Multi-label Classification of Digits and Alphabet

Abstract

In this project, we develop models to classify the image data into 36 categories (10 digits and 26 letters). We preprocessed the data by denoising and normalizing them, and applied CNN models, and VGG-19 model to make a comparison of them. We found the best model to be VGG-19, which gives us 0.9493 accuracy and 1.420 loss.

1. Introduction

CNNs has been widely used in image classification tasks. Compared to other classification methods such as AVM, CNN performs well in extracting the features of an image, hence giving you more optimal results when classifying the image dataset. In this project, we use a Combo MNIST dataset of 60000 images of size 56x56 where each of them contains one letter from the English alphabet and one digit number as the object that needs to be classified and use 2 types of CNN architecture and VGG as the base of our implementations, namely CNN3, CNN4 and VGG-19.

2. Dataset

2.1 Data Preprocessing

We applied normalization in order to change the values of numeric columns in the dataset to use a common scale, without distorting differences in the ranges of values or losing information.
We also used 10% of the testing dataset as validation dataset, the obtained accuracies and losses are based on validation dataset.

2.2 Usage of Unlabeled Data

Autoencoding is applied to unlabeled data for data denoising and dimensionality reduction.

3. Model Selection

Firstly, we chose Convolutional Neural Network (CNN) as our main setting, and tried various experiments on it and selected best-performed model architecture, hyperparameters, etc. And we also use VGG-19 to be our side setting to compare the performance with our CNN models.

3.1 CNN

3.1.1 Model Architecture Selection

We have 2 differentCNN networks to compare: 3L CNN, 4L CNN:
-- As figure 1 shows, 3L CNN has 3 sequentials of 6 layers and 1 fully connected layer in total, each sequential is a combination of Convolution, ReLU, batch normalization, max pooling and skip-connection. And each Convolutional layer follows an activation function, ReLU, or a batch normalization to overcome the gradient vanishing and overfitting issue.
-- As figure 2 shows, 4L CNN has 4 sequentials of 8 layers and 1 fully connected layer in total, each sequential has similar structure as the previous model but with different parameters. In order to insert more Convolutional layers in it, we reduce some max-pooling and put more Conv2D filters because Max-pool could reduce the dimension of data dramatically.
After running two models separately and getting the accuracy and loss on validation dataset, CNN with 6 layers shows an accuracy of 0.76 and loss of 3.14; CNN with 8 layers has an accuracy of 0.87 and loss of 1.94. As figure 3.1&3.2 shows, clearly CNN with 8 layers ran much better, thus we chose 4L CNN to further the comparison of hyperparameters.

3.1.2 Hyperparameter Selection for CNN

-Drop-out Rate

We tried the following 3 drop-out rates: 0.1, 0.25, 0.5. The drop-out rates inside of the model are all the same. By randomly dropping out each unit with a probability, we regularized the model to be less overfitting. As figure 4.1 & 4.2 presents, the dropout rate at level of 0.1 performs much better, which has an accuracy of 0.878.

-Learning Rate

We tried the following 3 learning rates: 1e-3, 1e-2, 1e-1. The learning rate is used inside the Adam optimization algorithm to update the model's parameters based on the computed gradients. As figure

5.1 & 5.2 presents, the learning rate at level of 1e-3 performs better than the rest, which has an accuracy of 0.878.

-Batch Size

We tried the following 4 batch sizes : 8, 16, 32, 64. As figure 6.1 & 6.2 presents, the batch size of 32 performs better than the rest, having the highest accuracy of 0.701 and lowest loss using the same CNN-4 layer model.

-Epoch

We tried the epoch size from 1 to 200. As figure 7 presents, loss of model decreases and begins to converge since epoch 200. Therefore, we chose epoch size 200 to train our best model.

3.2 VGG

We also implemented the model VGG-19 which has 16 convolutional layers and 3 fully connected layers. The basic components of a VGG network are the following sequence: 1. a convolutional layer with padding to maintain resolution; 2. a nonlinear activation function; 3. a convergence layer. The VGG neural network continuously connects several VGG blocks. We chose the SGD to be our optimizer and adjust the learning rate and the momentum to make the model faster and more accurate.

4. Creativity

4.1. Autoencoding is used on unlabeled data for denoising and dimensionality reduction. We constructed an Encoder network that maps images to a low dimensional space (56*56*1 → 14*14*8), and an Decoder network which maps the low dimensional space back to images (56*56*1 → 14*14*8). Then we train the Autoencoder by using Decoder(Encoder(img)) = img. After it's trained, we directly use encoded images to train the classifier. By doing these, we find that it can avoid the curse of dimensionality and make the classifying much faster.

4.2. We also add and adjust layers by summarizing the model and comparing them. For example, we find it performs much better when a convolutional layer followed by a batch normalization.

5. Discussion and Conclusion

5.1 Data preprocessing

The data preprocessing is extremely important. The raw data has many noises and numbers are very large, which could affect our classification negatively. The loss of the model reduced from over 100 to within 1 by simply normalizing and reshaping the data. And the denoiser can make the data clearer and resistant to the surrounding environment, it helps increase our accuracy by 4 percent.

5.2 Model selection

For CNN, after selecting the best-performed hyperparameters, we get the optimal model. The best model has 8 layers in it, with dropout = 0.1, learning rate = 1e-3, batch size = 32 and epoch of 200. The accuracy for validation dataset is 0.917 and loss is 1.4162.

For VGG-19, it is chosen because it has been adapted and used for the original MNIST dataset before. To prevent overfitting and improve the model's ability to generalize, we added batch normalization betion each convolution block and used a dropout layer with a rate of 0.5. We also used the GaussionNoise layer to improve model performance. The best model is with learning rate = 2e-3, momentum = 0.9, epoch of 20 using SGD optimization algorithm. The accuracy for validation dataset is 0.9493 and loss is 1.420.

5.3 The use of unlabeled data

The most obvious effect of training unlabeled data is that it makes our prediction much faster. This is because of the reduction of dimensionality, we compress the images to be smaller without losing the information on it.

6. Statement of Contributions

Hannah Zhang: Design and implementation of VGG-19, adjusting the hyperparameters in models

Jiangshan Yu: Implement CNN model and adjusting layers in the network, implementing AutoEncoder, visualization

Yiyuan Shang: Design and implementation of CNN and AutoEncoder, write-up

References

[1]Francois Chollet, Building Autoencoders in Keras, 14 May 2016,
https://blog.keras.io/building-autoencoders-in-keras.html
[2]Bentrevett, 4 - VGG, 26 Aug 2020
https://github.com/bentrevett/pytorch-image-classification/blob/master/4_vgg.ipynb
[3]Mffattesko, CNN on Modified MNIST.ipynb, Mar 15, 2019
https://github.com/mattesko/COMP551-Projects/blob/master/miniproject_3/CNN%20on%20Modified%20MNIST.ipynb

Appendix

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 28, 55, 55]	140
ReLU-2	[-1, 28, 55, 55]	0
Conv2d-3	[-1, 28, 54, 54]	3,164
BatchNorm2d-4	[-1, 28, 54, 54]	56
ReLU-5	[-1, 28, 54, 54]	0
MaxPool2d-6	[-1, 28, 27, 27]	0
Dropout-7	[-1, 28, 27, 27]	0
Conv2d-8	[-1, 56, 26, 26]	6,328
ReLU-9	[-1, 56, 26, 26]	0
Conv2d-10	[-1, 56, 24, 24]	28,280
BatchNorm2d-11	[-1, 56, 24, 24]	112
ReLU-12	[-1, 56, 24, 24]	0
MaxPool2d-13	[-1, 56, 12, 12]	0
Dropout-14	[-1, 56, 12, 12]	0
Conv2d-15	[-1, 112, 10, 10]	56,560
ReLU-16	[-1, 112, 10, 10]	0
Conv2d-17	[-1, 112, 8, 8]	113,008
BatchNorm2d-18	[-1, 112, 8, 8]	224
ReLU-19	[-1, 112, 8, 8]	0
MaxPool2d-20	[-1, 112, 3, 3]	0
Dropout-21	[-1, 112, 3, 3]	0
Linear-22	[-1, 36]	36,324

Figure 1 Architecture of CNN 3 layer

Layer (type)	Output Shape	Param #
Conv2d-1	[-1, 28, 55, 55]	140
ReLU-2	[-1, 28, 55, 55]	0
Conv2d-3	[-1, 28, 54, 54]	3,164
BatchNorm2d-4	[-1, 28, 54, 54]	56
ReLU-5	[-1, 28, 54, 54]	0
MaxPool2d-6	[-1, 28, 27, 27]	0
Dropout-7	[-1, 28, 27, 27]	0
Conv2d-8	[-1, 56, 26, 26]	6,328
ReLU-9	[-1, 56, 26, 26]	0
Conv2d-10	[-1, 56, 24, 24]	28,280
BatchNorm2d-11	[-1, 56, 24, 24]	112
ReLU-12	[-1, 56, 24, 24]	0
MaxPool2d-13	[-1, 56, 12, 12]	0
Dropout-14	[-1, 56, 12, 12]	0
Conv2d-15	[-1, 112, 10, 10]	56,560
ReLU-16	[-1, 112, 10, 10]	0
Conv2d-17	[-1, 112, 8, 8]	113,008
BatchNorm2d-18	[-1, 112, 8, 8]	224
ReLU-19	[-1, 112, 8, 8]	0
MaxPool2d-20	[-1, 112, 4, 4]	0
Dropout-21	[-1, 112, 4, 4]	0
Conv2d-22	[-1, 224, 3, 3]	100,576
ReLU-23	[-1, 224, 3, 3]	0
Conv2d-24	[-1, 224, 2, 2]	200,928
BatchNorm2d-25	[-1, 224, 2, 2]	448
ReLU-26	[-1, 224, 2, 2]	0
MaxPool2d-27	[-1, 224, 1, 1]	0
Dropout-28	[-1, 224, 1, 1]	0
Linear-29	[-1, 36]	8,100

Figure 2 Architecture of CNN 4 layer

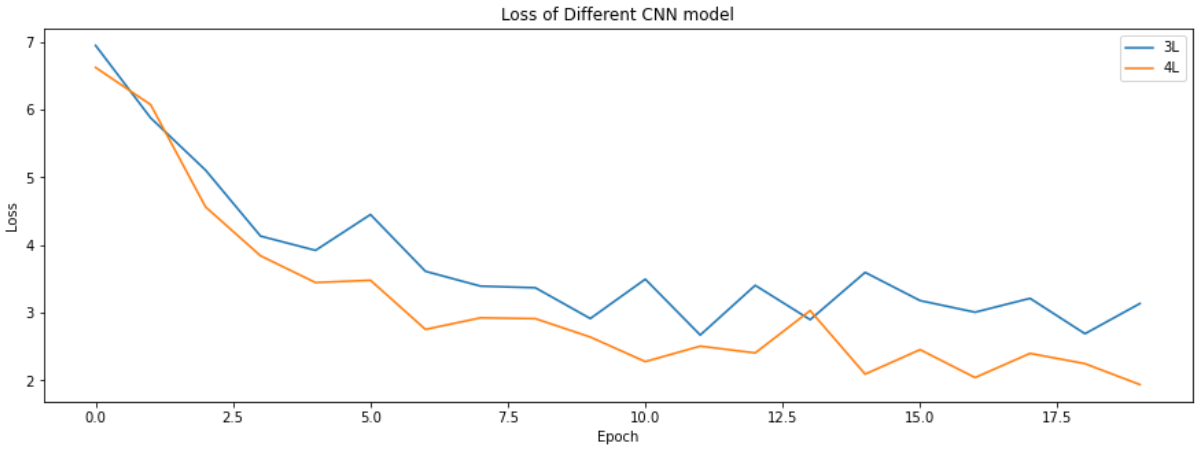

Figure 3.1 Model Architecture Comparison : Loss of different CNN models

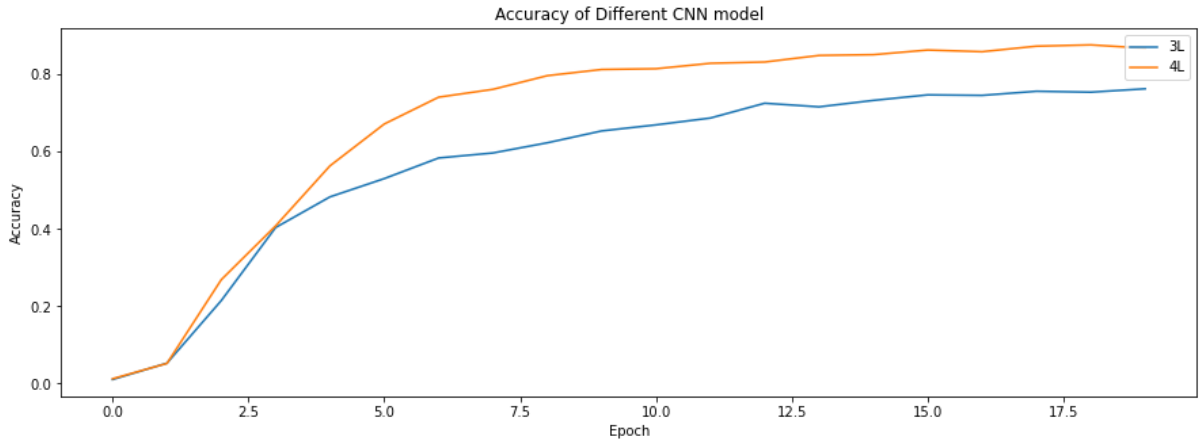

Figure 3.2. Model Architecture Comparison : Accuracy of different CNN models

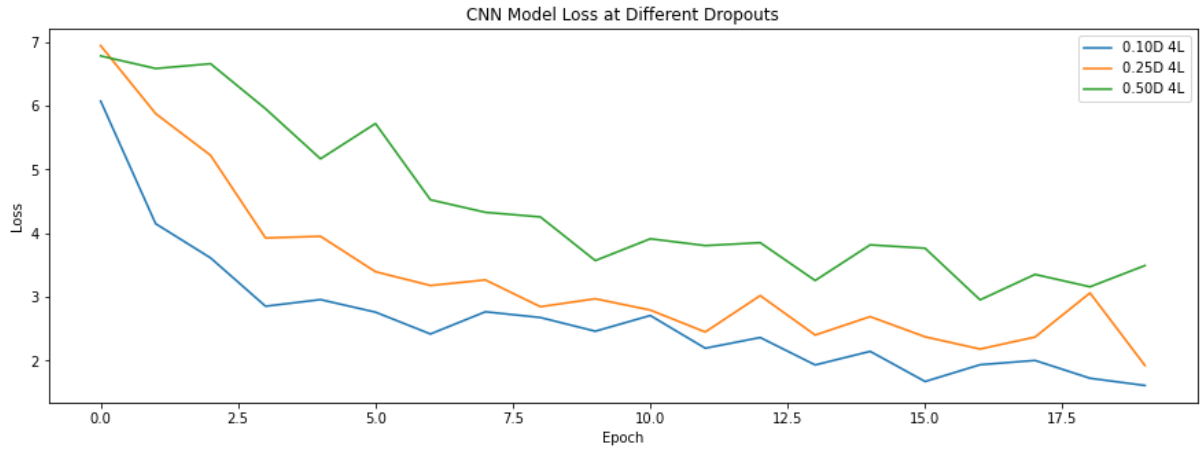

Figure 4.1. Dropout selection : Loss of different dropouts

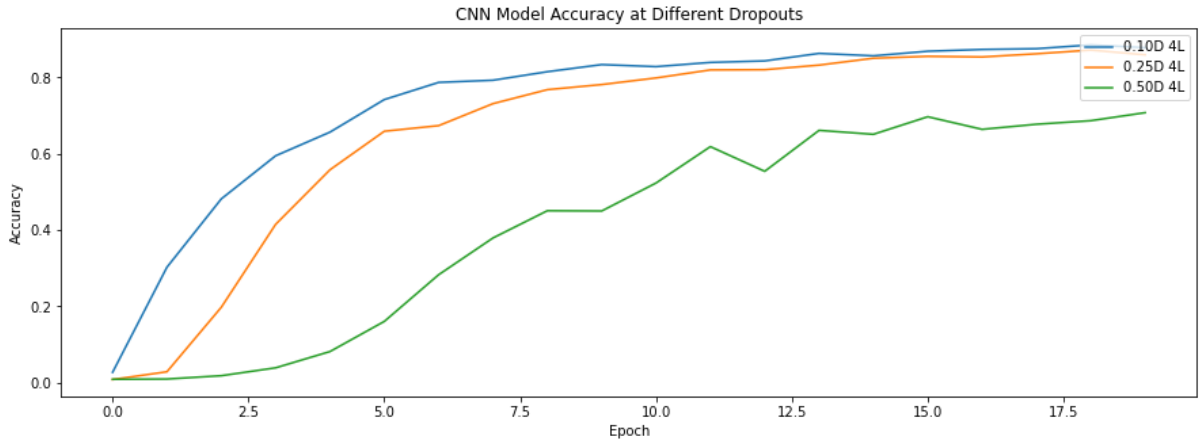

Figure 4.2. Dropout selection : Accuracy of different dropouts

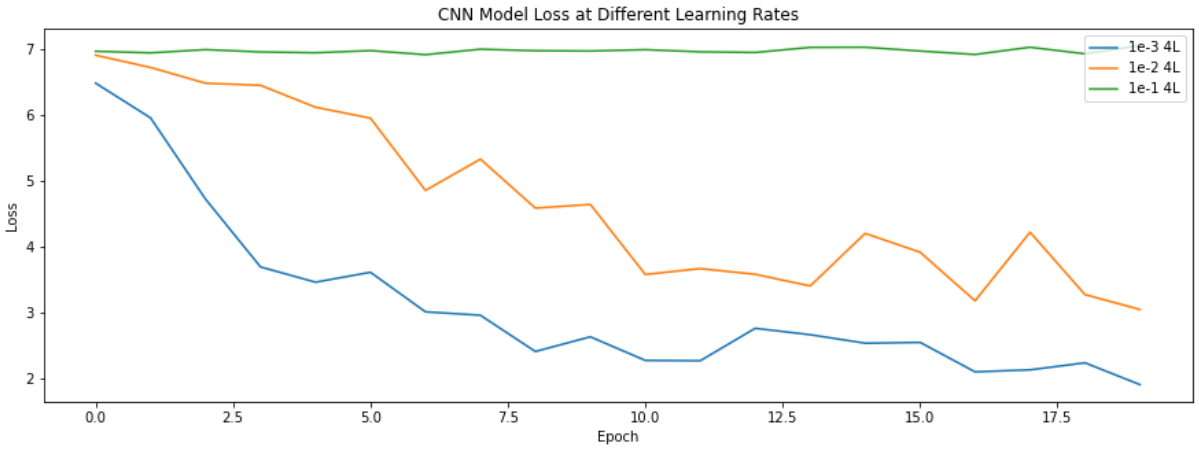

Figure 5.1. Learning rate selection : Loss of different learning rates

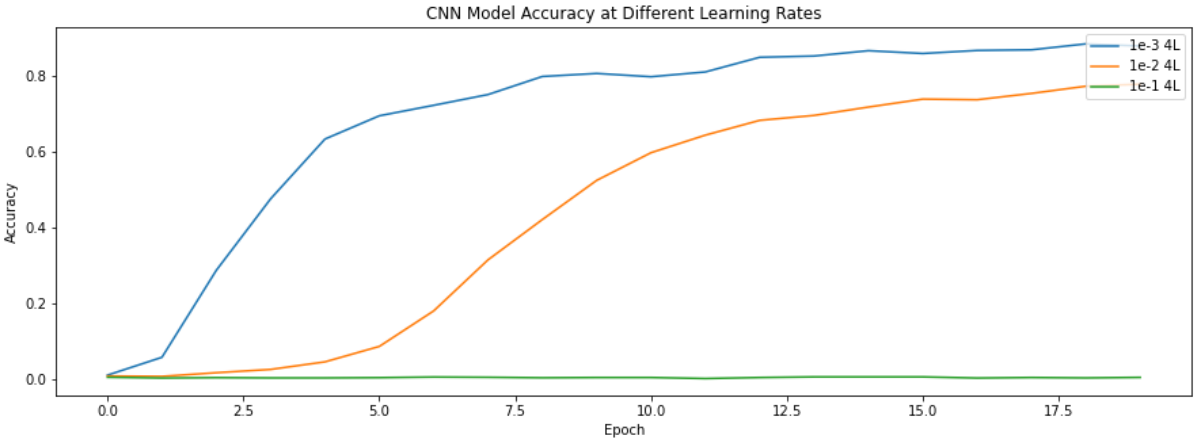

Figure 5.2. Learning rate selection : Accuracy of different learning rates

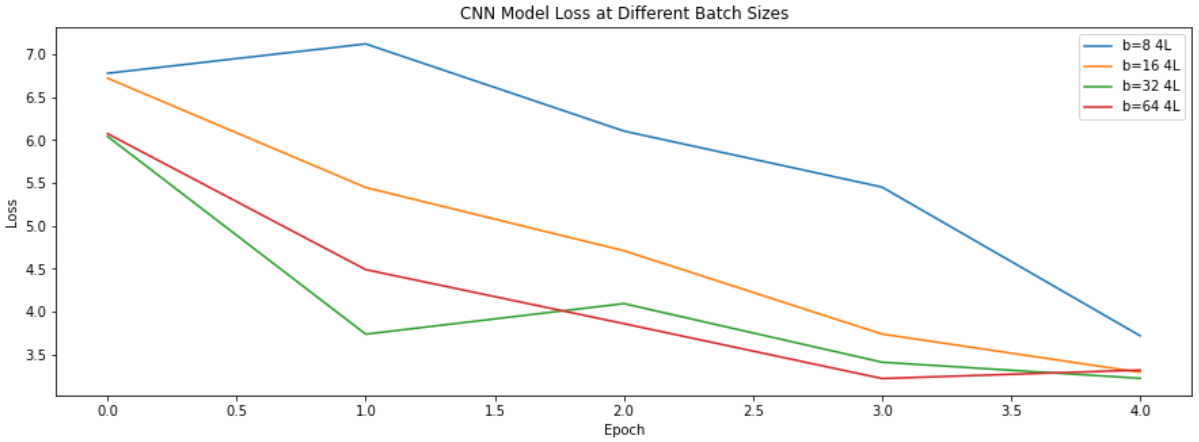

Figure 6.1. Batch size selection : Loss of different batch sizes

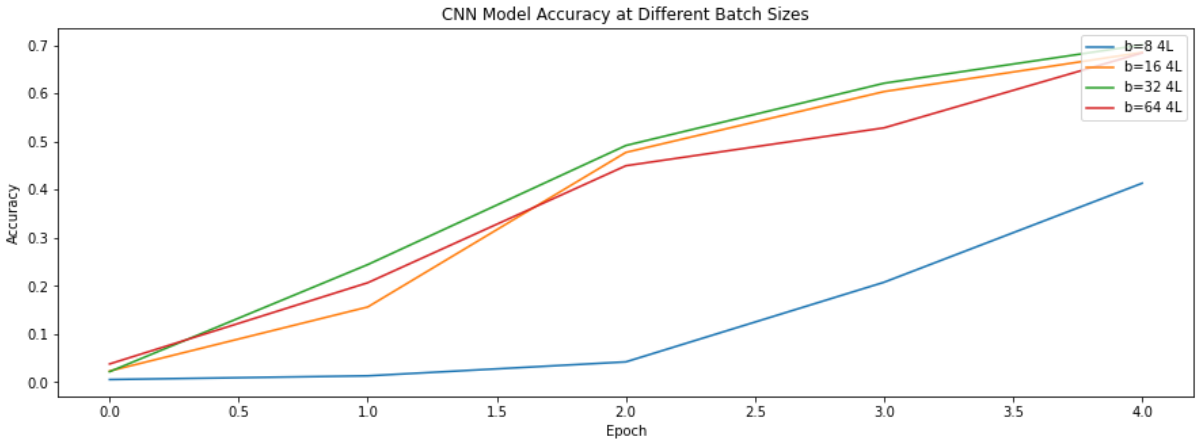

Figure 6.1. Batch size selection : Accuracy of different batch sizes

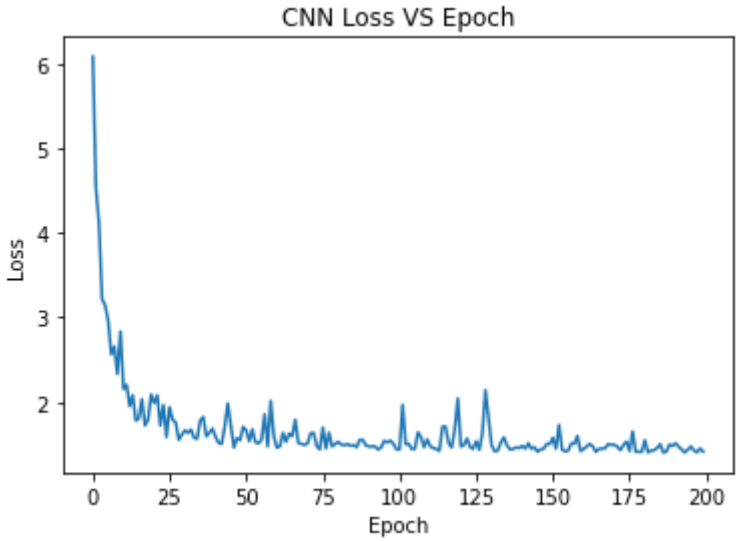

Figure 7. Epoch selection : Loss of different epochs