Improvement in Convolutional Neural Network for CIFAR-10 Dataset Image Classification

Suyesh Pandit
M.Tech Scholar
LNCT, Bhopal, India

Sushil Kumar
Professor & Head
LNCT, Bhopal, India

ABSTRACT
Image classification requires the generation of features capable of detecting image patterns informative of group identity. The objective of this study was to classify images from the public CIFAR10 image dataset by leveraging combinations of disparate image feature sources from deep learning approaches. The majority of regular convolutional neural networks (CNN) are based on the same structure: modification of convolution and the process of max-pooling layers connected with a number of entirely linked layers. In this paper, the prime objective is to improve the effectiveness of simple convolutional neural network models. The Artificial Neural Network (ANN) algorithm is applied on a Canadian Institute For Advanced Research dataset (CIFAR-10) using two different CNN structures. The result of the improved model achieves 88% classification accuracy rate by running for 10 hours. The deep learning models are implemented with the use of Keras library available for Python programming language.

Keywords
Artificial neural networks; cifar-10; classification; image; convolutional neural networks; keras; python; jupyter note book; machine learning

1. INTRODUCTION
The ability to classify things correctly requires many hours of training. People get things wrong many times, until eventually, they get it right. The same structure applies to machine learning. By using a high-quality set of data, deep learning can classify objects comparatively well or even better than humans can. With achieving utterly accurate image classifier, some of the monotonous jobs could be replaced by machines, so that humanity could focus on the most enjoyable activities.

Achieving high classification rate on a set of tiny images tends to be difficult, as some of the features that identify specific class are barely visible even to human eyes. The area of computing vision is under constant development in order to be the most effective in investigating and successfully classifying every kind of object. This type of analysis could advance, for example, the usefulness of autonomous cars, which tend to be ineffective in particular situations of object recognition, leading to significant damages. Most of the traditional neural network algorithms do not achieve as satisfying results to be acceptable for most available jobs. The indicated fact disqualifies machines from replacing the monotonous human activities.

This project implements the structure of CNNs different from traditional, where it performs classification on 10 classes of multiple, evenly distributed images available in the CIFAR- 10 dataset. The improved model replaces the max-pooling and dense function with two-dimensional convolution layers, with the achievement of higher classification rate, basing its structure on the model .

2. LITERATURE REVIEW
According to [1], —The traditional convolutional neural network usually initializes the weights of all network layers at one time before network training, and then updates the weights of the network by back-propagation algorithm to improve the accuracy of the network during network training. However, with the increase of network depth, the computational cost of this method will increase dramatically and the test accuracy will be affected. In order to solve this problem, a method of gradually reinitializing the weights of each layer is proposed, that is, after a certain training period, the weight of the previous layer is determined and remain unchanged, then initialize the weights of all subsequent layers, repeat this step until the weights of all layers are determined. In order to verify the performance of the method, a series of experiments were carried out on the CIFAR10 dataset. The results show that the accuracy of the network is improved by 9% and the training time is reduced by 29%. It shows that the method can improve the accuracy of the network and reduce the training time.

In [2], Training the deep learning models involves learning of the parameters to meet the objective function. Typically the objective is to minimize the loss incurred during the learning process. In a supervised mode of learning, a model is given the data samples and their respective outcomes. When a model generates an output, it compares it with the desired output and then takes the difference of generated and desired outputs and then attempts to bring the generated output close to the desired output. This is achieved through optimization algorithms. An optimization algorithm goes through several cycles until convergence to improve the accuracy of the model. There are several types of optimization methods developed to address the challenges associated with the learning process. Six of these have been taken up to be examined in this study to gain insights about their intricacies. The methods investigated are stochastic gradient descent, nesterov momentum, rmsprop, adam, adagrad, adadelta. Four datasets have been selected to perform the experiments which are mnist, fashionmnist, cifar10 and cifar100. The optimal training results obtained for mnist is 1.00 with RMSProp and adam at epoch 200, fashionmnist is 1.00 with rmsprop and adam at epoch 400, cifar10 is 1.00 with rmsprop at epoch 200, cifar100 is 1.00 with adam at epoch 100. The highest testing results are achieved with adam for mnist, fashionmnist, cifar10 and cifar100 are 0.9826, 0.9853, 0.9855, 0.9842 respectively. The analysis of results shows that adam optimization algorithm performs better than others at testing phase and rmsprop and adam at training phase.

In [3], Training neural networks is a computationally challenging problem that requires significant time efforts. In this paper, we propose two approaches that improve efficiency of this task by actively selecting most relevant points from a training data set. The first approach forms a batch that maximizes the reduction of the estimator's entropy, while the second approach only trains on

datapoints whose predicted probability is below a predetermined threshold. Both techniques rely on data metrics to speed up training while retaining the epoch based neural network training framework. The results demonstrate that the proposed methods enable significant reduction of training time in experiments on the CIFAR10 dataset without compromising the accuracy.

3. PROBLEM DEFINITION

Described problem holds significance in the entire area of autonomous devices, which implement a diversified number of classifiers to work independently from human control. The most common autonomous applications can be observed in the automobile and scientific industry, where scientists try to automate the running of vehicles or the analysis of medical scans, which would also output the accurate diagnosis. Both of the mentioned fields are crucial to people as the matter of life depends on a single decision. The goal of automating devices and letting them control the monotonous jobs is to provide the most accurate predictions in the shortest time followed by immediate reaction. With a knowledge of the fastest and most accurate image classification model, multiple industries could advance with the implementation of the most-suited algorithms. However, according to the no free lunch theorem, there is not a single algorithm that will solve every kind of a problem, as each neural network algorithm works differently with different datasets. For example, the CNN model examined in this research should find its usefulness mainly in classifying a low number of classes, made up of tiny images.

The attempt to solve the introduced problem is performed on the widely popular CIFAR-10 dataset, which is commonly used for the evaluation of image classification algorithms. The dataset consists of "60000 32x32 colour images in 10 classes, with 6000 images per class" [3]. The entire set of images is divided into the two sets of 50000 images for the training set and the rest of 10000 images for testing set. The training batch consists of 5000 images from each class, whereas the testing batch has 1000 images representing every class. The CIFAR-10 dataset is a set of the 10 classes divided into six types of animals (bird, cat, deer, dog, frog, horse) and four kinds of vehicles (airplane, automobile, ship, truck). As the publisher informs, the classes are mutually exclusive with no overlaps of similarity, for example, between trucks and automobiles.

The datasets – CIFAR-10 can be downloaded from the official website [5]. The files are available to download in various formats of Python, Matlab and binary version, which is suitable for C programs. After downloading the Python version (cifar-10-python.tar.gz), the data has to be extracted, to reveal the following files:

- batches.meta – consists of a Python dictionary object that defines label names of the 10 included classes.
- data_batch_1, data_batch_2, …, data_batch_5 – training data of 50000 images divided into five files of 30 MB each.
- readme.html – HTML document linking to the official website of the dataset.
- test_batch – testing data of 10000 images in one 30 MB file.

In order to see the actual images and use them, the files have to be decoded with the use of a Python script

IV PROPOSED WORK

The primary object of the current research is to test and compare the performance of various CNN models implemented with the use of scripts written in Python programming language. The image classification pipeline of tiny images used in this project reinvents state of the art, by dropping several components used in a regular CNN model. On a successful implementation, the reinvented model should improve the classification rate. Testing of the deep neural network models is applied on one of the famous image classification dataset, which is the CIFAR-10 dataset, as afterwards the results can be compared with the rest of the published solutions.

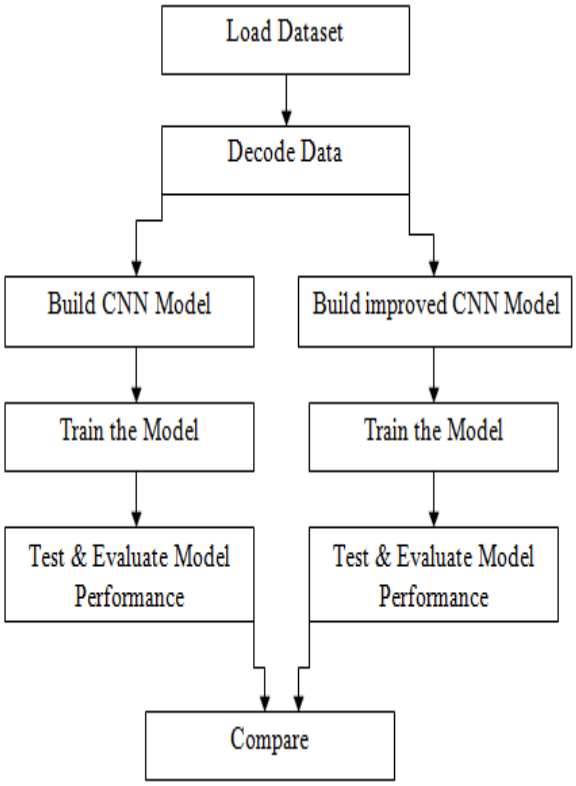

Figure 1. Proposed Block Diagram

Step 1: We can download the CIFAR-10 dataset from official website.

Step 2: Before building a model we can decode the dataset to see the actual images and use them, the files have to be decoded with the use of a Python script.

Step 3: We can build two CNN model and trained that models on training dataset.

Step 4: We can test the model on testing dataset and evaluate the performance result of both the model and compare the performance.

4. EXPERIMENTAL & RESULT ANALYSIS

Running deep neural network models requires a lot of time, All of the software is installed on top of the Python, which enables for Jupyter Notebook that is used to run the entire code. Python programming language is implemented in the decoder file to import CIFAR-10 dataset into a Jupyter Notebook in a compatible format.

In order to run CNN models, Python requires the following packages:

- Tensorflow 1.7.0
- Keras 2.1.5
- Pickle (Python's built-in package)

- NumPy 1.14.2
- Matplolib 2.2.2

The Keras library is the main component of the entire setup, which initialises CNN models [14]. The package allows basing its backend on CPU or GPU component what might vary the experiment running time.

Initially, each Python project requires the import of used packages shown in figure 2 that simplify the process of typing the code. The main component of the following setup is the Keras package running on top of Tensorflow – open source machine learning framework developed by Google, which enables faster implementation of CNN networks in Python script.

```
%matplotlib inline
import matplotlib
from matplotlib import pyplot as plt
import numpy as np
from keras.callbacks import ModelCheckpoint
from keras.layers import Lambda, Conv2D, MaxPooling2D, Dropout, Dense, Flatten, Activation
from keras.models import load_model, Sequential
from keras.optimizers import Adam
```

Figure 2. Import of packages

Secondly, the project requires the import of "decoder.py" file that contains a set of functions written for decoding the CIFAR-10 dataset using the Pickle package and plotting the labels and images data into arrays using the NumPy library. The code of the whole decoder can be accessed from the link on the end of this chapter for later investigation.

The next step requires using the imported functions from the "decoder.py" file to load class names, check the number of classes and define the size of the input image, which is 32 by 32 pixels, and specify the number of channels to be three (red, green and blue). The three arrays of numbers consist of values from 0 to 255 what indicates the pixel intensity at that point. With this information, the CNN can describe the probability of an object being of a specific class.

Another significant operation is to decode and fetch the images. The labels of classes are using integer data type, whereas class is using one-hot encoded vectors. The entire CIFAR-10 data divides into two sets – 83% (50000) of images for training and the rest 17% (10000) for testing, what is verified using the "print()" statement shown in figure 3.

Load the training dataset. Labels are integers whereas class is one-hot encoded vectors.

```
images_train, labels_train, class_train = get_train_data()
```

```
Decoding file: data/data_batch_1
Decoding file: data/data_batch_2
Decoding file: data/data_batch_3
Decoding file: data/data_batch_4
Decoding file: data/data_batch_5
```

Normal labels - Integers

```
print(labels_train)
```

```
[6 9 9 ..., 9 1 1]
```

Classes - One hot encoded labels

```
print(class_train)
```

```
[[ 0.  0.  0. ...,  0.  0.  0.]
 [ 0.  0.  0. ...,  0.  0.  1.]
 [ 0.  0.  0. ...,  0.  0.  1.]
 ...,
 [ 0.  0.  0. ...,  0.  0.  1.]
 [ 0.  1.  0. ...,  0.  0.  0.]
 [ 0.  1.  0. ...,  0.  0.  0.]]
```

Load the testing dataset.

```
images_test, labels_test, class_test = get_test_data()
```

```
Decoding file: data/test_batch
```

Print the size of the training and testing set.

```
print("Training set size:\t",len(images_train))
print("Testing set size:\t",len(images_test))
```

```
Training set size:      50000
Testing set size:       10000
```

Figure 3. Process of decoding and fetching data

Keras library makes building models very intuitive since every layer can be defined in one line of code using "model.add()" function. The code used for the entire operation is self-explanatory and summarises the CNN model using four two-dimensional layers after executing the function shown in Figure 4. Firstly, the model is initialised using a sequential function, which allows building a linear stack of layers treated as a stack of objects, where each of them passes data to the next one. The first two "Conv2D(32, (3, 3)" scripts initialise 32 convolution filters of 3x3 size each, after which another two "Conv2D(64, (3, 3)" scripts use increased number of filters.

```
model = cnn_model()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 32, 32, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 30, 30, 32)        9248
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 15, 15, 32)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 15, 15, 64)        18496
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 13, 13, 64)        36928
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 6, 6, 64)          0
_________________________________________________________________
dropout_2 (Dropout)          (None, 6, 6, 64)          0
_________________________________________________________________
flatten_1 (Flatten)          (None, 2304)              0
_________________________________________________________________
dense_1 (Dense)              (None, 512)               1180160
_________________________________________________________________
dropout_3 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5130
=================================================================
Total params: 1,250,858
Trainable params: 1,250,858
Non-trainable params: 0
```

Figure 4. Simple CNN model building

Afterwards, the model is trained on the training data. "ModelCheckpoint()" method saves the best model after every epoch. The compile method discussed in the previous chapter specifies the loss function, optimiser and metrics for model evaluation. Lastly, the model fits the provided data using a batch size equal to 128, which is the number of samples per gradient update. The model uses 100 iterations (epochs).

After the successful training, the model is evaluated and presents the accuracy and loss on the matplotlib graphs. The IPython notebook includes few scripts for predicting class for the test set of images shown in figure 5.

Predict class for test set images

```
class_pred = model.predict(images_test, batch_size=32)
print(class_pred[0])
```

```
[  2.34778727e-05   3.61718005e-03   7.07750034e-04   6.86623812e-01
   2.35363492e-04   2.97410578e-01   9.98710049e-04   1.62866409e-03
   8.46819486e-03   2.86295195e-04]
```

Get the index of the largest element in each vector

```
labels_pred = np.argmax(class_pred,axis=1)
print(labels_pred)
```

```
[3 8 8 ..., 5 1 7]
```

Check which labels have been predicted correctly

```
correct = (labels_pred == labels_test)
print(correct)
print("Number of correct predictions: %d" % sum(correct))
```

```
[ True  True  True ...,  True  True  True]
Number of correct predictions: 7883
```

Calculate accuracy using manual calculation

```
num_images = len(correct)
print("Accuracy: %.2f%%" % ((sum(correct)*100)/num_images))
```

```
Accuracy: 78.83%
```

Figure 5. Sample predictions

Improved CNN Model

Firstly, the notebook requires importing a few additional functions from Keras and decoder packages. The process of importing class names as well as fetching and decoding the data remains unchanged.

Definition of the improved CNN model consists of the most essential changes shown in figure 6. As mentioned in the introduction, the improved model replaces the max-pooling and dense function with two-dimensional convolution layers. The architecture uses nine layers with a different number of convolution filters. In the end, the model uses the operation of two-dimensional global average pooling. After the structure definition, the model is built and summarised.

```
model = improved_cnn_model()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_10 (Conv2D)           (None, 32, 32, 96)        2688
_________________________________________________________________
dropout_4 (Dropout)          (None, 32, 32, 96)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 32, 32, 96)        83040
_________________________________________________________________
conv2d_12 (Conv2D)           (None, 16, 16, 96)        83040
_________________________________________________________________
dropout_5 (Dropout)          (None, 16, 16, 96)        0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 16, 16, 192)       166080
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 16, 16, 192)       331968
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 8, 8, 192)         331968
_________________________________________________________________
dropout_6 (Dropout)          (None, 8, 8, 192)         0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 8, 8, 192)         331968
_________________________________________________________________
activation_4 (Activation)    (None, 8, 8, 192)         0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 8, 8, 192)         37056
_________________________________________________________________
activation_5 (Activation)    (None, 8, 8, 192)         0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 8, 8, 10)          1930
_________________________________________________________________
global_average_pooling2d_2 ( (None, 10)                0
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
Total params: 1,369,738
Trainable params: 1,369,738
Non-trainable params: 0
```

Figure 6. Improved CNN model building

The model is trained using the same parameters, where the only difference is the increased number of 350 epochs. and the prediction result are shown in figure 7.

Evaluate the model

```
scores = model.evaluate(images_test, class_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 84.96%
```

Figure 7. improved Model prediction

Comparison

The accuracy, as well as running time of all the tested models, are presented in the following table.

Table 1. Accuracy Of The CNN Models

Classification Model	Accuracy
Simple CNN	78.83%
Modified CNN	84.96%

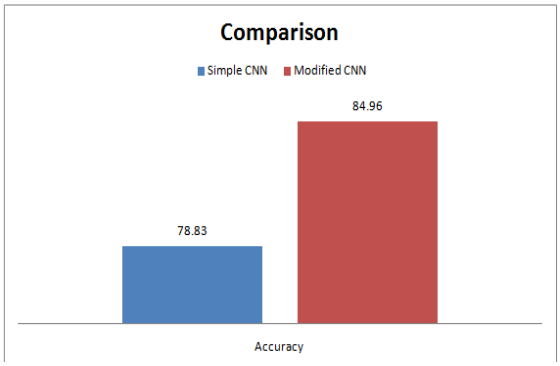

Figure 8. Comparison of models

The results indicate 10% improvement between the simplest and the most advanced model used in the experiment. CNN model after dropping max-pooling and dense function did improve its accuracy up to 87.94%; however, as of running three times more epochs, the running time increased for the improved CNN.

Now we illustrate the model accuracy and model loss of each of the tested CNN model shown in figure 9.

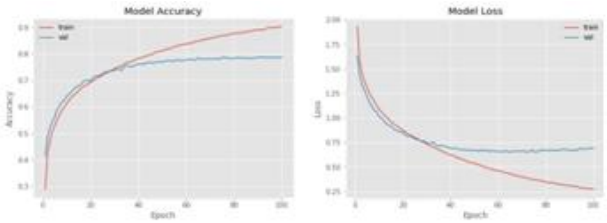

Figure 9. Simple CNN model accuracy and loss

From the first figure, it is observed that the model stopped improving its accuracy after about 60 epochs, whereas the loss stabilised after about 40 epochs.

The second graph shown in figure 10 of the improved model indicates identical situation as the simple CNN model. The model similarly stopped improving its accuracy after about 60 epochs, whereas the model loss started slowly increasing after about 150 epochs resulting with overfitting the data.

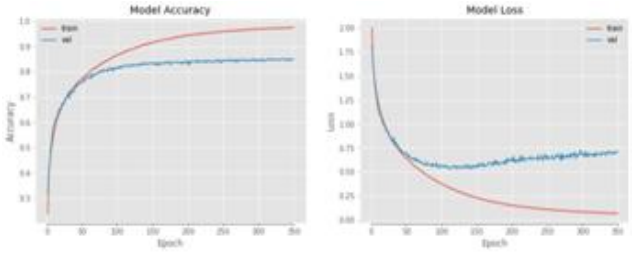

Figure 10. Improved CNN model accuracy and loss

5. CONCLUSION

The results obtained in this research are important as they indicate that it is possible to increase the accuracy of CNN models by easily running and modifying its traditional structure with the use of programming language. One of the interesting conclusions is the ratio between accuracy and running time, as the last model required the most computational power to achieve the best accuracy, whereas the most traditional CNN structure obtained the best running time to accuracy ratio. Basing on the possessed components, the AI implementers would have to decide if it is worth to go for the more robust models.

6. REFERENCES

[1] Yifeng Zhao1 ,Weimin Lang1 ,Bin Li2 "Performence analysis of neural network with improved weight training process" in IEEE 2019.

[2] Raniah Zaheer , Humera Shaziya "A Study of the Optimization Algorithms in Deep Learning" in IEEE 2019.

[3] Sara Mourad, Haris Vikalo and Ahmed Tewfik "ONLINE SELECTIVE TRAINING FOR FASTER NEURAL NETWORK LEARNING" in IEEE 2019.

[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for Large-Scale image recognition," Sep. 2014.

[5] S. Liu and W. Deng, "Very deep convolutional neural network based image classification using small training sample size," in 2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR), Nov. 2015, pp. 730–734.

[6] Y. Chen, Y. Yang, W. Wang, and C. C. Jay Kuo, "Ensembles of feedforward-designed convolutional neural networks," Jan. 2019.

[7] Y. Huang, Y. Cheng, A. Bapna, O. Firat, M. X. Chen, D. Chen, H. Lee, J. Ngiam, Q. V. Le, Y. Wu, and Z. Chen, "GPipe: Efficient training of giant neural networks using pipeline parallelism," Nov. 2018.

[8] M. Tan and Q. V. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," May 2019.

[9] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "AutoAugment: Learning augmentation policies from data," May 2018.

[10] N. Nayman, A. Noy, T. Ridnik, I. Friedman, R. Jin, and L. Zelnik-Manor, "XNAS: Neural architecture search with expert advice," Jun. 2019.

[11] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in 2009 IEEE Conference on Computer Vision and Pattern Recognition, Jun. 2009, pp. 248–255.

[12] IPython Development Team (2018) Ipython Interactive Computing [online] available from <https://ipython.org/> [26 April 2018]

[13] Project Jupyter (2018) Jupyter [online] available from <http://jupyter.org/> [26 April 2018]

[14] Chollet, F. (2018) Keras Documentation [online] available from<https://keras.io/> [26 April 2018]

[15] Google Brain Team (2018) Tensorflow [online] available from<https://www.tensorflow.org/> [21 April 2018]