Deep Learning

Learning Graded Project

Week 5

Table of Contents

Street View Housing Number Digit Recognition

Business Context

Recognizing multi-digit numbers in photographs captured at street level is an important component of modern-day map making. A classic example of a corpus of such streetlevel photographs is Google’s Street View imagery composed of hundreds of millions of geo-located 360-degree panoramic images.

The ability to automatically transcribe an address number from a geo-located patch of pixels and associate the transcribed number with a known street address helps pinpoint, with a high degree of accuracy, the location of the building it represents. More broadly, recognizing numbers in photographs is a problem of interest to the optical character recognition community.

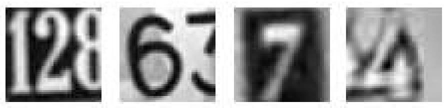
While OCR on constrained domains like document processing is well studied, arbitrary multi-character text recognition in photographs is still highly challenging. This difficulty arises due to the wide variability in the visual appearance of text in the wild on account of a large range of fonts, colours, styles, orientations, and character arrangements.

The recognition problem is further complicated by environmental factors such as lighting, shadows, specularities, and occlusions as well as by image acquisition factors such as resolution, motion, and focus blurs. In this project, we will use the dataset with images centred around a single digit (many of the images do contain some distractors at the sides). Although we are taking a sample of the data which is simpler, it is more complex than MNIST because of the distractors.

Dataset description

SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with the minimal requirement on data formatting but comes from a significantly harder, unsolved, real-world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

Some examples of the samples from this dataset are:



Where the labels for each of this image are the prominent number in that image i.e. 2,6,7 and 4 respectively.

The dataset has been provided in the form of h5py files.

1. Read the data from the h5py file and understand the train/test splits

From the h5py file after loading it to the jupiter notebook, we can read the data.

The data here contains the keys such as:

KeysViewHDF5 ['X_test', 'X_train', 'X_val', 'y_test', 'y_train', 'y_val']

In which each key represents :

X test – testing data set

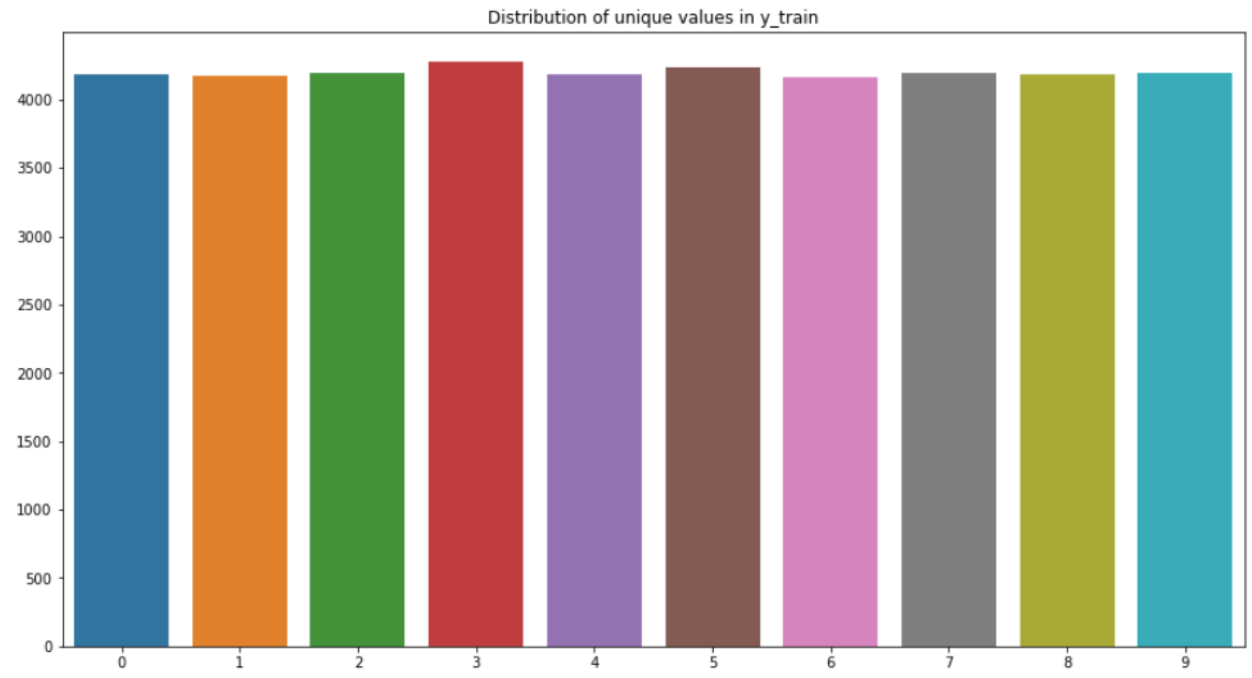
X train – training data set

X val – validation data set of independent variables

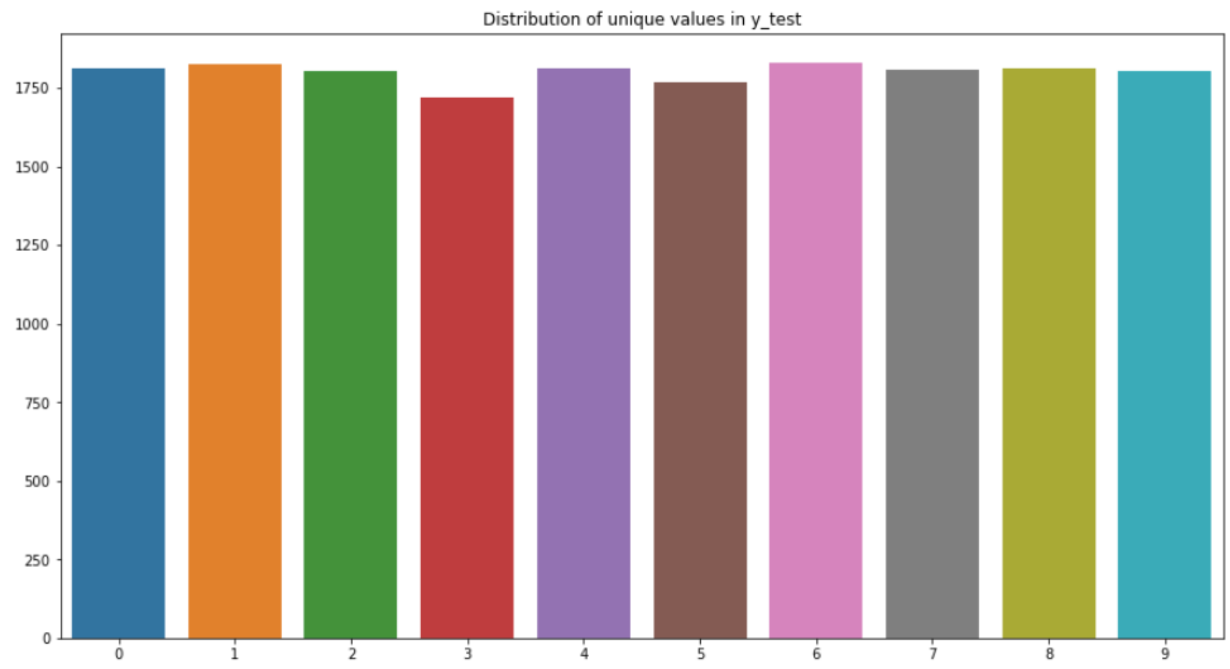
And similarly for the dependent datset.

```
X_train shape:  (42000, 32, 32)
y_train shape:  (42000,)
X_val shape:    (60000, 32, 32)
y_val shape:    (60000,)
X_test shape:   (18000, 32, 32)
y_test shape:   (18000,)
```

- The training set has 42 000 records on which we can train upon of matrix size of 32x32 i.e. image size of 32x32.
- The test set has 18 000 records on which we can train upon of matrix size of 32x32 i.e. image size of 32x32.
- y_train and y_test contain labels for given image matrix

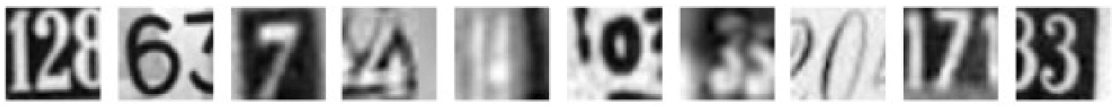


The plot above represents the distribution of label names in training set, the data is almost equaly distributed



The distribution of labels in test set is approximately the same as training set.

```
label for each of the below image: 2
label for each of the below image: 6
label for each of the below image: 7
label for each of the below image: 4
label for each of the below image: 4
label for each of the below image: 0
label for each of the below image: 3
label for each of the below image: 0
label for each of the below image: 7
label for each of the below image: 3
```



For every 32x32 image there is a respective label and it is shown in the above table.

The partial number appearing in the images is the noise.

2. Reshape and normalize the train and test features so that the same can be fed for model building. We need to feed a 2D tensor into the model and currently we have a 3D tensor.

The common practice to overcome challenges during data acquisition and model deployment is to reshape the input images so that they can be fed into the networks.

```
Shape of X_train :  (42000, 1024)
Shape of X_test  :  (60000, 1024)
Shape of X_test  :  (18000, 1024)
```

Thus we successfully flatten the data from 3d to 2d by using reshape function.

Data normalization is an important step which ensures that each input parameter (pixel, in this case) has a similar data distribution. This makes convergence faster while training the network.

The minimum value of pixel is 0 and maximum is 254.9745. After reshaping the values of the pixel are in range from 0 to 1

```
Before normalization
Max:  254.9745
Min:  0.0

After normalization
Max:  1.0
Min:  0.0
```

3. One hot encode the labels for train and test data

Each row in the array contains zeros in all columns, except the column corresponding to a unique label, which is set to 1.

```
Before one hot encoding:  0
After one hot encoding:  [1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

4. Define the model architecture using TensorFlow with a flatten layer followed by dense layers with activation as ReLu and softmax

From the above table we can see the total number of trainable and untrainable parameters from the whole network that we created.

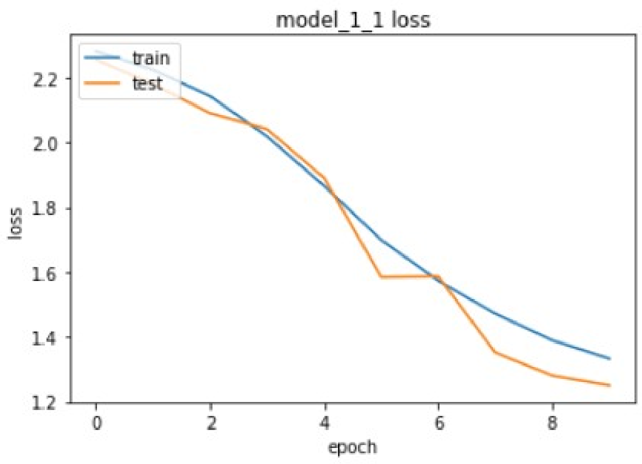
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
dense_12 (Dense)             (None, 512)               524800    
_________________________________________________________________
activation_12 (Activation)   (None, 512)               0         
_________________________________________________________________
dense_13 (Dense)             (None, 256)               131328    
_________________________________________________________________
activation_13 (Activation)   (None, 256)               0         
_________________________________________________________________
dense_14 (Dense)             (None, 128)               32896     
_________________________________________________________________
activation_14 (Activation)   (None, 128)               0         
_________________________________________________________________
dense_15 (Dense)             (None, 64)                8256      
_________________________________________________________________
activation_15 (Activation)   (None, 64)                0         
_________________________________________________________________
dense_16 (Dense)             (None, 32)                2080      
_________________________________________________________________
activation_16 (Activation)   (None, 32)                0         
_________________________________________________________________
dense_17 (Dense)             (None, 10)                330       
_________________________________________________________________
activation_17 (Activation)   (None, 10)                0         
=================================================================
Total params: 699,690
Trainable params: 699,690
Non-trainable params: 0
```

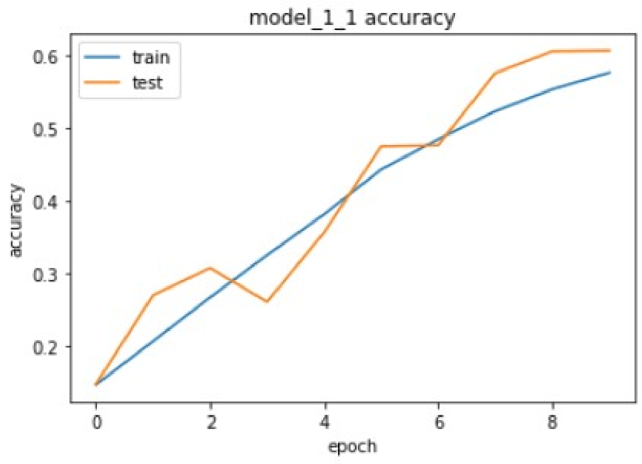
The model used in this process is a sequential model in which the model architecture is through TensorFlow by using a flatten layer of images followed by input layer, 4 hidden (size of 1st hidden layer — 512 neurons, 2nd — 256, 3rd — 128, 4th - 32) layers and last layer of output has 10 neurons.

The rectified linear activation function or ReLU is employed for first five dense layers which is a piecewise linear function that will output the input directly if it is positive, otherwise, it will output zero. It has become the default activation function for many types of neural networks because a model that uses it is easier to train and often achieves better performance.

Specifically, the network is configured to output 10 values in this case, one for each class in the classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Each value in the output of the softmax function is interpreted as the probability of membership for each class.

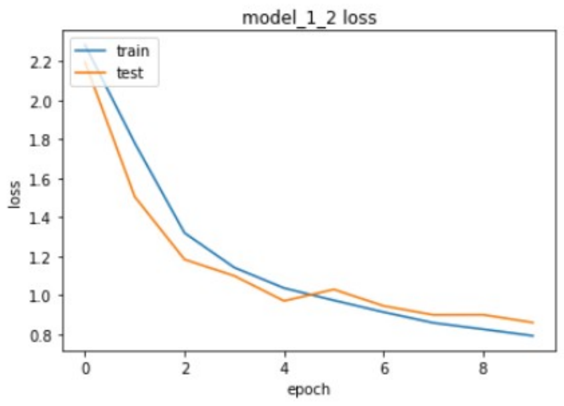
In this model we are using SGD optimizer and Categorical Crossentropy as loss function and Accuracy as the metric to monitor

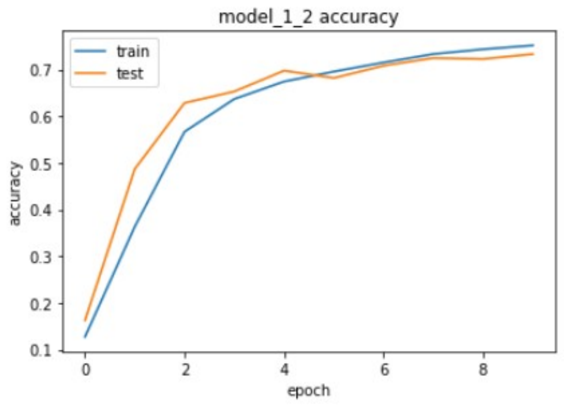




Relationship between loss and number of epoch is likely to linear. It means that model is week leaner.

5. Compile the model with loss as categorical cross-entropy and adam optimizers. Use accuracy as the metric for evaluation



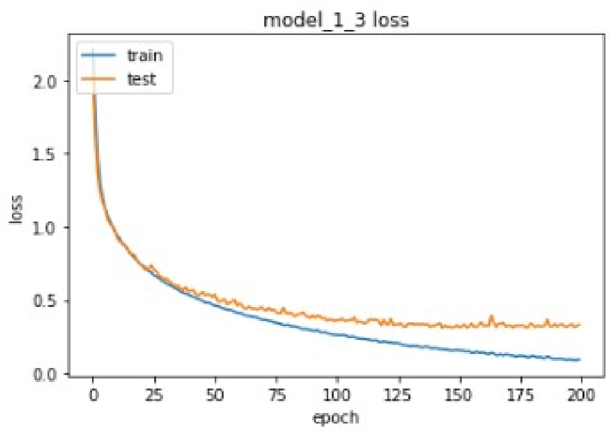


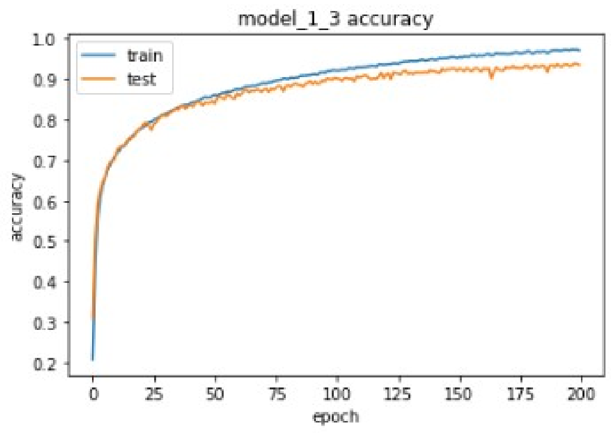
The gradient descent algorithm seeks to change the weights so that the next evaluation reduces the error, meaning the optimization algorithm is navigating down the gradient (or slope) of error

Cross-entropy loss is often simply referred to as “cross-entropy,” “logarithmic loss,” “logistic loss,” or “log loss” for short. Each predicted probability is compared to the actual class output value (0 or 1) and a score is calculated that penalizes the probability based on the distance from the expected value. The penalty is logarithmic, offering a small score for small differences (0.1 or 0.2) and enormous score for a large difference (0.9 or 1.0).

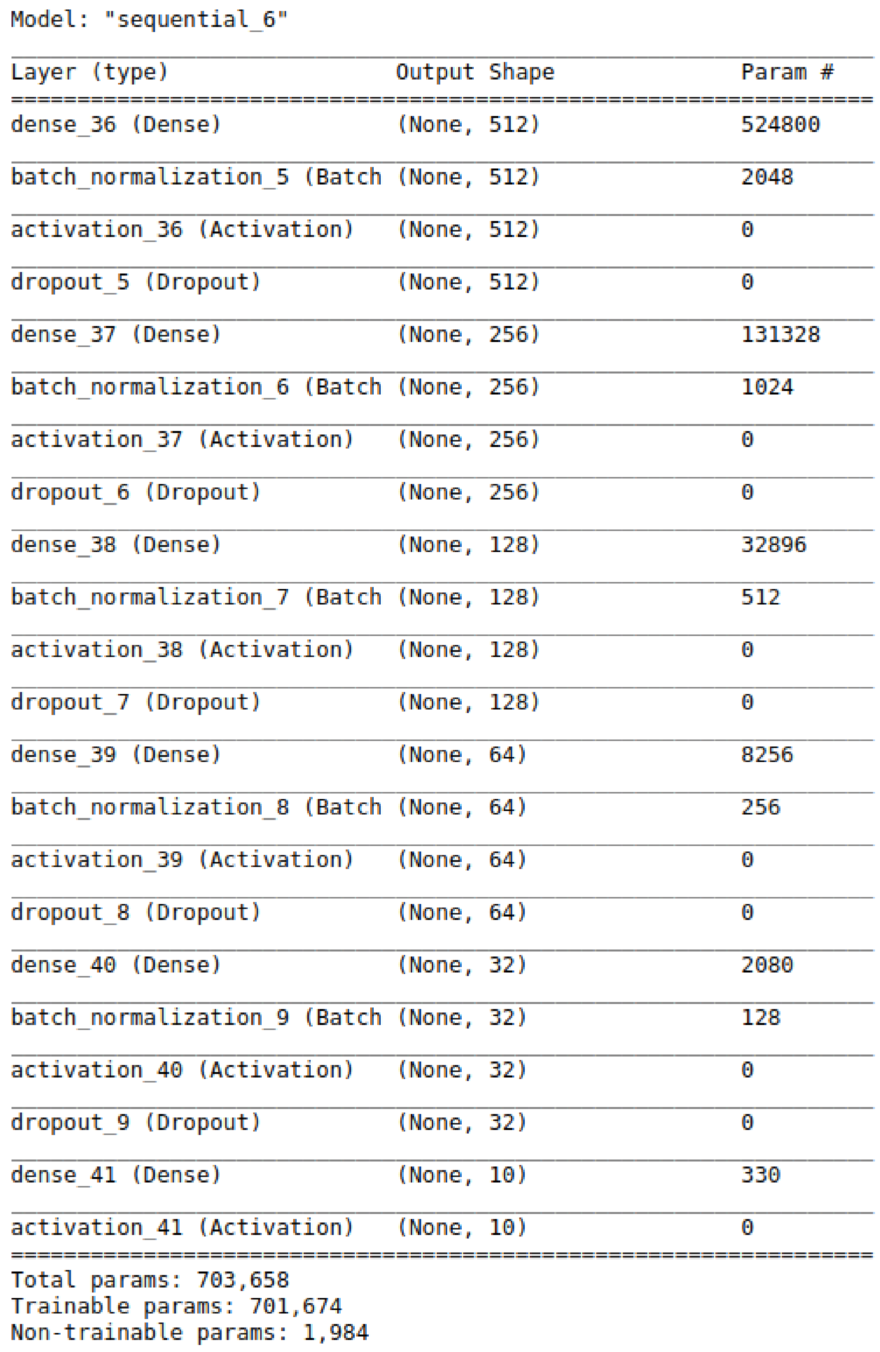
Cross-entropy loss is minimized, where smaller values represent a better model than larger values. A model that predicts perfect probabilities has a cross entropy or log loss of 0.0.After implementing adam optimization the model is working better, and is a good learner

To get better accuracy we tune paramethers, and set learning rate to 0.0001.The model is perfoming great, and we are mostly sattisfied with results. Of coure we could try to achive better results, but it has higher rist to overfit





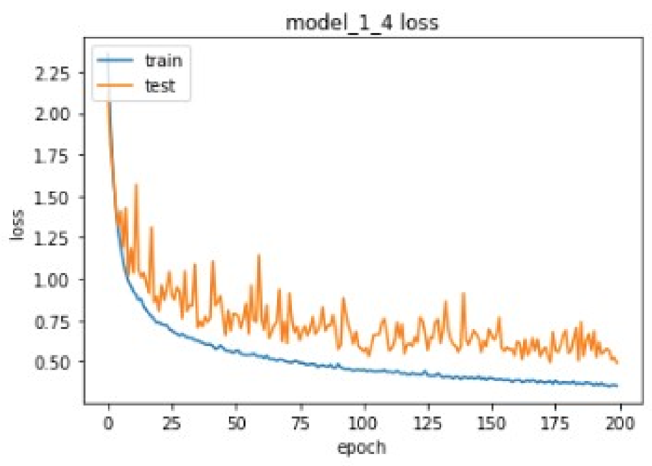
6. Fit and evaluate the model. Print the loss and accuracy for the test data

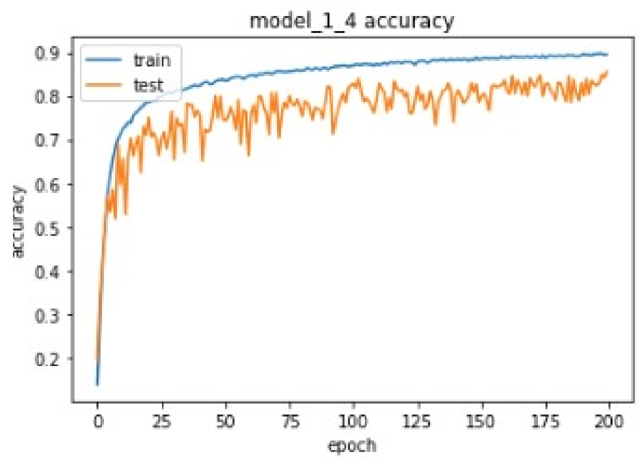


```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_36 (Dense)             (None, 512)               524800
_________________________________________________________________
batch_normalization_5 (Batch (None, 512)               2048
_________________________________________________________________
activation_36 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_37 (Dense)             (None, 256)               131328
_________________________________________________________________
batch_normalization_6 (Batch (None, 256)               1024
_________________________________________________________________
activation_37 (Activation)   (None, 256)               0
_________________________________________________________________
dropout_6 (Dropout)          (None, 256)               0
_________________________________________________________________
dense_38 (Dense)             (None, 128)               32896
_________________________________________________________________
batch_normalization_7 (Batch (None, 128)               512
_________________________________________________________________
activation_38 (Activation)   (None, 128)               0
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_39 (Dense)             (None, 64)                8256
_________________________________________________________________
batch_normalization_8 (Batch (None, 64)                256
_________________________________________________________________
activation_39 (Activation)   (None, 64)                0
_________________________________________________________________
dropout_8 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_40 (Dense)             (None, 32)                2080
_________________________________________________________________
batch_normalization_9 (Batch (None, 32)                128
_________________________________________________________________
activation_40 (Activation)   (None, 32)                0
_________________________________________________________________
dropout_9 (Dropout)          (None, 32)                0
_________________________________________________________________
dense_41 (Dense)             (None, 10)                330
_________________________________________________________________
activation_41 (Activation)   (None, 10)                0
=================================================================
Total params: 703,658
Trainable params: 701,674
Non-trainable params: 1,984
```

The next model is with batch normalization + dropout layer yields (about 30% of neurons were dropped)

Dropout is a technique used to prevent a model from overfitting. After creating our model architecture, we fit the training dataset in the model and evaluate.

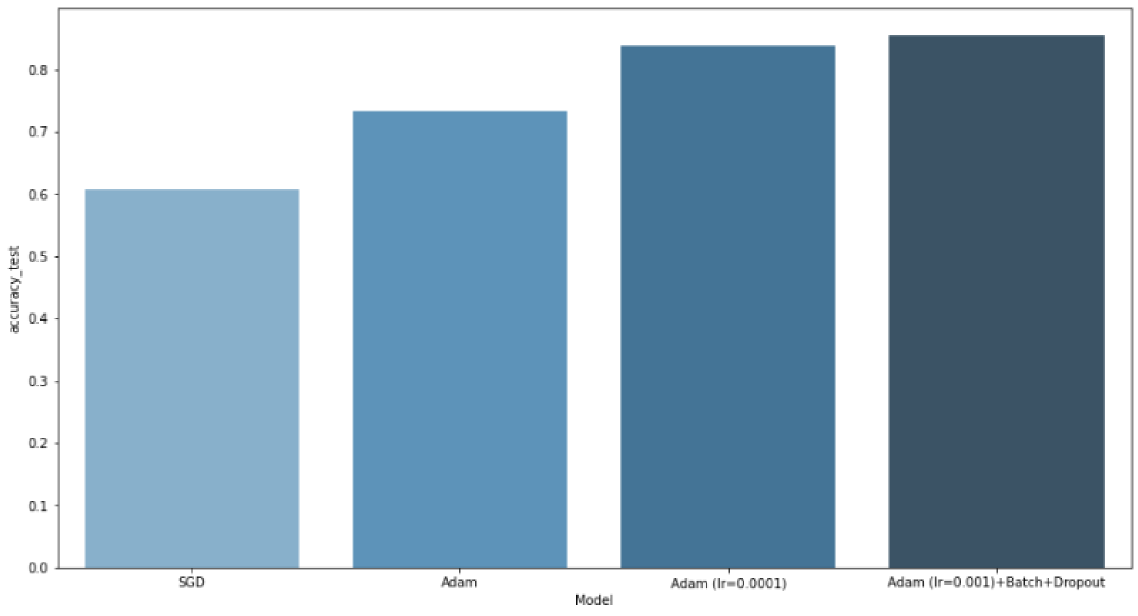




On the training set the model is well trained, although it performs worse in the testing set and has a lot of small peaks

After creating our model architecture, we fit the training dataset in the model and evaluate.

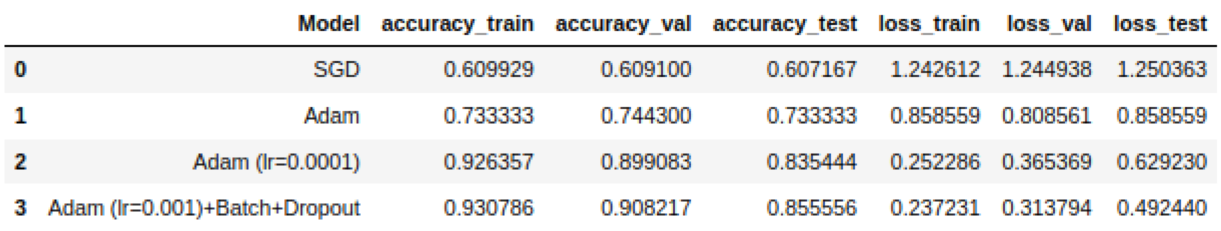
	Model	accuracy_train	accuracy_val	accuracy_test	loss_train	loss_val	loss_test
0	SGD	0.609929	0.609100	0.607167	1.242612	1.244938	1.250363
1	Adam	0.733333	0.744300	0.733333	0.858559	0.808561	0.858559
3	Adam (lr=0.0001)	0.976214	0.934850	0.838333	0.080856	0.330668	0.913561
2	Adam (lr=0.001)+Batch+Dropout	0.930786	0.908217	0.855556	0.237231	0.313794	0.492440



The plot above shows accuracy score on testing data. According to this, the best result is in model with adam optimization(learning rate 0.001), batch and dropout(0.3)

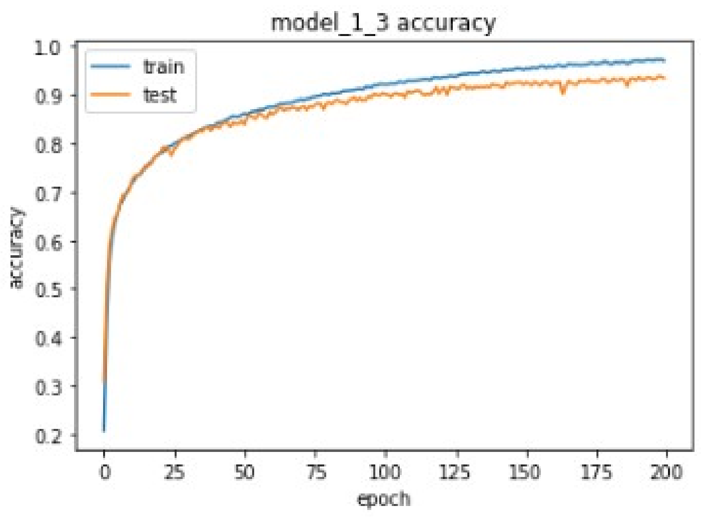
Althouth perfomimg on the test data the model is quite unstable, so we would prefer to chose model with Adam optimization (learning rate = 0.0001)

7. Plot the training loss, validation loss vs number of epochs and training accuracy, validation accuracy vs number of epochs plot and write your observations on the same.

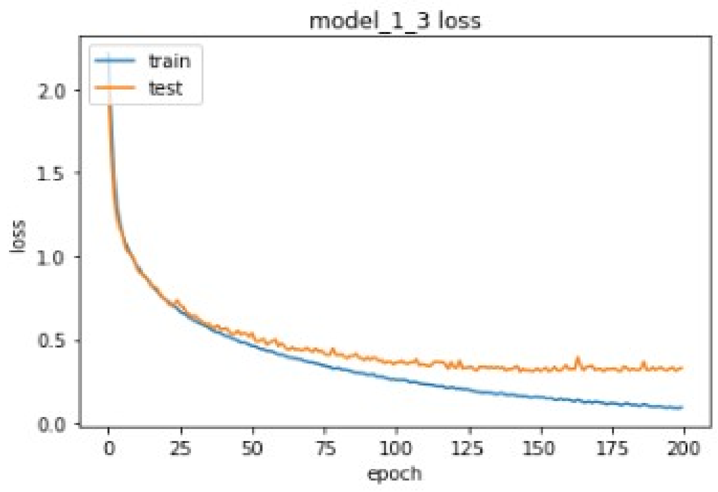


	Model	accuracy_train	accuracy_val	accuracy_test	loss_train	loss_val	loss_test
0	SGD	0.609929	0.609100	0.607167	1.242612	1.244938	1.250363
1	Adam	0.733333	0.744300	0.733333	0.858559	0.808561	0.858559
2	Adam (lr=0.0001)	0.926357	0.899083	0.835444	0.252286	0.365369	0.629230
3	Adam (lr=0.001)+Batch+Dropout	0.930786	0.908217	0.855556	0.237231	0.313794	0.492440

- Also, the optimizer Adam is preferred over the SGD.
- The learning rate of 0.001 is optimal for the learning curve and coupled along with the Batch Normalization and Dropout layer architecture proves to be the best model for classification in this scenario.



The plot above shows the training accuracy and testing accuracy vs number of iterations done, the training accuracy has a smooth gradual increasing curve up to 97%, the testing dataset slightly diverges at the end and has small peaks



And similar case can be observed in the next plot, training loss and test loss vs number of epochs, the loss for training dataset is gradually decreasing and settled at 0.08 whereas the test losses doesn't follow a smooth cure rather thelosses are unexpectedly decreasing and settled at 0.91 which is acceptable. possible solutions for reducing the fluctuations in the test dataset:

- obtain more data-points (or artificially expand the set of existing ones)
- play with hyper-parameters (increase/decrease capacity or regularization term for instance)
- regularization: try dropout, early-stopping, so on

Since we can see that the training dataset is a bit overfitted we can avoid this by choosing best fit parameters and more iteration(epochs).

In order to increase the accuracy of the test dataset and decrease the losses we need to employee hyperparameter tuning method, in which the learning rate and lambda values will not be considered as constants rather we explore the values and find the best fitted values for using in cross entropy loss and Adam optimizers.

To avoid the fluctuations in the test curves, we need to modify our model and change the number of dense layers/hidden layers through the model and also by hyper parameter tuning.