986	0.0047	0.0050

Note: For inference problem, we will use **basic model** as the **custom pretrained CNN model** due to its high accuracy and low loss values compare to other models.

Result and Conclusion:

- Input handwritten image= "mnist_sample_image.png" is an even number and on prediction output is coming a value close to '0' hence, an even number (as per code logic for even and odd numbers).

```
## Prediction of input image
output= model.predict(input_img_for_cnn)
output → array([[0.]], dtype=float32)
```

4) *Multiclass Classification:*

Result and Conclusion:

Model Type	train_accuracy	test accuracy	train_loss	test loss
Basic model (epochs=10)	0.8482	0.1107	0.4427	2.30
'cnn_model_ht' (on increasing epochs) (epochs=15)	0.9476	0.7147	0.1473	1.4453
'cnn_model_with_inception' (epochs=10)	0.915	0.7374	0.2219	0.8753
'cnn_model_with_residual_blocks' (epochs=10)	0.8667	0.7196	0.3886	0.8886

Where;

- 'cnn_model_ht' is representation for basic cnn model with hyperparameter tuning
- 'cnn_model_with_inception' is representation for inception block model
- 'cnn_model_with_residual_blocks' is representation for residual block model.

Conclusion:

- We can say that the best performing model is: **'cnn_model_ht'**
- But if we compare for same number of epochs as per train accuracy:

'cnn_model_with_inception' > 'cnn_model_with_residual_blocks' > Basic model

5. References: -

- https://www.cs.toronto.edu/~kriz/cifar.html
- https://www.tensorflow.org/guide/keras/functional