Improve Convolutional Neural Network model in Images Classification

Table of Contents:

Title and author:

1. **Title**:

   Improve Convolutional Neural Network model in Images Classification

2. **Author**:

   - Name: Thai Quoc Hoang
   - Date of birth: January 09, 2000
   - Email: quocthai9120@gmail.com (mailto:quocthai9120@gmail.com) / qthai912@uw.edu (mailto:qthai912@uw.edu)
   - GitHub: https://github.com/quocthai9120 (https://github.com/quocthai9120)

Introduction:

This project is done based on the result from the final research project in the CSE163 course from the University of Washington about Using different models in Image Classification. The link below gives more information about that research project: https://github.com/quocthai9120/Images-Classification (https://github.com/quocthai9120/Images-Classification).

The previous result ends up with 70.09% correct accuracy for 10000 testing instances using the CIFAR_10 dataset. This is not a bad start for the image classification task. However, we need to improve the model significantly if we want to use in real-life classification tasks.

This project focuses on experimenting with different methods to improve the Convolutional Neural Network model that used to image classification using the CIFAR_10 dataset, including data preprocessing, pre-trained model modification, and training modification.

Dataset:

CIFAR_10 dataset:

Source and Information:

The CIFAR-10 dataset contains 60000 32x32 colour images in 10 classes (6000 images/class) that are randomly shuffled and are divided into 2 parts:

- 50000 images for training (5000 images for each class).
- 10000 images for testing (1000 images for each class).

The classes are completely mutually exclusive and there is no overlap between classes.

10 classes of CIFAR-10 dataset are:

0. airplane
1. automobile
2. bird
3. cat
4. deer
5. dog
6. frog
7. horse
8. ship
9. truck

The dataset were collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton and can be accessed through the link provided here : https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html) or directly download here: https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz (https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz).

Dataset layout:

The CIFAR-10 datset used in this project is the Python Version.

The download compressed file contains 7 files: data_batch_1, data_batch_2, data_batch_4, data_batch_5,test_batch, and batches.meta, which are Python "pickled" objects.

The source of the dataset (https://www.cs.toronto.edu/~kriz/cifar.html (https://www.cs.toronto.edu/~kriz/cifar.html)) provides a function called 'unpickle(file)' to open such files and returns the data as dictionaries.

Methodology:

To improve the CNN model in Images Classification task, we experiment with multiple methods for the model.

Data Augmentation:

A fundamental principle of Deep Learning is that the quantity and quality of data plays an important role in determining how well the model behaves. However, collecting new data requires a trade of money and time. With that reason, data augmentation is an appropriated solution to generate more data using the existing data.

For this project, as CIFAR_10 is a complex dataset in which images have different shapes, colors, backgrounds,... we will generate new data by augmenting existing data while training the model by each batch rather than generate images before training to utilize more augmenting data and also avoid overfitting during the training process.

Several examples of augmented images generated while training the model:

Data Normalization:

The deeper network requires longer training time. To decrease the training time, we need to make convergence faster while training the model. Therefore, data normalization plays an important role in contributing to this process.

For this project, we normalization the data using the formula:

$$X_n = (X - \mu)/\sigma, \text{ where:}$$

- X_n: The normalized image
- X: The original image
- μ: The mean of values of the image
- σ: The standard deviation of values of the image

The algorithm is applied in two steps:

- Subtracting the mean from each pixel
- Dividing each pixel by the standard deviation

Batch Normalization:

During the training process, the distribution of activations is continuously changing after each step of backpropagation, which causes the training process slower. Batch Normalization allows us to normalize the distribution of activations by adjusting and scaling the activations so that the input for every layer will have a similar distribution after each training step.

For this project, we set up the batch normalization after using ReLU activation function for each Convolutional layer and for Dense layer.

Modify the Neural Network model:

The previous model resulted in 70.09% accuracy after training 25 epochs. If we continue training the model, we will have an overfitting problem. To overcome this overfitting and improve the accuracy, we need to modify the whole model by tuning the hyperparameters and adding layers to the model.

According to the paper of Reconciling modern machine learning and the bias-variance trade-off, as described in [5], the deep learning model could have the "Double descent" risk curve. That is if we make the model complex enough, the loss will start decreasing again.

We need to add more convolutional layers to learn more abstract features of the data. To do so, we also need to decrease the complexity of the dense layer.

In addition, as we are using Batch Normalization during the training process, it is necessary to decrease the Dropout rate to avoid confliction with batch normalization.

We use the loss function categorical crossentropy:

$$\text{Cross Entroy Loss} = -\sum_{i}^{C} t_i log(s_i), \text{ where:}$$

Summary of the improved model:

Layer (type)	Output Shape	Param #
conv2d_13 (Conv2D)	(None, 28, 28, 32)	2432
activation_21 (Activation)	(None, 28, 28, 32)	0
max_pooling2d_9 (MaxPooling2	(None, 14, 14, 32)	0
conv2d_14 (Conv2D)	(None, 10, 10, 64)	51264
activation_22 (Activation)	(None, 10, 10, 64)	0
conv2d_15 (Conv2D)	(None, 8, 8, 128)	73856
activation_23 (Activation)	(None, 8, 8, 128)	0
max_pooling2d_10 (MaxPooling	(None, 4, 4, 128)	0
flatten_5 (Flatten)	(None, 2048)	0
dense_9 (Dense)	(None, 256)	524544
activation_24 (Activation)	(None, 256)	0
dropout_5 (Dropout)	(None, 256)	0
dense_10 (Dense)	(None, 10)	2570
activation_25 (Activation)	(None, 10)	0

Total params: 654,666
Trainable params: 654,666
Non-trainable params: 0

Previous Model - 70.09% Testing Accuracy

Layer (type)	Output Shape	Param #
conv2d_5 (Conv2D)	(None, 30, 30, 32)	896
activation_7 (Activation)	(None, 30, 30, 32)	0
batch_normalization_6 (Batch	(None, 30, 30, 32)	128
conv2d_6 (Conv2D)	(None, 28, 28, 64)	18496
activation_8 (Activation)	(None, 28, 28, 64)	0
batch_normalization_7 (Batch	(None, 28, 28, 64)	256
max_pooling2d_3 (MaxPooling2	(None, 14, 14, 64)	0
conv2d_7 (Conv2D)	(None, 12, 12, 128)	73856
activation_9 (Activation)	(None, 12, 12, 128)	0
batch_normalization_8 (Batch	(None, 12, 12, 128)	512
conv2d_8 (Conv2D)	(None, 10, 10, 256)	295168
activation_10 (Activation)	(None, 10, 10, 256)	0
batch_normalization_9 (Batch	(None, 10, 10, 256)	1024
max_pooling2d_4 (MaxPooling2	(None, 5, 5, 256)	0
flatten_2 (Flatten)	(None, 6400)	0
dense_3 (Dense)	(None, 64)	409664
activation_11 (Activation)	(None, 64)	0
batch_normalization_10 (Batc	(None, 64)	256
dropout_2 (Dropout)	(None, 64)	0
dense_4 (Dense)	(None, 10)	650
activation_12 (Activation)	(None, 10)	0

Total params: 800,906
Trainable params: 799,818
Non-trainable params: 1,088

Improved Model - 87.06% Testing Accuracy

Reduce Learning Rate:

In addition, we also need to prevent the case that the high learning rate causes the model hard to get converge or even get diverge.

For our current model, we use a small modification by reducing the learning rate using the formula:

$$L_n = L * \alpha, \text{ where:}$$

- L_n: new learning rate
- L: current learning rate
- α: multiplication factor

We set up the patience for the model to decrease the learning rate if the model does not decrease loss for 3 training epochs.

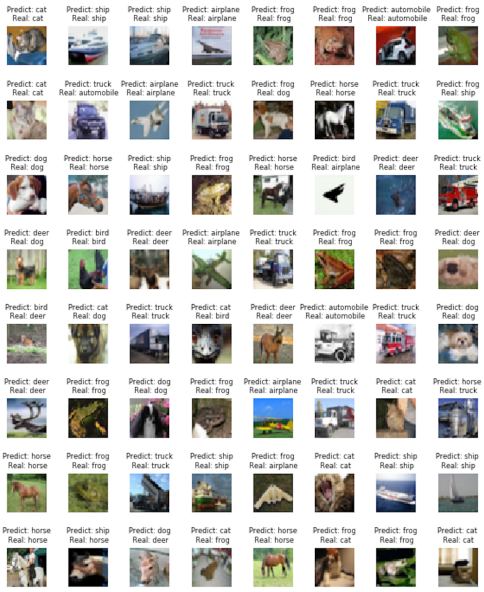
Early Stopping:

Because we are experimenting with different methods, we need to explore a huge range of training epochs. Early stopping allows us to stop when the model converges or starts to get diverge so that we can explore training the model with more epochs.

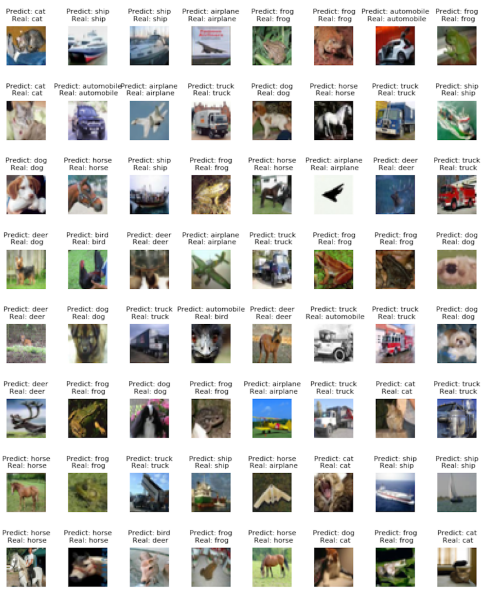
For our current model, we set up the patience for the model to stop training if the model does not decrease loss for 10 training epochs.

Results:

- The project has a 87.06% correct accuracy using 10000 testing instances for classification, which is increased 16.97% from 70.09% as the previous project. Using several images from testing set to compare two models:

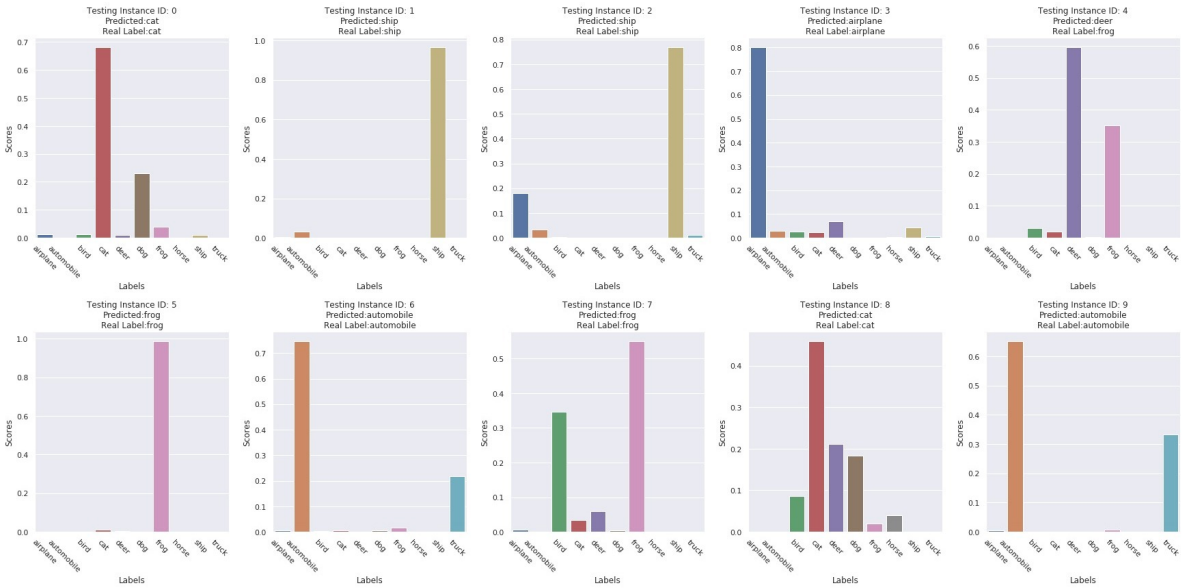


Previous Model - 70.09% Testing Accuracy

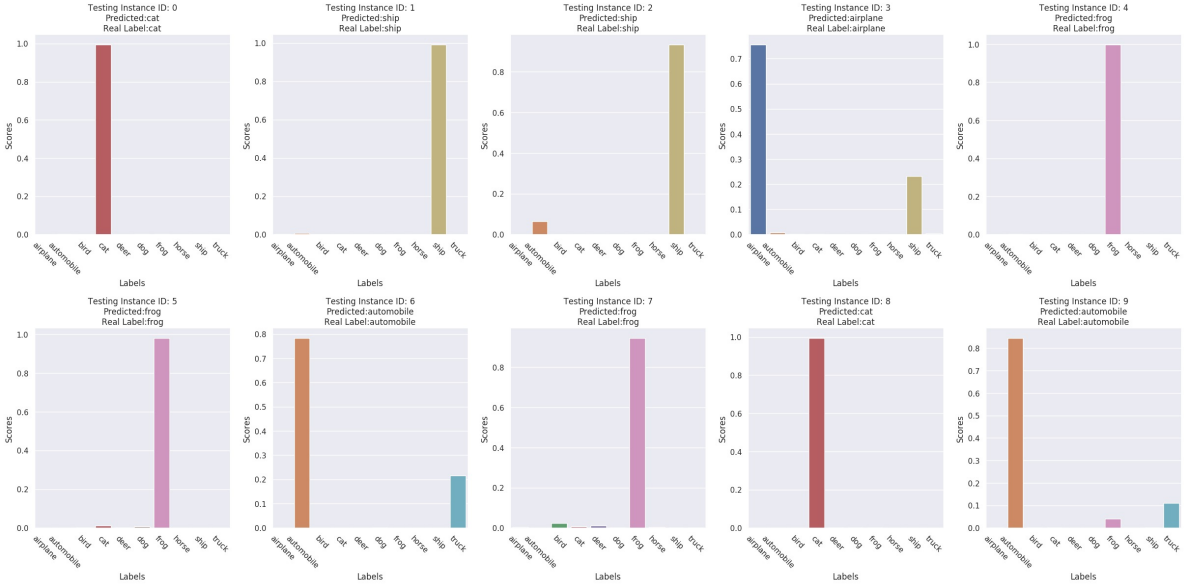


Improved Model - 87.06% Testing Accuracy

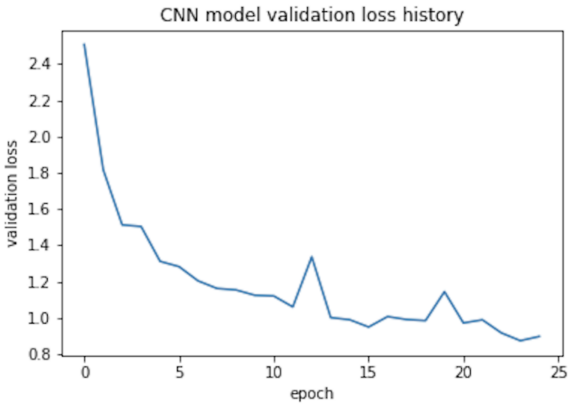
- Using the first ten examples in test set and compute the probabilities for each label, we got the results for the previous model and the improved model:
  - Previous model:



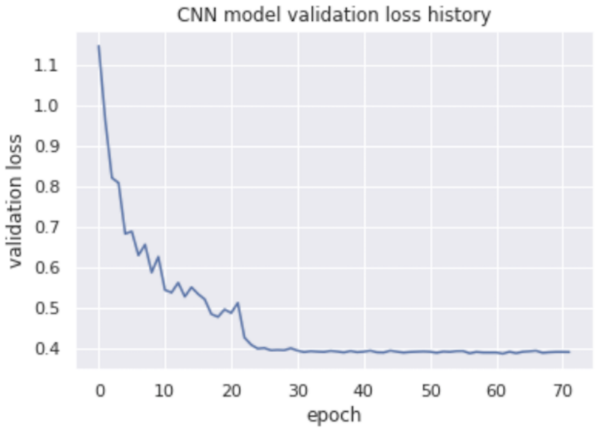
  - Improved model:



    - As the 10 examples are shuffled randomly, the results can represent the trend of the CIFAR_10 dataset. From the results we have produced, we recognize the fact that for most images in this dataset, the improved CNN model gives predictions with higher confidence because along with the high value of resulted label's probabilities, the improved model reduces the probabilities of wrong predictions (the values of other labels) that we can refer to the loss of the prediction.
- The improved model also decreases loss smoother compared to the previous model and does not have the overfitting problem at the last few epochs like the previous model:



Previous Model - 70.09% Testing Accuracy



Improved Model - 87.06% Testing Accuracy

References:

[1]. CS231n Convolutional Neural Networks for Visual Recognition - Data Preprocessing: http://cs231n.github.io/neural-networks-2/#datapre (http://cs231n.github.io/neural-networks-2/#datapre)

[2]. Jason Brownlee. A Gentle Introduction to Early Stopping to Avoid Overtraining Deep Learning Neural Network Models: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/ (https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/)

[3]. Sergey Ioffe, Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift: https://arxiv.org/pdf/1502.03167v3.pdf (https://arxiv.org/pdf/1502.03167v3.pdf)

[4]. Francois Chollet. Building powerful image classification models using very little data: https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html (https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html)

[5]. Mikhail Belkin, Daniel Hsu, Siyuan Ma , and Soumik Mandal. Reconciling modern machine learning and the bias-variance trade-off: https://arxiv.org/pdf/1812.11118.pdf (https://arxiv.org/pdf/1812.11118.pdf)