**CS512 – AS4 - Report**

**Akshay Jain**

**Department of Computer Science**

**Illinois Institute of Technology**

**April 7, 2022**

**Abstract**

In this assignment, we have implemented Planar Camera Calibration and Review Questions.

**1. Problem Statements**

We have 4 problem statements for this assignment:

1. Camera Calibration -1
2. Camera Calibration -1

**2. Proposed solution**

*1) Feature Point Extraction*

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

page4image47395008

g2D)

page4image47395392

conv2d\_21 (Conv2D)

(None, 4, 4, 64) 18496

max\_pooling2d\_18 (MaxPoolin (None, 2, 2, 64) 0

page4image47394048

g2D)

page4image47394240

flatten\_5 (Flatten)

dropout\_5 (Dropout)

dense\_10 (Dense)

dense\_11 (Dense)

(None, 256) 0 (None, 256) 0 (None, 64) 16448 (None, 1) 65

=================================================================

page5image47396928

Total params: 35,329

page5image47397504

Trainable params: 35,329

page5image47383296

Non-trainable params: 0

page5image47405824page5image47406208

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”**

page5image47261248

Model: "sequential\_13" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param # =================================================================

conv2d\_26 (Conv2D) (None, 26, 26, 32) 320

batch\_normalization\_4 (Batc (None, 26, 26, 32) hNormalization)

128

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

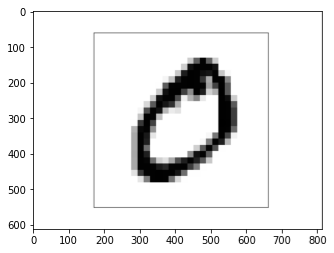
dense\_16 (Dense) (None, 64) 102464

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:*

**“Input image: Handwritten digit”**



|  |
| --- |
| **“Resized Image”**  page6image47398656 |
| page6image53901760 |

page6image47547904

**“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”**

max\_pooling2d\_7 (MaxPooling (None, 16, 16, 32) 0

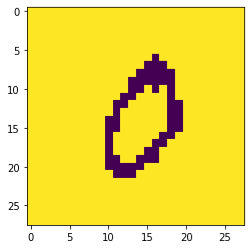
2D)

conv2d\_10 (Conv2D) (None, 16, 16, 64) 18496

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

page7image47328704

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

page7image47578112

hNormalization)

page7image47571008

batch\_normalization\_4 (Batc (None, 16, 16, 64) 256

page7image47566208

hNormalization)

page7image47568128

hNormalization)  
max\_pooling2d\_9 (MaxPooling (None, 4, 4, 128) 0

2D)

flatten\_3 (Flatten) (None, 2048) 0 dense\_3 (Dense) (None, 10) 20490

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

=================================================================

page8image47518976

Total params: 114,634

page8image47517056

Trainable params: 114,186

Non-trainable params: 448

page8image47521856

**“Model Summary”**

page8image47480832

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

page8image47481984

hNormalization)

page8image47482176

max\_pooling2d\_13 (MaxPoolin (None, 16, 16, 32) 0

page8image47482368

g2D)

page8image47482560

conv2d\_16 (Conv2D)

conv2d\_17 (Conv2D)

flatten\_5 (Flatten)

dense\_5 (Dense)

(None, 16, 16, 64)

(None, 8, 8, 128)

(None, 2048)

(None, 10)

18496

73856

0 20490

batch\_normalization\_10 (Bat (None, 16, 16, 64) 256

page8image47482944

chNormalization)

page8image47483136

max\_pooling2d\_14 (MaxPoolin (None, 8, 8, 64) 0

page8image47483328

g2D)

page8image47483520

batch\_normalization\_11 (Bat (None, 8, 8, 128) 512

page8image47483904

chNormalization)

page8image47484096

max\_pooling2d\_15 (MaxPoolin (None, 4, 4, 128) 0

page8image47484288

g2D)

page8image47484480

=================================================================

page8image47485056

Total params: 114,634

page8image47485248

Trainable params: 114,186

Non-trainable params: 448

page8image47485440

c) on adding 2 layers of inception blocks:

**“Model Summary”**

page9image47485824

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

page9image47487552

['input\_2[0][0]']

page9image47487744

batch\_normalization\_2 (BatchNo (None, 32, 32, 32) 128

page9image47487936

['conv2d\_14[0][0]']

page9image47488128

rmalization)

page9image47488320

max\_pooling2d\_4 (MaxPooling2D) (None, 16, 16, 32) 0

page9image47488512

['batch\_normalization\_2[0][0]']

page9image47488704

conv2d\_16 (Conv2D) (None, 16, 16, 64) 2112

page9image47488896

['max\_pooling2d\_4[0][0]']

page9image47489088

conv2d\_18 (Conv2D) (None, 16, 16, 64) 2112

page9image47489280

['max\_pooling2d\_4[0][0]']

page9image47489472

max\_pooling2d\_5 (MaxPooling2D) (None, 16, 16, 32) 0

page9image47489664

['max\_pooling2d\_4[0][0]']

page9image47489856

conv2d\_15 (Conv2D) (None, 16, 16, 64) 2112

page9image47490048

['max\_pooling2d\_4[0][0]']

page9image47490240

conv2d\_17 (Conv2D) (None, 16, 16, 64) 36928

page9image47490432

['conv2d\_16[0][0]']

page9image47490624

conv2d\_19 (Conv2D) (None, 16, 16, 64) 102464

page9image47490816

['conv2d\_18[0][0]']

page9image47491008

conv2d\_20 (Conv2D) (None, 16, 16, 64) 2112

page9image47491200

['max\_pooling2d\_5[0][0]']

page9image47491392

tf.concat\_2 (TFOpLambda) (None, 16, 16, 256) 0

page9image47491584

['conv2d\_15[0][0]',

page9image47491776

'conv2d\_17[0][0]',

'conv2d\_19[0][0]',

'conv2d\_20[0][0]']

conv2d\_21 (Conv2D) (None, 16, 16, 64) 147520

page9image47492544

['tf.concat\_2[0][0]']

page9image47492736

batch\_normalization\_3 (BatchNo (None, 16, 16, 64) 256

page10image47493120

['conv2d\_21[0][0]']

page10image47493312

rmalization)

page10image47493504

max\_pooling2d\_6 (MaxPooling2D) (None, 8, 8, 64) 0

page10image47493696

['batch\_normalization\_3[0][0]']

page10image47493888

conv2d\_23 (Conv2D) (None, 8, 8, 64) 4160

page10image47494080

['max\_pooling2d\_6[0][0]']

page10image47494272

conv2d\_25 (Conv2D) (None, 8, 8, 64) 4160

page10image47494464

['max\_pooling2d\_6[0][0]']

page10image47494656

max\_pooling2d\_7 (MaxPooling2D) (None, 8, 8, 32) 0

page10image47494848

['max\_pooling2d\_4[0][0]']

page10image47495040

conv2d\_22 (Conv2D) (None, 8, 8, 64) 4160

page10image47495232

['max\_pooling2d\_6[0][0]']

page10image47495424

conv2d\_24 (Conv2D) (None, 8, 8, 64) 36928

page10image47495616

['conv2d\_23[0][0]']

page10image47495808

conv2d\_26 (Conv2D) (None, 8, 8, 64) 102464

page10image47496000

['conv2d\_25[0][0]']

page10image47496192

conv2d\_27 (Conv2D) (None, 8, 8, 64) 2112

page10image47496384

['max\_pooling2d\_7[0][0]']

page10image47496576

tf.concat\_3 (TFOpLambda) (None, 8, 8, 256) 0

page10image47496768

['conv2d\_22[0][0]',

page10image47496960

'conv2d\_24[0][0]',

'conv2d\_26[0][0]',

'conv2d\_27[0][0]']

flatten\_1 (Flatten) (None, 16384) 0

page10image47571584

['tf.concat\_3[0][0]']

page10image47464448

dense\_1 (Dense) (None, 10) 163850

page10image47464640

['flatten\_1[0][0]']

page10image47464832

=======================================================================

page10image47465024

===========================

page10image47465216

Total params: 614,474

page10image47465408

Trainable params: 614,282

Non-trainable params: 192

page10image47465600

d) on adding 3 layers of residual block:

**“Model Definition”**

page11image47468096

|  |
| --- |
| 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 (1st file)
  + Multiclass\_Classification\_CIFAR10.ipynb (2nd file)

So, the first file is for the implementation of Binary Classification, Hyper parameter Tuning

and Inference questions and the 2nd 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:**

***Note:***

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

page12image47466560page12image47467136page12image47466176

from google.colab import drive

page12image47466752

drive.mount('/content/gdrive')

page12image47465792

**5. Results comparison and Conclusion:**

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

page13image47507776page13image47501056page13image47499136page13image47501248page13image47500096

**Model Type**

basic model model\_a model\_b model\_c

**train\_accuracy test accuracy train\_loss**

0.9988 0.9972 0.0040 0.9983 0.9984 0.0051 0.9022 0.9034 0.0000e+00 0.9983 0.9986 0.0047

**test loss**

0.0062 0.0054 0.0000e+00 0.0050

page13image47498752page13image47502016page13image47497408page13image47499328page13image47500288page13image47498368page13image47500480page13image47501824page13image47501440page13image47502976page13image47508544page13image47503936page13image47504704page13image47507008page13image47509120page13image47506624page13image47508352page13image47504128page13image47508736page13image47508928page13image47509312page13image47509504page13image47509696page13image47509888page13image47510080

*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:**

page13image47510464page13image47510656

• 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.**

page13image47511040page13image47511424

**“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.**

page14image47511808

3) *Inference:***Table: accuracy and loss values for various models:**

page14image47512000page14image47512192page14image47512384page14image47512576page14image47512768page14image47512960

Model Type basic model model\_a model\_b model\_c

train\_accuracy test accuracy 0.9988 0.9972 0.9983 0.9984 0.9022 0.9034 0.9983 0.9986

train\_loss test loss 0.0040 0.0062 0.0051 0.0054 0.0000e+00 0.0000e+00 0.0047 0.0050

page14image47513152page14image47513344page14image47448064page14image47448256page14image47448448page14image47448640page14image47448832page14image47449024page14image47449216page14image47449408page14image47449600page14image47449792page14image47449984page14image47450176page14image47450368page14image47450560page14image47450752page14image47450944page14image47451136page14image47451328page14image47451520page14image47451712page14image47451904page14image47452096page14image47452288

**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).

page14image47452480

## Prediction of input image

output= model.predict(input\_img\_for\_cnn) output →

array([[0.]], dtype=float32)

4) *Multiclass Classification:* **Result and Conclusion:**

Model Type  
Basic model (epochs=10)

‘cnn\_model\_ht’ (on increasing epochs)

(epochs=15)

‘cnn\_model\_with\_inception’

(epochs=10)

Where;

train\_accuracy test  
accuracy loss

page15image47453440page15image47454016page15image47454208page15image47454400page15image47454592page15image47454784

train\_loss test

page15image47454976page15image47455168page15image47455360page15image47455552page15image47455744

0.8482 0.1107 0.9476 0.7147

0.915 0.7374 0.8667 0.7196

0.4427 2.30 0.1473 1.4453

0.2219 0.8753 0.3886 0.8886

page15image47458240page15image47458432page15image47458624page15image47458816page15image47459008page15image47459200

‘cnn\_model\_with\_residual\_blocks’ (epochs=10)

page15image47459392page15image47459584page15image47459776page15image47459968page15image47460160

➢ ‘cnn\_model\_ht’ is representation for basic cnn model with hyperparameter tuning

**Conclusion:**

page15image47460352

➢ ‘cnn\_model\_with\_inception’ is representation for inception block model ➢ ‘cnn\_model\_with\_residual\_blocks’ is representation for residual block

model.

page15image47460928

➢ 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