YANIS TAZI HOMEWORK. 2 DEEP LEARNING SYSTEMS

yt1600@nyu.edu

Question 1)

Co-adaptation is when neurons depend highly on each other . This is a very important matters because one affected neuron (receiving bad input for example) will affect all the neurons that depend on this one and this is the kind of issue leading to overfitting for example.

Internal covariate shift refers to the change in the distribution of network activations due to change in network parameters during training. To reduce this, we can use normalization at each layer so that we achieve fix distribution of inputs for every layer. One of the most common technque is to use Batch normalization.

Internal covariate shift often leads to slow training and can create non convergence

Question 2)

Train LeNet 5 :

In [56]:
```
import tensorflow as tf
from sklearn.preprocessing import StandardScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras import models, layers
import tensorflow.keras as keras
from tensorflow.keras.layers import BatchNormalization, LayerNormalizati
on
import tensorflow as tf
from tensorflow.keras.layers import Dropout
import matplotlib.pyplot as plt
if tf.test.gpu_device_name():
    print('Default GPU Device: {}'.format(tf.test.gpu_device_name()))
else:
    print("Please install GPU version of TF")
```

```
Please install GPU version of TF
```

Model 1: Standard normalization for input layer and batch normalization for hidden layers

a) Data with standard normalization

In [2]:
```
# Load dataset as train and test sets
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data
()

# Set numeric type to float32 from uint8
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize value to [0, 1]
x_train /= 255
x_test /= 255

# Standard normalization
mean_train = x_train.mean()
std_train = x_train.std()
x_train -= mean_train
x_train /= std_train

x_test -= mean_train
x_test /= std_train

# Transform lables to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Reshape the dataset into 4D array
x_train = x_train.reshape(x_train.shape[0], 28,28,1)
x_test = x_test.reshape(x_test.shape[0], 28,28,1)

x_train_std_input = x_train
x_test_std_input = x_test
```

In [3]:
```
tf.random.set_seed(17)

#Instantiate an empty model
model = Sequential()
# C1 Convolutional Layer
model.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), activatio
n='tanh', input_shape=(28,28,1), padding='same'))
model.add(BatchNormalization())
# S2 Pooling Layer
model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(1, 1), padd
ing='valid'))
model.add(BatchNormalization())
# C3 Convolutional Layer
model.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), activati
on='tanh', padding='valid'))
model.add(BatchNormalization())
# S4 Pooling Layer
model.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2), padd
ing='valid'))
model.add(BatchNormalization())
# C5 Fully Connected Convolutional Layer
model.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), activat
ion='tanh', padding='valid'))
model.add(BatchNormalization())
#Flatten the CNN output so that we can connect it with fully connected l
ayers
model.add(layers.Flatten())
# FC6 Fully Connected Layer
model.add(layers.Dense(84, activation='tanh'))
model.add(BatchNormalization())
#Output Layer with softmax activation
model.add(layers.Dense(10, activation='softmax'))

# Compile the model
model.compile(loss=keras.losses.categorical_crossentropy, optimizer='SG
D', metrics=['accuracy'])
```

While the original paper talks about applying batch norm just before the activation function, it has been found in practice that applying batch norm after the activation yields better results.

Therefore, I apply it after activation for the hidden layers

In [4]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156       
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 6)         24        
_________________________________________________________________
average_pooling2d (AveragePo (None, 27, 27, 6)         0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 27, 27, 6)         24        
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 23, 23, 16)        2416      
_________________________________________________________________
batch_normalization_2 (Batch (None, 23, 23, 16)        64        
_________________________________________________________________
average_pooling2d_1 (Average (None, 11, 11, 16)        0         
_________________________________________________________________
batch_normalization_3 (Batch (None, 11, 11, 16)        64        
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 120)         48120     
_________________________________________________________________
batch_normalization_4 (Batch (None, 7, 7, 120)         480       
_________________________________________________________________
flatten (Flatten)            (None, 5880)              0         
_________________________________________________________________
dense (Dense)                (None, 84)                494004    
_________________________________________________________________
batch_normalization_5 (Batch (None, 84)                336       
_________________________________________________________________
dense_1 (Dense)              (None, 10)                850       
=================================================================
Total params: 546,538
Trainable params: 546,042
Non-trainable params: 496
_________________________________________________________________
```

In [5]:
```
hist = model.fit(x=x_train_std_input,y=y_train, epochs=20, batch_size=12
8, validation_data=(x_test_std_input, y_test), verbose=1)
```

```
Epoch 1/20
469/469 [==============================] - 13s 27ms/step - loss: 0.2075
- accuracy: 0.9425 - val_loss: 0.0996 - val_accuracy: 0.9736
Epoch 2/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0860
- accuracy: 0.9778 - val_loss: 0.0716 - val_accuracy: 0.9795
Epoch 3/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0634
- accuracy: 0.9839 - val_loss: 0.0545 - val_accuracy: 0.9848
Epoch 4/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0514
- accuracy: 0.9867 - val_loss: 0.0542 - val_accuracy: 0.9838
Epoch 5/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0436
- accuracy: 0.9889 - val_loss: 0.0437 - val_accuracy: 0.9871
Epoch 6/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0375
- accuracy: 0.9908 - val_loss: 0.0445 - val_accuracy: 0.9865
Epoch 7/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0333
- accuracy: 0.9919 - val_loss: 0.0407 - val_accuracy: 0.9885
Epoch 8/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0297
- accuracy: 0.9929 - val_loss: 0.0359 - val_accuracy: 0.9891
Epoch 9/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0265
- accuracy: 0.9938 - val_loss: 0.0359 - val_accuracy: 0.9889
Epoch 10/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0238
- accuracy: 0.9945 - val_loss: 0.0329 - val_accuracy: 0.9901
Epoch 11/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0223
- accuracy: 0.9952 - val_loss: 0.0340 - val_accuracy: 0.9898
Epoch 12/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0203
- accuracy: 0.9959 - val_loss: 0.0306 - val_accuracy: 0.9911
Epoch 13/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0184
- accuracy: 0.9962 - val_loss: 0.0303 - val_accuracy: 0.9908
Epoch 14/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0171
- accuracy: 0.9966 - val_loss: 0.0314 - val_accuracy: 0.9901
Epoch 15/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0160
- accuracy: 0.9968 - val_loss: 0.0288 - val_accuracy: 0.9909
Epoch 16/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0149
- accuracy: 0.9973 - val_loss: 0.0317 - val_accuracy: 0.9907
Epoch 17/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0139
- accuracy: 0.9976 - val_loss: 0.0294 - val_accuracy: 0.9913
Epoch 18/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0131
- accuracy: 0.9978 - val_loss: 0.0300 - val_accuracy: 0.9909
Epoch 19/20
469/469 [==============================] - 6s 12ms/step - loss: 0.0123
- accuracy: 0.9979 - val_loss: 0.0275 - val_accuracy: 0.9917
```

```
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0113
- accuracy: 0.9983 - val_loss: 0.0268 - val_accuracy: 0.9919
```

In [6]:
```
model.save('model1_standnorm_input_batchnorm_hidden.h5')
```

In [3]:
```
model = keras.models.load_model('model1_standnorm_input_batchnorm_hidde
n.h5')
```

In [49]:
```
for i in [1,3,5,7,9,12]:
    print(model.layers[i].name)
    print()
    print('Gamma :         '+ str(model.layers[i].get_weights()[0].toli
st()))
    print()
    print('Beta :         '+ str(model.layers[i].get_weights()[1].tolis
t()))
    print ()
    print ('############################################################
########################################################################
########################################################################
########################')
    print ()
```

```
batch_normalization

Gamma :          [1.0001033544540405, 1.0001364946365356, 1.00005304813
385, 1.0000249147415161, 1.0000289678573608, 1.000061273574829]

Beta :          [-3.193265651901811e-09, -3.1261129240789387e-09, 1.074
4865441836282e-08, -1.8673049773099137e-09, -2.641025842464728e-09, -6.
1343565782578935e-09]

########################################################################
########################################################################
########################################################################
###############

batch_normalization_1

Gamma :          [1.080140471458435, 1.1022557020187378, 1.047233939170
8374, 1.026122808456421, 1.024849772453308, 1.055171012878418]

Beta :          [-0.07621415704488754, -0.02155950292944908, 0.22738882
899284363, -0.011400452814996243, 0.10398653149604797, -0.0037515745498
239994]

########################################################################
########################################################################
########################################################################
###############

batch_normalization_2

Gamma :          [1.0000356435775757, 1.0000033378601074, 1.00000834465
02686, 1.0000169277191162, 1.0000183582305908, 1.0000218152999878, 1.00
0011682510376, 1.0000137090682983, 1.0000096559524536, 1.00000619888305
66, 1.0000522136688232, 1.0000017881393433, 1.0000300407409668, 1.00001
31130218506, 1.0000200271606445, 1.0000075101852417]

Beta :          [-2.463348858228187e-09, 2.005402288673963e-09, 1.62071
37021329032e-10, 1.5322434432363252e-09, -1.3839591694875253e-09, -3.12
52422871830277e-09, -3.3732927562368786e-10, -3.91772919661193e-10, 4.1
625791702415427e-10, 1.1608524558281985e-10, 1.0016080187469356e-09, -
1.153854012336808e-09, 1.8028366577382826e-09, 2.276902222320132e-09,
3.3491312501077175e-10, -4.598384728549121e-10]

########################################################################
########################################################################
########################################################################
###############

batch_normalization_3

Gamma :          [1.0338406562805176, 1.006487250328064, 1.014200329780
5786, 1.017224907875061, 1.0104438066482544, 1.0181764364242554, 1.0087
862014770508, 1.0160397291183472, 1.005645751953125, 1.007005929946899
4, 1.047123670578003, 0.9986793994903564, 1.0332375764846802, 1.0144470
930099487, 1.0219016075134277, 1.0124233961105347]

Beta :          [-0.002855873666703701, 0.001166807720437646, -0.004663
```

181956857443, 0.004320188425481319, -0.009904230013489723, -0.011273208
074271679, 0.007533228490501642, -0.013439025729894638, -0.005260499194
264412, 0.008676453493535519, -0.0031983396038413048, 0.003110519843176
0073, 0.012730601243674755, 0.0038798220921307802, -0.00816553272306919
1, 0.002551551442593336]

###
###
###
###############

batch_normalization_4

Gamma :   [1.0056623220443726, 1.0002540349960327, 1.00180220603
94287, 1.0066707134246826, 1.0092597007751465, 1.0071388483047485, 1.00
72309970855713, 0.9957995414733887, 1.0101786851882935, 1.0164235830307
007, 1.0066404342651367, 1.0052077770233154, 1.0055514574050903, 1.0036
792755126953, 1.0043503046035767, 1.0096917152404785, 1.00645089149475
1, 1.0025393962860107, 1.0074964761734009, 1.0047550201416016, 1.000993
013381958, 0.9988912343978882, 1.0064479112625122, 0.9974077939987183,
1.0029772520065308, 1.0045174360275269, 1.0033620595932007, 1.007863759
9945068, 1.0041853189468384, 1.002315878868103, 1.0109084844589233, 1.0
090458393096924, 1.0084309577941895, 1.0023661851882935, 1.006669163703
9185, 1.0024234056472778, 1.003269076347351, 1.005169153213501, 1.00593
3165550232, 0.9983053803443909, 1.0032063722610474, 0.9978740215301514,
1.0113455057144165, 1.008239507675171, 1.0073041915893555, 1.0022174119
94934, 1.002747654914856, 0.9994053840637207, 0.9990352392196655, 1.003
086805343628, 0.9998452067375183, 1.0092865228652954, 0.998435318470001
2, 1.0023771524429321, 1.0014634132385254, 1.0023584365844727, 1.008378
7441253662, 1.013611912727356, 0.9999356269836426, 1.0093258619308472,
1.0075217485427856, 1.0061134099960327, 1.0038620233535767, 1.015642166
1376953, 1.0063326358795166, 1.002451419830322 3, 1.0096383094787598, 1.
0029706954956055, 1.0026702880859375, 1.0199156999588013, 1.00515305995
94116, 1.0050839185714722, 1.0054091215133667, 1.0023939609527588, 0.99
96649026870728, 1.013514518737793, 1.0103176832199097, 1.01385784149169
92, 0.9989141225814819, 1.0042575597763062, 0.9978541135787964, 1.00949
82385635376, 0.9994216561317444, 1.0033928155899048, 1.001430630683899,
1.0026663541793823, 1.0066559314727783, 1.0063748359680176, 1.002204418
182373, 1.007834792137146, 0.9986542463302612, 1.001767873764038, 1.004
9982070922852, 1.0009055137634277, 1.0031565427780151, 1.00262212753295
9, 1.002833366394043, 1.0056177377700806, 1.0114803314208984, 0.9991413
950920105, 0.9986771941184998, 1.005359172821045, 1.0078128576278687,
1.0053293704986572, 1.001425862312317, 1.0197887420654297, 1.0087956190
109253, 1.0069087743759155, 1.0082004070281982, 1.008539080619812, 1.01
63979530334473, 1.0013022422790527, 0.9979588985443115, 1.0011752843856
812, 1.0037761926651, 1.0104435682296753, 0.9994767904281616, 0.9976818
561553955, 1.0070868730545044, 1.0001554489135742]

Beta :   [0.001124291098676622, 0.0005901519907638431, -0.002198
457717895508, 0.00010894873412325978, -0.0019466871162876487, -0.000535
2898151613772, -0.003395838430151343 3, -0.0010184940183535218, -0.00345
3400218859315, 0.0015058420831337571, -0.0014921720139682293, 0.0012483
29528607428, -0.0004696552350651473, 2.4880162527551875e-05, 0.00301438
9891177416, -0.0017457314534112811, -0.0007751599187031388, 0.002562392
5030231476, -0.0015817326493561268, 0.002309000352397561, -0.0012567474
04113412, 0.0020975738298147917, 0.002014009281992 9123, -0.000500334601
3836563, 0.0015787516022101045, -0.0004303813620936125 5, -0.00155605922

```
91876674, 0.0005733513389714062, -0.0025358612183481455, 5.677256194758
229e-05, 0.004662738647311926, -0.00040494761196896434, -0.000196141860
21499336, -0.0009334749775007367, 0.0007417193264700472, 0.001036077970
6388712, -0.0031069170217961073, -0.0001801229373086244, -0.00023045636
771712452, 0.0010385994100943208, 0.0006547464872710407, -0.00150509621
0166812, -0.0013453575083985925, -0.0017636660486459732, -0.00174194399
73309636, 0.002555257175117731, 0.0019506033277139068, -0.0007243838626
891375, -0.0023608196061104536, 0.0013325171312317252, -0.0002909032336
9018734, -5.2844512538285926e-05, -0.001144959358498454, 0.001336176646
873355, 0.0013341348385438323, 0.0018369388999417424, 0.000185029493877
6642, -0.002749239094555378, -0.0012618439504876733, 0.0018150972900912
166, -0.0007615818758495152, 0.00012803601566702127, -0.002366545610129
8332, 0.00011361933866282925, -0.0007389850215986371, 0.000483945943415
16495, -0.0003616343019530177, -0.0005119805573485792, -0.0002927032182
9244494, -0.00016640231478959322, 0.0020666508935391903, 0.000586774316
6163564, -0.0008726856322027743, -0.0005122057627886534, -0.00118294090
4982388, -0.0011479889508336782, -0.0019359468715265393, 0.002917088801
0412455, -0.0020914669148623943, 0.0008942880085669458, -0.001271036569
9604154, -0.0011192521778866649, -9.706370474305004e-05, -0.00089256936
91708148, -0.001172817312180996, -0.0008061046828515828, 0.002108468441
2926435, 0.0013441269984468818, 0.0020949963945895433, 0.00089947861852
12433, -0.0008241339819505811, 0.00014087320596445352, 0.00083104439545
42279, -2.537005093472544e-05, 0.0010218500392511487, 0.001176875084638
5956, -0.0028767904732376337, -0.0002652867406141013, 4.538970097200945
e-05, -0.00208503776229918, -0.0010540963849052787, -0.0013663598801940
68, -0.0014983313158154488, -0.002451713662594557, -0.00058318435912951
83, 0.0027133007533848286, -0.0011870344169437885, 0.002734000794589519
5, 0.0021771600004285574, -0.0011616094270721078, 0.000269299314823001
6, -0.004542726557701826, 0.0011777520412579179, 0.0014645763440057635,
9.493681136518717e-05, -0.002649039961397648, 0.0027524407487362623, -
0.0003103430208284408, -0.0007679365808144212, 0.0005505988374352455]

##########################################################################
##########################################################################
##########################################################################
###############

batch_normalization_5

Gamma :          [1.0615262985229492, 1.039718508720398, 1.038655996322
6318, 1.0391103029251099, 1.020777702331543, 1.0288842916488647, 1.0355
119705200195, 1.0326181650161743, 1.0356547832489014, 1.03204476833343
5, 1.0745348930358887, 1.0581867694854736, 1.054201602935791, 1.0620614
290237427, 1.0565823316574097, 1.0438120365142822, 1.0563002824783325,
1.03714919090271, 1.0537210702896118, 1.0315293073654175, 1.03064179420
4712, 1.0392262935638428, 1.037245273590088, 1.056341096832275, 1.0364
030599594116, 1.050891637802124, 1.062117099761963, 1.0535331964492798,
1.055459260940552, 1.0582879781723022, 1.0416603088378906, 1.025170207
0236206, 1.0372323989868164, 1.0637003183364868, 1.0374497175216675, 1.
0296249389648438, 1.0574452877044678, 1.0502383708953857, 1.06677579879
76074, 1.0457359552383423, 1.057141900062561, 1.0459693670272827, 1.056
2243461608887, 1.0574266910552979, 1.0156569480895996, 1.05148971080780
03, 1.0520153045654297, 1.0462865829467773, 1.0244661569595337, 1.03965
43741226196, 1.049127459526062, 1.0304591655731201, 1.0449737310409546,
1.034093976020813, 1.0514894723892212, 1.0646445751190186, 1.0314774513
24463, 1.0358623266220093, 1.0460479259490967, 1.0569781064987183, 1.06
28811120986938, 1.0610231161117554, 1.0231566429138184, 1.0534456968307
```

```
495, 1.0600019693374634, 1.0535024404525757, 1.043460488319397, 1.02035
20059585571, 1.0438870191574097, 1.0487699508666992, 1.021136164665222
2, 1.0495728254318237, 1.0329267978668213, 1.0658795833587646, 1.036788
7020111084, 1.065355896949768, 1.0290696620941162, 1.028918743133545,
1.0448110103607178, 1.044732689857483, 1.0588737726211548, 1.0495487451
553345, 1.041137933731079, 1.0506408214569092]

Beta :          [-0.0006841294816695154, 7.09985542926006e-05, -0.00735
7184309512377, 0.009201510809361935, 0.0005656683933921158, -0.00413913
931697607, 0.0047630551271140575, 0.007424724753946066, -0.002051409101
113677, 0.0039805080741643906, -0.0006112591945566237, -0.0052905264310
53877, -0.0018889709608629346, -0.004128921311348677, -0.00392347527667
8801, -0.00962114054709673, -8.10254059615545e-05, 0.002112170215696096
4, 0.003959633409976959, -0.0020060292445123196, 0.010553175583481789,
0.009844976477324963, -0.007894470356404781, -0.00780536700040102, 0.00
7520067505538464, 0.01277607399970293, -0.005623600445687771, -0.014072
90156930685, -0.0021203721407800913, -0.013125053606927395, -0.00855076
6855478287, -0.006940011400729418, 0.003943008370697498, 0.002421098761
2605095, -0.0057405284605920315, 0.0026334745343774557, 0.0031888689845
8004, -0.0005172445089556277, 0.005010656546801329, -0.0028656867798417
807, 0.004122504033148289, -0.0031393086537718773, 0.000797886052168905
7, -0.00093594950158149, -0.0023288445081561804, -0.007952450774610043,
-0.0017311711562797427, 0.0003689189616125077, 0.004880804568529129, 0.
0024659589398652315, 0.0005395978223532438, -0.0014985940651968122, -0.
011738463304936886, 0.010742729529738426, -0.0016814022092148662, 0.002
0558559335768223, 0.006316308863461018, -0.005518085788935423, 0.008858
502842485905, 0.0036259950138628483, -0.0023634417448192835, 0.00163141
26551151276, 0.007857171818614006, 0.0055394042283296585, 0.00921817030
7576656, -0.010433212853968143, 0.004861378576606512, 0.003201061626896
262, 0.006094268523156643, -0.006958217825740576, -0.00235053035430610
2, 0.007631663233041763, 0.0004118015931453556, -0.0002614204131532460
5, -0.0048215556889772415, 0.011411644518375397, 0.00257382751442492, -
0.00251532974652946, -0.013403575867414474, 0.007119462359696627, 0.005
5313087068498135, -0.010606273077428341, -0.0036545847542583942, -0.004
875462036579847]

########################################################################
########################################################################
########################################################################
###############
```

Question 3:

a) Data without standard normalization

In [7]:
```
# Load dataset as train and test sets
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data
()

# Set numeric type to float32 from uint8
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize value to [0, 1]
x_train /= 255
x_test /= 255

# Transform lables to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Reshape the dataset into 4D array
x_train = x_train.reshape(x_train.shape[0], 28,28,1)
x_test = x_test.reshape(x_test.shape[0], 28,28,1)
```

Model 2: Batch normalization for input and hidden layers

In [8]:
```
tf.random.set_seed(17)
#Instantiate an empty model
model_batch = Sequential()
# C1 Convolutional Layer
model_batch.add(BatchNormalization(input_shape=(28,28,1)))
model_batch.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), act
ivation='tanh', input_shape=(28,28,1), padding='same'))
model_batch.add(BatchNormalization())
# S2 Pooling Layer
model_batch.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(1, 1
), padding='valid'))
model_batch.add(BatchNormalization())
# C3 Convolutional Layer
model_batch.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1), ac
tivation='tanh', padding='valid'))
model_batch.add(BatchNormalization())
# S4 Pooling Layer
model_batch.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2, 2
), padding='valid'))
model_batch.add(BatchNormalization())
# C5 Fully Connected Convolutional Layer
model_batch.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1), a
ctivation='tanh', padding='valid'))
model_batch.add(BatchNormalization())
#Flatten the CNN output so that we can connect it with fully connected l
ayers
model_batch.add(layers.Flatten())
# FC6 Fully Connected Layer
model_batch.add(layers.Dense(84, activation='tanh'))
model_batch.add(BatchNormalization())
#Output Layer with softmax activation
model_batch.add(layers.Dense(10, activation='softmax'))

# Compile the model
model_batch.compile(loss=keras.losses.categorical_crossentropy, optimize
r='SGD', metrics=['accuracy'])
```

In [9]:
```
model_batch.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_6 (Batch (None, 28, 28, 1)         4         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 6)         156       
_________________________________________________________________
batch_normalization_7 (Batch (None, 28, 28, 6)         24        
_________________________________________________________________
average_pooling2d_2 (Average (None, 27, 27, 6)         0         
_________________________________________________________________
batch_normalization_8 (Batch (None, 27, 27, 6)         24        
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 23, 23, 16)        2416      
_________________________________________________________________
batch_normalization_9 (Batch (None, 23, 23, 16)        64        
_________________________________________________________________
average_pooling2d_3 (Average (None, 11, 11, 16)        0         
_________________________________________________________________
batch_normalization_10 (Batc (None, 11, 11, 16)        64        
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 7, 7, 120)         48120     
_________________________________________________________________
batch_normalization_11 (Batc (None, 7, 7, 120)         480       
_________________________________________________________________
flatten_1 (Flatten)          (None, 5880)              0         
_________________________________________________________________
dense_2 (Dense)              (None, 84)                494004    
_________________________________________________________________
batch_normalization_12 (Batc (None, 84)                336       
_________________________________________________________________
dense_3 (Dense)              (None, 10)                850       
=================================================================
Total params: 546,542
Trainable params: 546,044
Non-trainable params: 498
_________________________________________________________________
```

In [10]:
```
hist_batch = model_batch.fit(x=x_train,y=y_train, epochs=20, batch_size=
128, validation_data=(x_test, y_test), verbose=1)
```

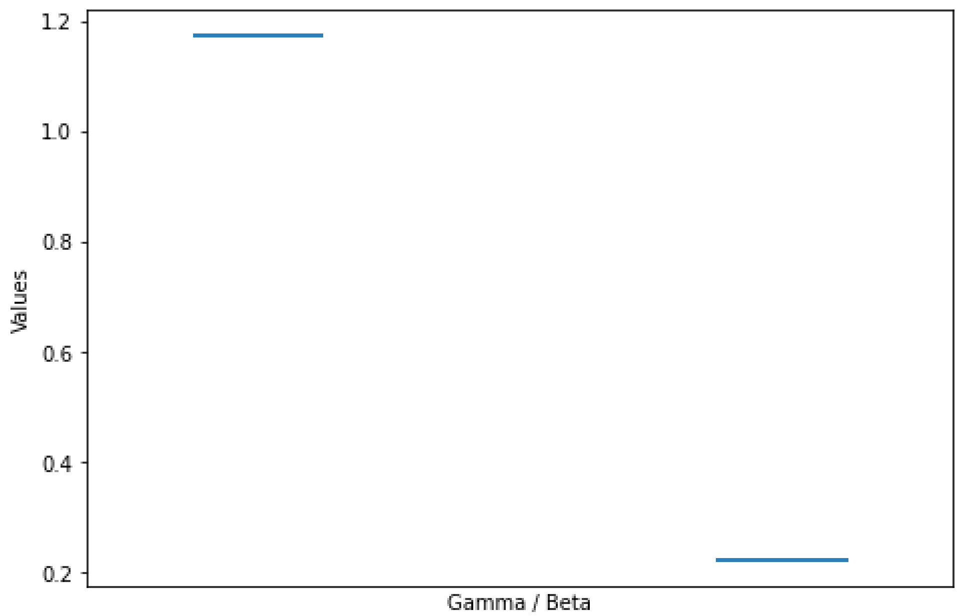
```
Epoch 1/20
469/469 [==============================] - 7s 15ms/step - loss: 0.2065
- accuracy: 0.9428 - val_loss: 0.0993 - val_accuracy: 0.9731
Epoch 2/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0848
- accuracy: 0.9782 - val_loss: 0.0704 - val_accuracy: 0.9804
Epoch 3/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0625
- accuracy: 0.9840 - val_loss: 0.0541 - val_accuracy: 0.9849
Epoch 4/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0507
- accuracy: 0.9867 - val_loss: 0.0535 - val_accuracy: 0.9850
Epoch 5/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0431
- accuracy: 0.9890 - val_loss: 0.0434 - val_accuracy: 0.9871
Epoch 6/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0370
- accuracy: 0.9909 - val_loss: 0.0435 - val_accuracy: 0.9868
Epoch 7/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0329
- accuracy: 0.9920 - val_loss: 0.0402 - val_accuracy: 0.9886
Epoch 8/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0294
- accuracy: 0.9929 - val_loss: 0.0360 - val_accuracy: 0.9887
Epoch 9/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0262
- accuracy: 0.9941 - val_loss: 0.0357 - val_accuracy: 0.9888
Epoch 10/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0235
- accuracy: 0.9947 - val_loss: 0.0328 - val_accuracy: 0.9901
Epoch 11/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0220
- accuracy: 0.9952 - val_loss: 0.0341 - val_accuracy: 0.9895
Epoch 12/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0201
- accuracy: 0.9958 - val_loss: 0.0307 - val_accuracy: 0.9912
Epoch 13/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0182
- accuracy: 0.9963 - val_loss: 0.0301 - val_accuracy: 0.9910
Epoch 14/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0169
- accuracy: 0.9967 - val_loss: 0.0313 - val_accuracy: 0.9901
Epoch 15/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0158
- accuracy: 0.9970 - val_loss: 0.0285 - val_accuracy: 0.9915
Epoch 16/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0148
- accuracy: 0.9972 - val_loss: 0.0317 - val_accuracy: 0.9901
Epoch 17/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0137
- accuracy: 0.9976 - val_loss: 0.0291 - val_accuracy: 0.9913
Epoch 18/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0130
- accuracy: 0.9978 - val_loss: 0.0300 - val_accuracy: 0.9911
Epoch 19/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0121
- accuracy: 0.9979 - val_loss: 0.0272 - val_accuracy: 0.9916
```

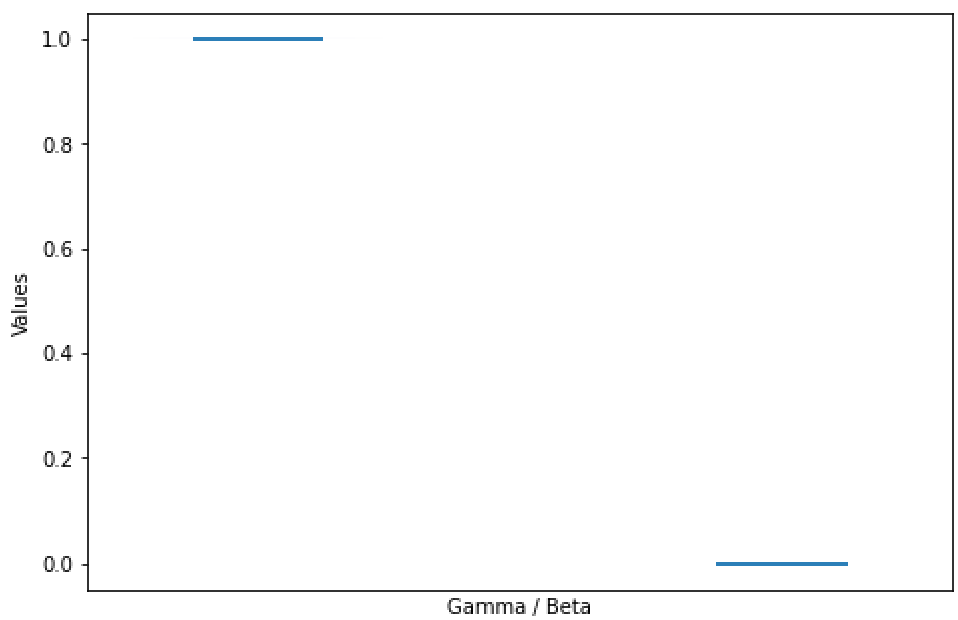
```
Epoch 20/20
469/469 [==============================] - 6s 13ms/step - loss: 0.0112
- accuracy: 0.9983 - val_loss: 0.0266 - val_accuracy: 0.9920
```

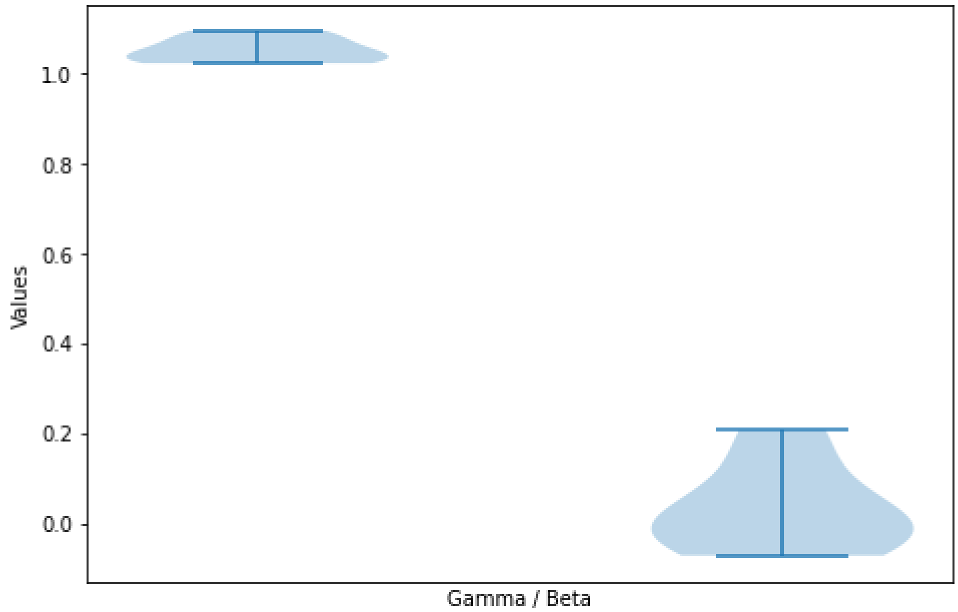
In [11]:
```
model_batch.save('model2_batchnorm_all.h5')
```

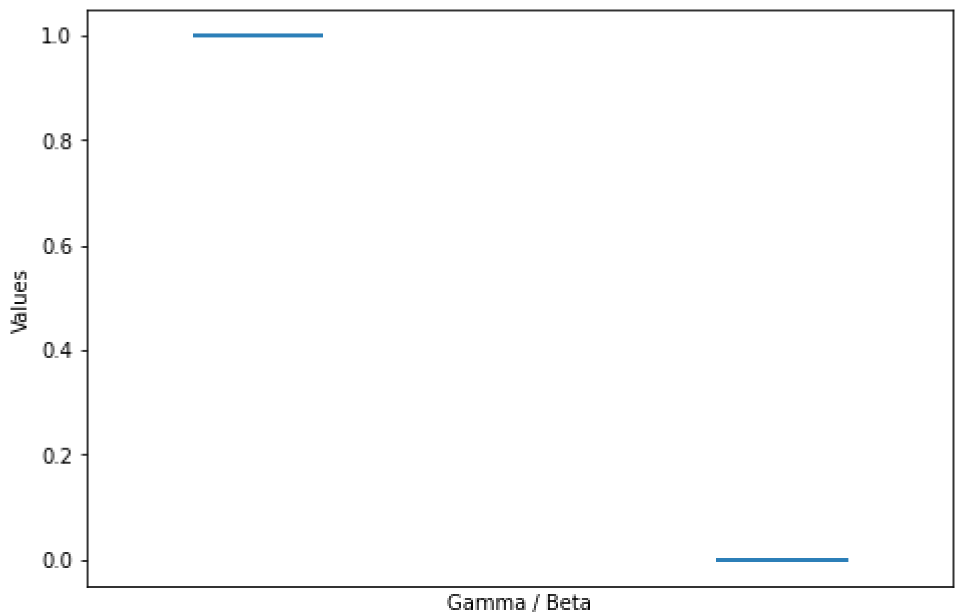
In [50]:
```
model_batch = keras.models.load_model('model2_batchnorm_all.h5')
```

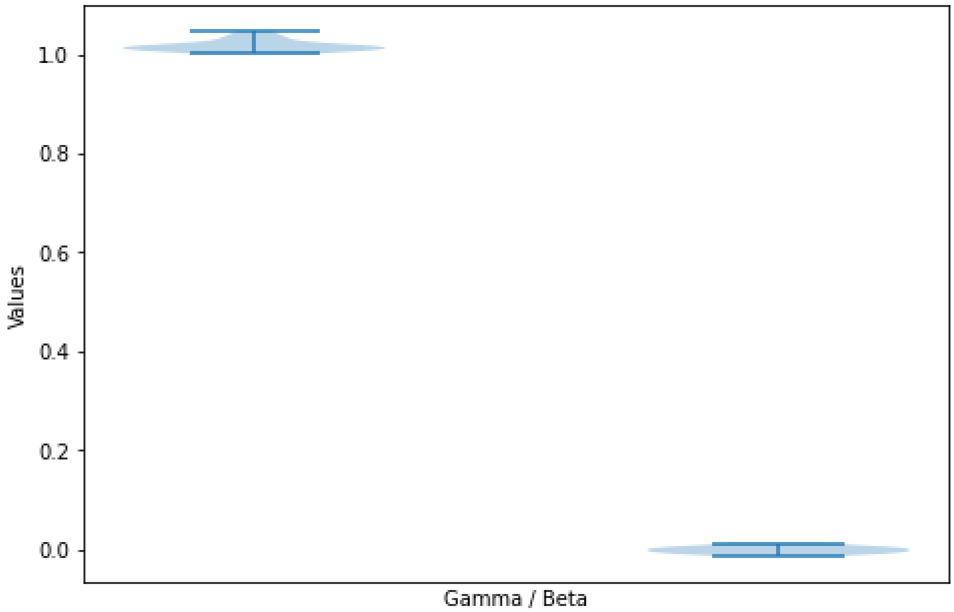
In [66]:
```
for i in [0,2,4,6,8,10,13]:
    data_to_plot = [model_batch.layers[i].get_weights()[0].tolist(), mod
el_batch.layers[i].get_weights()[1].tolist()]

    # Create a figure instance
    fig = plt.figure()

    # Create an axes instance
    ax = fig.add_axes([0,0,1,1])

    # Create the boxplot
    bp = ax.violinplot(data_to_plot)
    ax.axes.get_xaxis().set_ticks([])
    plt.xlabel("Gamma / Beta")
    plt.ylabel("Values")
    plt.show()
```

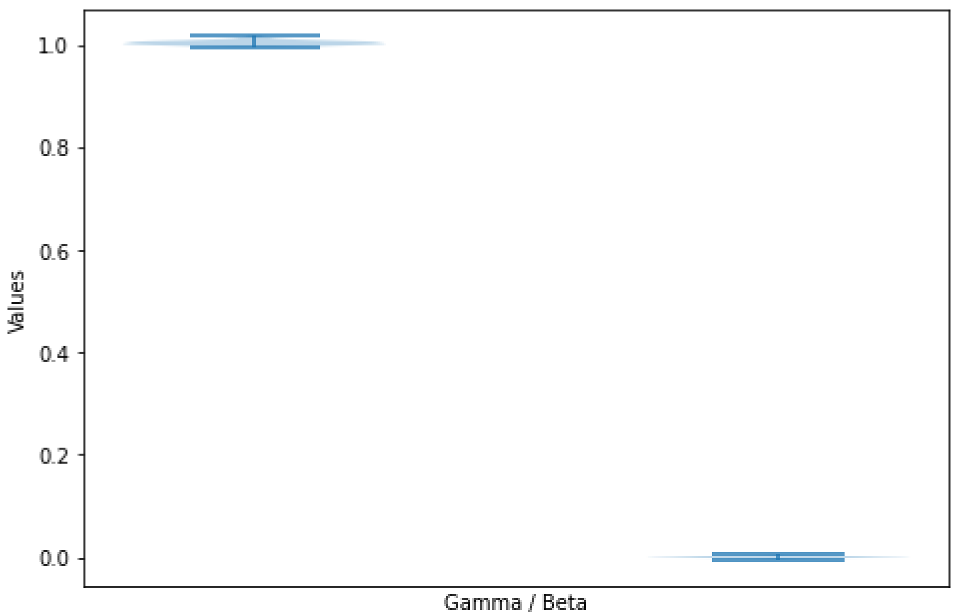

1.2
1.0
0.8
0.6
0.4
0.2
Values
Gamma / Beta

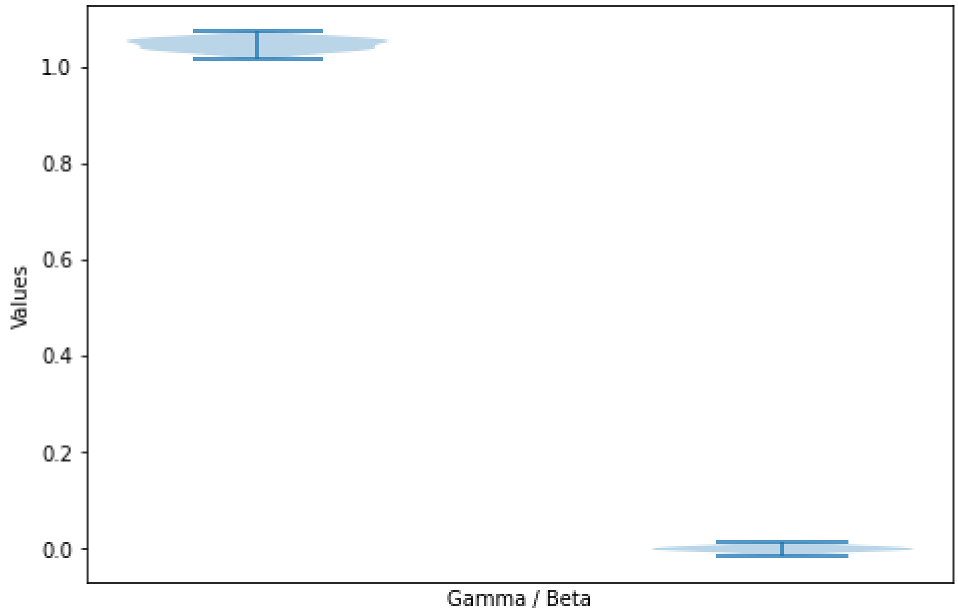

1.0
0.8
0.6
0.4
0.2
0.0
Values
Gamma / Beta


1.0
0.8
0.6
0.4
0.2
0.0
Values
Gamma / Beta


1.0
0.8
0.6
0.4
0.2
0.0
Values
Gamma / Beta


1.0
0.8
0.6
0.4
0.2
0.0
Values
Gamma / Beta


1.0
0.8
0.6
0.4
0.2
0.0
Values
Gamma / Beta


1.0
0.8
0.6
0.4
0.2
0.0
Values
Gamma / Beta

In [61]:
```
for i in [0,2,4,6,8,10,13]:
    print(model_batch.layers[i].name)
    print()
    print('Gamma :          '+ str(model_batch.layers[i].get_weights()[0].tolist()))
    print()
    print('Beta :          '+ str(model_batch.layers[i].get_weights()[1].tolist()))
    print ()
    print ('####################################################################################################################################################################################################################################')
    print ()
```

```
batch_normalization_6

Gamma :          [1.1744343042373657]

Beta :          [0.22457078099250793]

#######################################################################
#######################################################################
#######################################################################
###############

batch_normalization_7

Gamma :          [1.0000983476638794, 1.0001249313354492, 1.00004518032
07397, 1.000023365020752, 1.0000228881835938, 1.0000643730163574]

Beta :          [-7.3766730501745315e-09, -4.5491428402044676e-09, 1.07
02621427993719e-10, 4.27229318589184e-09, -4.696779853929911e-09, -1.12
79193135038668e-09]

#######################################################################
#######################################################################
#######################################################################
###############

batch_normalization_8

Gamma :          [1.0776439905166626, 1.0955561399459839, 1.04287016391
75415, 1.0234375, 1.0232908725738525, 1.0513699054718018]

Beta :          [-0.07066694647073746, -0.01616567187011242, 0.20733587
443828583, -0.008760466240346432, 0.0963069275021553, -0.00071161740925
16303]

#######################################################################
#######################################################################
#######################################################################
###############

batch_normalization_9

Gamma :          [1.0000356435775757, 1.0000054836273193, 1.00000786781
31104, 1.0000122785568237, 1.0000206232070923, 1.0000176429748535, 1.00
00079870224, 1.0000152587890625, 1.0000146627426147, 1.000005841255188,
1.000051498413086, 1.000001072883606, 1.0000293254852295, 1.00001811981
20117, 1.0000125169754028, 1.0000038146972656]

Beta :          [-5.601688002343508e-10, -3.754273281142417e-11, 7.4549
38333317784e-10, 1.3708636448228617e-09, -3.3621125883342984e-09, -2.12
63983907005013e-09, -5.515064793737423e-11, 8.792996886164417e-10, 1.79
04006055502464e-09, -2.0890402741002845e-09, -1.368022362058241e-09, 2.
1246681497499864e-11, 1.4884056209751861e-09, 7.725603490271737e-10, 3.
6389211643950148e-09, 1.5337006109561457e-09]

#######################################################################
#######################################################################
#######################################################################
```

###############

batch_normalization_10

Gamma : [1.0341215133666992, 1.0039653778076172, 1.01289129257
20215, 1.0154327154159546, 1.013261079788208, 1.0165857076644897, 1.009
5893144607544, 1.0178035497665405, 1.007718801498413, 1.006762385368347
2, 1.047182559967041, 1.0002268552780151, 1.0327712297439575, 1.0152435
302734375, 1.020142674446106, 1.010878086090088]

Beta : [-0.0045372662134468555, 0.0005203681066632271, -0.0035
58523254469037, 0.004170415457338095, -0.009523607790470123, -0.0110984
8078340292, 0.006995310075581074, -0.01202782429754734, -0.005124170798
808336, 0.008881361223757267, -0.0030007953755557537, 0.002937089651823
044, 0.012201976031064987, 0.0034679649397730827, -0.00696176895871758
5, 0.0014267554506659508]

###
###
###
###############

batch_normalization_11

Gamma : [1.0052587985992432, 1.000139832496643, 1.001922845840
454, 1.006265640258789, 1.008581280708313, 1.0073052644729614, 1.007601
3803482056, 0.9958900213241577, 1.0108489990234375, 1.0162434577941895,
1.0070840120315552, 1.005264163017273, 1.0054093599319458, 1.0031290054
32129, 1.004520058631897, 1.009005069732666, 1.0070281028747559, 1.0021
543502807617, 1.007168173789978, 1.0047032833099365, 1.000771999359130
9, 0.9994423389434814, 1.0065604448318481, 0.9974638819694519, 1.002911
8061065674, 1.0043927431106567, 1.0034414529800415, 1.0074909925460815,
1.0040936470031738, 1.0027133226394653, 1.0112006664276123, 1.008933067
3217773, 1.00880765914917, 1.0024033784866333, 1.0062967538833618, 1.00
24409294128418, 1.003140926361084, 1.0053129196166992, 1.00642943382263
18, 0.9981534481048584, 1.003822889328003, 0.9976579546928406, 1.01090
71731567383, 1.0082029104232788, 1.0073271989822388, 1.001951098442077
6, 1.0029511451721191, 0.9991244673728943, 0.9989830255508423, 1.003076
195716858, 1.0000693798065186, 1.0088599920272827, 0.9984415769577026,
1.0024769306182861, 1.0010745525360107, 1.001888632774353, 1.0082155466
079712, 1.013014793395996, 0.9997944831848145, 1.0093351602554321, 1.00
7354736328125, 1.0060356855392456, 1.0035005807876587, 1.01493799686431
88, 1.006460428237915, 1.002143383026123, 1.0090845823287964, 1.0028389
692306519, 1.0026636123657227, 1.0194510221481323, 1.0047345161437988,
1.0056579113006592, 1.005800485610962, 1.0022318363189697, 0.9996469020
843506, 1.0135589838027954, 1.0101220607757568, 1.0128202438354492, 0.9
995080232620239, 1.0044492483139038, 0.9978495836257935, 1.009016275405
8838, 0.9997790455818176, 1.0031810998916626, 1.0014065504074097, 1.002
4688243865967, 1.0062905550003052, 1.0067492723464966, 1.00246882438659
67, 1.0076700448989868, 0.9985764026641846, 1.0017824172973633, 1.00503
14664840698, 1.000797986984253, 1.0034449100494385, 1.0026159286499023,
1.0024282932281494, 1.0055179595947266, 1.011406660079956, 0.9994918704
032898, 0.9990472197532654, 1.0051100254058838, 1.0086188316345215, 1.0
054848194122314, 1.0013679265975952, 1.0197211503982544, 1.009021401405
3345, 1.0065478086471558, 1.0085139274597168, 1.0084258317947388, 1.015
3049230575562, 1.0012198686599731, 0.9983121156692505, 1.00108599662780
76, 1.0039886236190796, 1.0109390020370483, 0.9997484087944031, 0.99752

71821022034, 1.007455825805664, 1.0003316402435303]

Beta :   [0.0010533033637329936, 0.0007036763709038496, -0.00208
5362793877721, 9.090618550544605e-05, -0.001968103228136897, -0.0004403
733473736793, -0.003414415754377842, -0.0010265377350151539, -0.0035364
669747650623, 0.001471128314957023, -0.0014315377920866013, 0.00132564
24572318792, -0.00043726712465286255, 1.2391078030304925e-07, 0.0031030
38063272834, -0.0017474110936746001, -0.0007057064212858677, 0.00266171
27005010843, -0.0015560751780867577, 0.002399908611550927, -0.001198221
8129560351, 0.002085247077047825, 0.0018445991445332766, -0.00062133074
97091591, 0.0015760164242237806, -0.0005756649188697338, -0.00162180466
5774107, 0.0006325808935798705, -0.002656911965459585, 0.00015462891315
110028, 0.004566132090985775, -0.00044837422319687903, -0.0002062796411
337331, -0.0009527691872790456, 0.0006716742063872516, 0.00099207530729
47264, -0.003129907650873065, -0.00023055952624417841, -0.0002612094976
939261, 0.0010294950334355235, 0.0006142333149909973, -0.00167404580861
33003, -0.0014964313013479114, -0.0017876705387607217, -0.0016702774446
457624, 0.002510586753487587, 0.0019314270466566086, -0.000778038694988
9362, -0.0024118837900459766, 0.0012879582354798913, -0.000365159328794
10684, 0.0001681772992014885, -0.0010840485338121653, 0.001256949966773
3908, 0.001248324173502624, 0.0018012933433055878, 0.000135124937514774
5, -0.0028093259315937757, -0.0012254540342837572, 0.001892545027658343
3, -0.0006966259679757059, 0.00011096659727627411, -0.00233471300452947
6, 6.523869524244219e-05, -0.0006349883042275906, 0.000562687346246093
5, -0.0002250545658171177, -0.00048457770026288927, -0.0003365697921253
741, -0.00018297780479770154, 0.0019598889630287886, 0.0005735055310651
66, -0.0007671168423257768, -0.00064840231789276, -0.001130310934968292
7, -0.0010996715864166617, -0.0020463543478399515, 0.002863544505089521
4, -0.0020908762235194445, 0.0008373147575184703, -0.001286128303036093
7, -0.0011515046935528517, -0.0002559356507845223, -0.00089419342111796
14, -0.001217835582792759, -0.0006358709651976824, 0.00202113110572099
7, 0.001275059417821467, 0.002213583793491125, 0.0009163131471723318, -
0.0007526439148932695, 2.388357461313717e-05, 0.0007840882171876729, -
0.000145596648156643, 0.001123476424254477, 0.0012963585322722793, -0.
0029234394896775484, -0.00032991435728035867, 9.385023440700024e-05, -
0.0021514880936592817, -0.001039752154611051, -0.0013921601930633187, -
0.001326416851952672, -0.0024970988743007183, -0.0004410594410728663,
0.0027187583036720753, -0.0012052664533257484, 0.0027848673053085804,
0.002017634455114603, -0.0011310462141409516, 0.0004675565578509122, -
0.004531759303063154, 0.0012351487530395389, 0.0014310673577710986, -4.
221291237627156e-05, -0.0027097766287624836, 0.0027845948934555054, -0.
00028952330467291176, -0.0007827087538316846, 0.0005921937408857048]

##
##
##
###############

batch_normalization_12

Gamma :   [1.0612136125564575, 1.0398560762405396, 1.03955388069
15283, 1.0386760234832764, 1.020702838897705, 1.028437614440918, 1.0354
441404342651, 1.0325572490692139, 1.0350160598754883, 1.031826496124267
6, 1.0736597776412964, 1.0579160451889038, 1.0536558628082275, 1.062543
8690185547, 1.0552042722702026, 1.043561339378357, 1.0550681352615356,
1.0371156930923462, 1.0531249046325684, 1.0316314697265625, 1.030505061
1495972, 1.0384547710418701, 1.036906361579895, 1.0566296577453613, 1.0

```
364665985107422, 1.0501279830932617, 1.061496376991272, 1.0529338121414
185, 1.055050015449524, 1.057979702949524, 1.0413564443588257, 1.025724
8878479004, 1.0366038084030151, 1.063535213470459, 1.0369486808776855,
1.0294815301895142, 1.0567388534545898, 1.0504050254821777, 1.066788673
400879, 1.0452245473861694, 1.0561500787734985, 1.046491265296936, 1.05
67829608917236, 1.0565285682678223, 1.0154483318328857, 1.0512311458587
646, 1.05126953125, 1.0461827516555786, 1.0238131284713745, 1.039643287
6586914, 1.04896879196167, 1.0306458473205566, 1.0446697473526, 1.03395
676612854, 1.0511891841888428, 1.0643560886383057, 1.031791090965271,
1.0357686281204224, 1.0453524589538574, 1.0564340353012085, 1.062582373
6190796, 1.060795545578003, 1.024033784866333, 1.052892804145813, 1.059
4714879989624, 1.0530486106872559, 1.0443867444992065, 1.02026724815368
65, 1.0437109470367432, 1.0490092039108276, 1.0210005044937134, 1.04907
41729736328, 1.0338023900985718, 1.065394639968872, 1.0365632772445679,
1.0652766227722168, 1.0288386344909668, 1.0286762714385986, 1.044765353
2028198, 1.0443572998046875, 1.0587347745895386, 1.0487102270126343, 1.
0407977104187012, 1.0509847402572632]

Beta :         [-0.0008692556293681264, 0.0004808119556400925, -0.0072
137764655053616, 0.009035460650920868, 0.0006586991949006915, -0.004097
374621778727, 0.004673219751566485, 0.007280650082975626, -0.001932174
4330227375, 0.003851494286209345, 2.85836558759911e-05, -0.005384916905
31373, -0.0017548587638884783, -0.004126016516238451, -0.00391753762960
434, -0.009419938549399376, 0.0001730532676447183, 0.002150830347090959
5, 0.003596246475353837, -0.002177702495828271, 0.010159330442547798,
0.009885822422802448, -0.007434303406625986, -0.007985561154782772, 0.0
072942329570651054, 0.012714598327875137, -0.005438168533146381, -0.013
720403425395489, -0.0019007171504199505, -0.012599308975040913, -0.0081
23181760311127, -0.006509695202112198, 0.003702538087964058, 0.00215646
81082963943, -0.005515757016837597, 0.0023110629990696907, 0.0031397666
316479445, -0.000557487946934998, 0.00464083906263113, -0.0029092032928
01976, 0.0039813644252717495, -0.003198189428076148, 0.0007314523099921
644, -0.0008894737111404538, -0.0023849380668252707, -0.007846469059586
525, -0.00178744294680655, 0.000701487879268825, 0.0046749962493777275,
0.0022615070920437574, 0.00046671839663758874, -0.0012962330365553498,
-0.011212168261408806, 0.010269800201058388, -0.0014958319952711463, 0.
002203281968832016, 0.00637700455263257, -0.005733649246394634, 0.00880
1814168691635, 0.003929481841623783, -0.002303466899320483, 0.001259794
8079928756, 0.007495399098843336, 0.0052605909295380116, 0.008851370774
2095, -0.009923121891915798, 0.004494238644838333, 0.00320960395038127
9, 0.005816757213324308, -0.0069657862186431885, -0.002194028347730636
6, 0.007474885787814856, 0.00040936964796856046, -0.000433505076216533
8, -0.004530641250312328, 0.011161871254444122, 0.002604279201477766, -
0.00278839492239058, -0.012905885465443134, 0.007219281978905201, 0.005
192923359572887, -0.010496263392269611, -0.0037754857912659645, -0.0044
82236225157976]

########################################################################
########################################################################
########################################################################
###############
```

The performances have been similar .

Question4:

a) Data without standard normalization

In [12]:
```
# Load dataset as train and test sets
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data
()

# Set numeric type to float32 from uint8
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize value to [0, 1]
x_train /= 255
x_test /= 255

# Transform lables to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Reshape the dataset into 4D array
x_train = x_train.reshape(x_train.shape[0], 28,28,1)
x_test = x_test.reshape(x_test.shape[0], 28,28,1)
```

Model 3: Dropout and no batch normalization

(dropout not applied on pooling layers)

In [14]:
```
#Instantiate an empty model
model_dropout = Sequential()
# C1 Convolutional Layer
model_dropout.add(Dropout(0.2, input_shape=(28,28,1)))
model_dropout.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1, 1), a
ctivation='tanh', input_shape=(28,28,1), padding='same'))
# S2 Pooling Layer
model_dropout.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(1,
1), padding='valid'))

# C3 Convolutional Layer
model_dropout.add(Dropout(0.5))
model_dropout.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1, 1),
activation='tanh', padding='valid'))

# S4 Pooling Layer
model_dropout.add(layers.AveragePooling2D(pool_size=(2, 2), strides=(2,
2), padding='valid'))

# C5 Fully Connected Convolutional Layer
model_dropout.add(Dropout(0.5))
model_dropout.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(1, 1),
activation='tanh', padding='valid'))

#Flatten the CNN output so that we can connect it with fully connected l
ayers
model_dropout.add(layers.Flatten())

# FC6 Fully Connected Layer
model_dropout.add(Dropout(0.5))
model_dropout.add(layers.Dense(84, activation='tanh'))

#Output Layer with softmax activation
model_dropout.add(Dropout(0.5))
model_dropout.add(layers.Dense(10, activation='softmax'))

# Compile the model
model_dropout.compile(loss=keras.losses.categorical_crossentropy, optimi
zer='SGD', metrics=['accuracy'])
```

In [15]:
```
model_dropout.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dropout_5 (Dropout)          (None, 28, 28, 1)         0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 28, 28, 6)         156
_________________________________________________________________
average_pooling2d_6 (Average (None, 27, 27, 6)         0
_________________________________________________________________
dropout_6 (Dropout)          (None, 27, 27, 6)         0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 23, 23, 16)        2416
_________________________________________________________________
average_pooling2d_7 (Average (None, 11, 11, 16)        0
_________________________________________________________________
dropout_7 (Dropout)          (None, 11, 11, 16)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 7, 7, 120)         48120
_________________________________________________________________
flatten_3 (Flatten)          (None, 5880)              0
_________________________________________________________________
dropout_8 (Dropout)          (None, 5880)              0
_________________________________________________________________
dense_6 (Dense)              (None, 84)                494004
_________________________________________________________________
dropout_9 (Dropout)          (None, 84)                0
_________________________________________________________________
dense_7 (Dense)              (None, 10)                850
=================================================================
Total params: 545,546
Trainable params: 545,546
Non-trainable params: 0
_________________________________________________________________
```

In [16]:
```
hist_dropout = model_dropout.fit(x=x_train,y=y_train, epochs=20, batch_size=128, validation_data=(x_test, y_test), verbose=1)
```

```
Epoch 1/20
469/469 [==============================] - 5s 12ms/step - loss: 1.0002
- accuracy: 0.6741 - val_loss: 0.3821 - val_accuracy: 0.8858
Epoch 2/20
469/469 [==============================] - 5s 11ms/step - loss: 0.5666
- accuracy: 0.8236 - val_loss: 0.3058 - val_accuracy: 0.9061
Epoch 3/20
469/469 [==============================] - 5s 11ms/step - loss: 0.5069
- accuracy: 0.8433 - val_loss: 0.2700 - val_accuracy: 0.9205
Epoch 4/20
469/469 [==============================] - 5s 11ms/step - loss: 0.4630
- accuracy: 0.8571 - val_loss: 0.2478 - val_accuracy: 0.9299
Epoch 5/20
469/469 [==============================] - 5s 11ms/step - loss: 0.4338
- accuracy: 0.8677 - val_loss: 0.2283 - val_accuracy: 0.9333
Epoch 6/20
469/469 [==============================] - 5s 12ms/step - loss: 0.4050
- accuracy: 0.8755 - val_loss: 0.2157 - val_accuracy: 0.9369
Epoch 7/20
469/469 [==============================] - 6s 12ms/step - loss: 0.3858
- accuracy: 0.8819 - val_loss: 0.2026 - val_accuracy: 0.9407
Epoch 8/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3731
- accuracy: 0.8870 - val_loss: 0.1925 - val_accuracy: 0.9429
Epoch 9/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3556
- accuracy: 0.8914 - val_loss: 0.1830 - val_accuracy: 0.9453
Epoch 10/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3442
- accuracy: 0.8954 - val_loss: 0.1723 - val_accuracy: 0.9483
Epoch 11/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3344
- accuracy: 0.8979 - val_loss: 0.1676 - val_accuracy: 0.9492
Epoch 12/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3213
- accuracy: 0.9029 - val_loss: 0.1581 - val_accuracy: 0.9507
Epoch 13/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3091
- accuracy: 0.9066 - val_loss: 0.1519 - val_accuracy: 0.9543
Epoch 14/20
469/469 [==============================] - 5s 11ms/step - loss: 0.3021
- accuracy: 0.9086 - val_loss: 0.1465 - val_accuracy: 0.9555
Epoch 15/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2942
- accuracy: 0.9119 - val_loss: 0.1423 - val_accuracy: 0.9554
Epoch 16/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2850
- accuracy: 0.9159 - val_loss: 0.1342 - val_accuracy: 0.9581
Epoch 17/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2776
- accuracy: 0.9159 - val_loss: 0.1291 - val_accuracy: 0.9603
Epoch 18/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2658
- accuracy: 0.9186 - val_loss: 0.1247 - val_accuracy: 0.9611
Epoch 19/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2603
- accuracy: 0.9216 - val_loss: 0.1224 - val_accuracy: 0.9617
```

```
Epoch 20/20
469/469 [==============================] - 5s 11ms/step - loss: 0.2518
- accuracy: 0.9244 - val_loss: 0.1151 - val_accuracy: 0.9641
```

In [17]:
```
model_dropout.save('model3_dropout_all.h5')
```

Test accuracy has significantly dropped ! (0.9641 for dropout only vs 0.9920 for batch norm only)

Question 5 :

a) Data without standard normalization

In [18]:
```
# Load dataset as train and test sets
(x_train, y_train), (x_test, y_test) = tf.keras.datasets.mnist.load_data
()

# Set numeric type to float32 from uint8
x_train = x_train.astype('float32')
x_test = x_test.astype('float32')

# Normalize value to [0, 1]
x_train /= 255
x_test /= 255

# Transform lables to one-hot encoding
y_train = tf.keras.utils.to_categorical(y_train, 10)
y_test = tf.keras.utils.to_categorical(y_test, 10)

# Reshape the dataset into 4D array
x_train = x_train.reshape(x_train.shape[0], 28,28,1)
x_test = x_test.reshape(x_test.shape[0], 28,28,1)
```

Model 4: Dropout and batch normalization

In [19]:

```
from tensorflow.keras.layers import Dropout
#Instantiate an empty model
model_dropout_batch = Sequential()
# C1 Convolutional Layer
model_dropout_batch.add(BatchNormalization(input_shape=(28,28,1)))
model_dropout_batch.add(Dropout(0.2, input_shape=(28,28,1)))
model_dropout_batch.add(layers.Conv2D(6, kernel_size=(5, 5), strides=(1,
1), activation='tanh', input_shape=(28,28,1), padding='same'))
model_dropout_batch.add(BatchNormalization())
# S2 Pooling Layer
model_dropout_batch.add(layers.AveragePooling2D(pool_size=(2, 2), stride
s=(1, 1), padding='valid'))
model_dropout_batch.add(BatchNormalization())
# C3 Convolutional Layer
model_dropout_batch.add(Dropout(0.5))
model_dropout_batch.add(layers.Conv2D(16, kernel_size=(5, 5), strides=(1
, 1), activation='tanh', padding='valid'))
model_dropout_batch.add(BatchNormalization())
# S4 Pooling Layer
model_dropout_batch.add(layers.AveragePooling2D(pool_size=(2, 2), stride
s=(2, 2), padding='valid'))
model_dropout_batch.add(BatchNormalization())
# C5 Fully Connected Convolutional Layer
model_dropout_batch.add(Dropout(0.5))
model_dropout_batch.add(layers.Conv2D(120, kernel_size=(5, 5), strides=(
1, 1), activation='tanh', padding='valid'))
model_dropout_batch.add(BatchNormalization())
#Flatten the CNN output so that we can connect it with fully connected l
ayers
model_dropout_batch.add(layers.Flatten())
# FC6 Fully Connected Layer
model_dropout_batch.add(Dropout(0.5))
model_dropout_batch.add(layers.Dense(84, activation='tanh'))
model_dropout_batch.add(BatchNormalization())
#Output Layer with softmax activation
model_dropout_batch.add(Dropout(0.5))
model_dropout_batch.add(layers.Dense(10, activation='softmax'))

# Compile the model
model_dropout_batch.compile(loss=keras.losses.categorical_crossentropy,
optimizer='SGD', metrics=['accuracy'])
```

In [20]:
```
model_dropout_batch.summary()
```

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_13 (Batc (None, 28, 28, 1)         4         
_________________________________________________________________
dropout_10 (Dropout)         (None, 28, 28, 1)         0         
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 28, 28, 6)         156       
_________________________________________________________________
batch_normalization_14 (Batc (None, 28, 28, 6)         24        
_________________________________________________________________
average_pooling2d_8 (Average (None, 27, 27, 6)         0         
_________________________________________________________________
batch_normalization_15 (Batc (None, 27, 27, 6)         24        
_________________________________________________________________
dropout_11 (Dropout)         (None, 27, 27, 6)         0         
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 23, 23, 16)        2416      
_________________________________________________________________
batch_normalization_16 (Batc (None, 23, 23, 16)        64        
_________________________________________________________________
average_pooling2d_9 (Average (None, 11, 11, 16)        0         
_________________________________________________________________
batch_normalization_17 (Batc (None, 11, 11, 16)        64        
_________________________________________________________________
dropout_12 (Dropout)         (None, 11, 11, 16)        0         
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 7, 7, 120)         48120     
_________________________________________________________________
batch_normalization_18 (Batc (None, 7, 7, 120)         480       
_________________________________________________________________
flatten_4 (Flatten)          (None, 5880)              0         
_________________________________________________________________
dropout_13 (Dropout)         (None, 5880)              0         
_________________________________________________________________
dense_8 (Dense)              (None, 84)                494004    
_________________________________________________________________
batch_normalization_19 (Batc (None, 84)                336       
_________________________________________________________________
dropout_14 (Dropout)         (None, 84)                0         
_________________________________________________________________
dense_9 (Dense)              (None, 10)                850       
=================================================================
Total params: 546,542
Trainable params: 546,044
Non-trainable params: 498
_________________________________________________________________
```

In [22]:
```
hist_dropout_batch = model_dropout_batch.fit(x=x_train,y=y_train, epochs
=50, batch_size=128, validation_data=(x_test, y_test), verbose=1)
```

```
Epoch 1/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1473
- accuracy: 0.9561 - val_loss: 0.0587 - val_accuracy: 0.9812
Epoch 2/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1435
- accuracy: 0.9575 - val_loss: 0.0568 - val_accuracy: 0.9817
Epoch 3/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1419
- accuracy: 0.9586 - val_loss: 0.0554 - val_accuracy: 0.9827
Epoch 4/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1397
- accuracy: 0.9583 - val_loss: 0.0540 - val_accuracy: 0.9824
Epoch 5/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1377
- accuracy: 0.9592 - val_loss: 0.0525 - val_accuracy: 0.9831
Epoch 6/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1375
- accuracy: 0.9588 - val_loss: 0.0529 - val_accuracy: 0.9823
Epoch 7/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1359
- accuracy: 0.9596 - val_loss: 0.0527 - val_accuracy: 0.9832
Epoch 8/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1355
- accuracy: 0.9597 - val_loss: 0.0500 - val_accuracy: 0.9835
Epoch 9/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1310
- accuracy: 0.9604 - val_loss: 0.0508 - val_accuracy: 0.9846
Epoch 10/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1266
- accuracy: 0.9628 - val_loss: 0.0523 - val_accuracy: 0.9833
Epoch 11/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1258
- accuracy: 0.9632 - val_loss: 0.0501 - val_accuracy: 0.9849
Epoch 12/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1237
- accuracy: 0.9632 - val_loss: 0.0474 - val_accuracy: 0.9845
Epoch 13/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1240
- accuracy: 0.9635 - val_loss: 0.0490 - val_accuracy: 0.9846
Epoch 14/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1250
- accuracy: 0.9624 - val_loss: 0.0481 - val_accuracy: 0.9847
Epoch 15/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1224
- accuracy: 0.9633 - val_loss: 0.0448 - val_accuracy: 0.9852
Epoch 16/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1189
- accuracy: 0.9650 - val_loss: 0.0463 - val_accuracy: 0.9848
Epoch 17/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1179
- accuracy: 0.9652 - val_loss: 0.0449 - val_accuracy: 0.9856
Epoch 18/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1189
- accuracy: 0.9639 - val_loss: 0.0453 - val_accuracy: 0.9847
Epoch 19/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1161
- accuracy: 0.9659 - val_loss: 0.0451 - val_accuracy: 0.9850
```

```
Epoch 20/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1152
- accuracy: 0.9649 - val_loss: 0.0462 - val_accuracy: 0.9853
Epoch 21/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1150
- accuracy: 0.9657 - val_loss: 0.0436 - val_accuracy: 0.9853
Epoch 22/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1124
- accuracy: 0.9663 - val_loss: 0.0433 - val_accuracy: 0.9858
Epoch 23/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1141
- accuracy: 0.9657 - val_loss: 0.0432 - val_accuracy: 0.9855
Epoch 24/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1119
- accuracy: 0.9663 - val_loss: 0.0440 - val_accuracy: 0.9861
Epoch 25/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1092
- accuracy: 0.9671 - val_loss: 0.0417 - val_accuracy: 0.9860
Epoch 26/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1104
- accuracy: 0.9671 - val_loss: 0.0423 - val_accuracy: 0.9862
Epoch 27/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1100
- accuracy: 0.9670 - val_loss: 0.0423 - val_accuracy: 0.9863
Epoch 28/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1053
- accuracy: 0.9680 - val_loss: 0.0404 - val_accuracy: 0.9871
Epoch 29/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1067
- accuracy: 0.9680 - val_loss: 0.0423 - val_accuracy: 0.9863
Epoch 30/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1054
- accuracy: 0.9684 - val_loss: 0.0399 - val_accuracy: 0.9868
Epoch 31/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1068
- accuracy: 0.9680 - val_loss: 0.0404 - val_accuracy: 0.9868
Epoch 32/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1061
- accuracy: 0.9677 - val_loss: 0.0395 - val_accuracy: 0.9868
Epoch 33/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1042
- accuracy: 0.9682 - val_loss: 0.0401 - val_accuracy: 0.9864
Epoch 34/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1060
- accuracy: 0.9687 - val_loss: 0.0390 - val_accuracy: 0.9869
Epoch 35/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1020
- accuracy: 0.9691 - val_loss: 0.0397 - val_accuracy: 0.9870
Epoch 36/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1038
- accuracy: 0.9687 - val_loss: 0.0397 - val_accuracy: 0.9873
Epoch 37/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1030
- accuracy: 0.9693 - val_loss: 0.0394 - val_accuracy: 0.9865
Epoch 38/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1007
- accuracy: 0.9699 - val_loss: 0.0408 - val_accuracy: 0.9870
```

```
Epoch 39/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1017
- accuracy: 0.9699 - val_loss: 0.0375 - val_accuracy: 0.9880
Epoch 40/50
469/469 [==============================] - 7s 14ms/step - loss: 0.1001
- accuracy: 0.9703 - val_loss: 0.0374 - val_accuracy: 0.9883
Epoch 41/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0987
- accuracy: 0.9704 - val_loss: 0.0361 - val_accuracy: 0.9876
Epoch 42/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0978
- accuracy: 0.9704 - val_loss: 0.0370 - val_accuracy: 0.9873
Epoch 43/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0973
- accuracy: 0.9709 - val_loss: 0.0376 - val_accuracy: 0.9877
Epoch 44/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0960
- accuracy: 0.9711 - val_loss: 0.0380 - val_accuracy: 0.9873
Epoch 45/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0954
- accuracy: 0.9711 - val_loss: 0.0372 - val_accuracy: 0.9877
Epoch 46/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0984
- accuracy: 0.9712 - val_loss: 0.0363 - val_accuracy: 0.9880
Epoch 47/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0946
- accuracy: 0.9710 - val_loss: 0.0361 - val_accuracy: 0.9880
Epoch 48/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0958
- accuracy: 0.9711 - val_loss: 0.0368 - val_accuracy: 0.9881
Epoch 49/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0922
- accuracy: 0.9719 - val_loss: 0.0371 - val_accuracy: 0.9880
Epoch 50/50
469/469 [==============================] - 7s 14ms/step - loss: 0.0929
- accuracy: 0.9717 - val_loss: 0.0367 - val_accuracy: 0.9883
```

In [ ]:

```
model_dropout_batch.save('model4_dropout_batch_all.h5')
```

The performance compared to dropout only has significantly increased (0.9850 for dropout. + batch. norm vs 0.9641 for dropout only) . (compared on 20 epochs)

The performance compared to batch norm only has been slightly reduced(0.9850 vs 0.9920) . (compared on 20 epochs)

Conclusion :

Dropout seems to increase the training time but also avoid overfitting . Dropout is mostly a technique for regularization. It introduces noise into a neural network to force the neural network to learn to generalize well enough to deal with noise.

Batch normalization is mostly a technique for improving optimization and it happens that it also. introduce some noise and therefore help for regularization. As for large dataset like ours, optimization is more important than regularizationso batch normalization seems to be more important. A combination of both can also be applied.

In [ ]:

```
```