IMAGE CLASSIFICATION USING CONVOLUTIONAL NEURAL NETWORKS - AN EXPERIMENTAL APPROACH FOR PERFORMANCE ANALYSIS

Xiaowen Li

ABSTRACT

Convolutional Neural Networks (CNN) have been shown to be a powerful method to solve image classification problems and produce state-of-art accuracy training results. While $CIFAR-10$[1] data set is a well-known data set that is commonly used in the machine learning field for bench-marking. The purpose of this paper is to construct a baseline structured CNN and explore how its performance is related to different hyper-parameters(Number of epochs, Type of optimizer, Loss function, Learning rate, Batch size, and Regularization). The performance is represented by test accuracy and loss and the experiment generates such results each time we fine-tune the network with a different combination of hyper-parameters. The experimental analysis shows that while the baseline network archives 70.29% accuracy, the fine-tune networks produce 71.24% correct classification rate as the best result. This paper also discussed how changing of above hyper-parameters affects the accuracy of CNN.

1. INTRODUCTION

Artificial neural network(ANN) in machine learning is inspired by the neural network that functional in human beings' brain. ANN is formed by connected artificial neurons that is also a simulation of real-human brain neurons. Just like we get information from visual system and send it to our brain, after it processes we knows what is in front of eyes by best of our knowledge. ANN is like such a 'process'. Convolutional neural network(CNN) is a class of ANN. Just like the name said, CNN consists of one or more convolutional layers and then fully connected layer on top, pooling layers also can be applied. CNN is most commonly applied with large size image classification. In this paper, we also focus on CNN in the image classification application. In deep learning society, image classification accounted for a large proportion of the attention, especially in medical image analysis field. The image classification takes images as input and output identifications of disease type or if it is present or not.[2] With large size of training data, neural network image classification can identify more accurately and more efficiently than a human doctor. This also applies for other fields, we are know living in the era of data, and images takes a significant portion of data sharing worldwide. Classifying such huge amount of images accurately and time-efficiently is the job for image classification.

This paper will take one pre-built dataset for image classification - $CIFAR-10$[1]. It contains 60k images with 10 classifications which represents a rich variety of internet figures. We start with implementing a baseline network, see figure 1 for the detailed architecture of it. And fine-tune this network by make change of it's hyper-parameters, see session 3 for options of parameters, in total we had $3^5 + 5 = 248$ combinations of hyper-parameters tested. Each time we fine-tuned a network, we compare its test accuracy with our baseline. Our goal is to see how change of hyper-parameters affects the accuracy and see if we can get better accuracy than the baseline.

2. RELATED WORKS

There are lots of works on improving the accuracy of $CIFAR-10$ dataset. As by today, the known highest accuracy is 99.37% by Kolesnikov, Beyer, zhai et al[3]. In their paper they use a pre-trained representations on large supervised datasets and transfer the model on $CIFAR-10$, they name such pre-trained model Big Transfer and it also performs well on a wide range of image datasets. Cubuk[4] performs a 98.53% accuracy on $CIFAR-10$. In his paper, instead of manually designing the data augmentation for the classifier, he came up with a procedure that can automatically search for improved data augmentation policies. Recurrent neural network(RNN) is mostly used for 1 dimensional input datasets, but Phong[5] explored it's possibility on classifying 2 dimensional images. He used RNN as additional layers for multi image classification models and successfully performed a state-of-art accuracy on $CIFAR-10$ as 98.36%. Yamada[6] approches the goal by studying to avoid over-fitting issue in deep networks. His study was mainly based on ResNet and its improvements. A new regularization method is introduced based on Shake-Shake that give the stabulity of training, he made 97.69% accuracy on $CIFAR-10$. Liang's work[7] had a similar motivation - using appropriate regularization method to avoid over-fitting problem. His idea is to drop nonlinear activation functions by setting to randomization. His work also gives a high accuracy on $CIFAR-10$ dataset, which is 96.55%.

3. METHOD

We start on a network with the following architecture[8]: 32 filter 3×3 convolutional layer with ReLu activation function add-on, 2×2 max pooling layer, 64 filter 3×3 convolutional layer with ReLu, following by another 2×2 max pooling layer, 128 filter 3×3 convolutional layer with ReLu, followed by 256 filter 3×3 convolutional layer, two 2048 neuron densely connected layer, and finally a softmax activation output layer with 10 possible classes. As shown in figure 1.

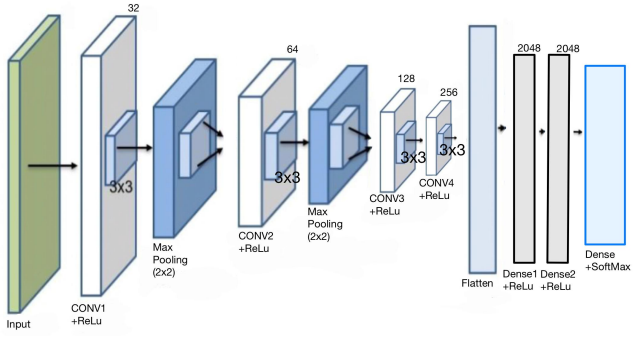

Fig. 1: Baseline Architecture Figure

There are hyper-parameters can be modified for this network, and we give following[8] options for each parameter:

1. Number of epochs – 1, 5, 10, 15, 25
2. Optimizer – Adam, SGD, RMSprop
3. Loss function– SparseCategoricalCrossEntropy, Poisson, KLDivergence
4. Learning rate – 0.1, 0.01, 0.001
5. Batch size – 128, 256, 512
6. Regularization – l1 regularization(0.2), l2 regularization(0.2)

For the baseline network, we define the hyper-parameters as following: Adam optimizer with learning rate equates to 0.01, SparseCategoricalCrossEntropy loss function, SparseCategoricalAccuracy metrics, a 128 batch size and 25 epochs with no regularization applied to dense layers.

4. EXPERIMENTS AND RESULTS

4.1. Datasets and Architectures

We ran experiments on the CIFAR-10 datasets[1]. It contains $50k$ training 32×32 RGB images and $10k$ for testing. All images from dataset are labeled as one of the ten categories. Break down to each category, the dataset has $5k$ training and $1k$ testing images.

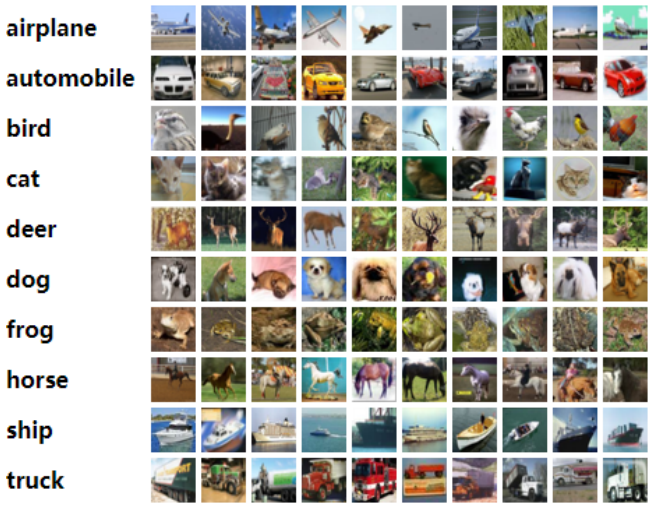

Fig. 2: Example images from CIFAR-10 for each class[1]

4.2. Implementation and Required Packages

We run our experiments on TensorFlow 2 with python version 3.9. Keras API is used in experiments.

4.3. Experiment Methods

Our experiments is aimed to explore the relation between hyper-parameters and CNN's performance. For this purpose, we fine-tune our CNN with all possible hyper-parameter combinations, both network structure and hyper-parameters are described in previous section 3. See figure 3 for how combinations were taken.

```
import os

Regularization = ['none','l1', 'l2']
Optimizer  = ['Adam', 'SGD', 'RMSprop']
Loss_function = ['sparse_categorical_crossentropy', 'Poisson', 'KLDivergence']
Learning_rate = [0.1, 0.01, 0.001]
Batch_size = [128, 256, 512]

for g in Regularization:
    for o in Optimizer:
        for f in Loss_function:
            for r in Learning_rate:
                for b in Batch_size:
                    os.system("python dlproj1.py "+g+" "+o+" "+f+" "+str(r)+" "+str(b))
```

Fig. 3: All combinations of hyper-parameters ran in experiments

4.4. Experiment Observations

From above combinations the lowest accuracy and also is the commonest accuracy we got from the network is 0.10000000149011612. This number might seems odd as it looks completely arbitrary but appears frequently. Some combinations of hyper-parameters that results in this accuracy can be seen in table 1. But if we have some knowledge on hardware's binary floating-point arithmetic, we know this number is $0.1f$ in python environment. Then how this number appears to be the lowest and commonest accuracy can

now be easily explained - As we introduced above 4.1, the dataset we use have $1k$ test images for each of the classification and in total we have $10k$ test images. When the network produces a pretty high test loss, the trained model cannot get any advances on how to identify the images into different classes, which results in giving the same class for all testing images - and therefore the accuracy is $\frac{1k}{10k} = 0.1$.

Optimizer	Loss function	Learning Rate	Batch Size	Regularization	Test Loss
Adam	sparse_categorical_crossentropy	0.1	256	none	2.304537296
Adam	Poisson	0.01	256	none	65.37831116
RMSprop	sparse_categorical_crossentropy	0.01	128	none	2.303021193
Adam	KLDivergence	0.1	512	l1	13461.16016
SGD	Poisson	0.01	128	l2	10.46163368

Table 1: Some examples on combinations that produces 0.1 accuracy, with fixed number of epochs=10

The highest accuracy we got among all combinations is 0.7124000191688538, which happens with the following selected hyper-parameters: Adam optimizer with learning rate 0.001, sparsecategoricalcrossentropy loss function, a batch size 256 and 10 epochs, no regularization applied. Comparing to our baseline architecture3 which produces 0.7028999924659729 test accuracy, we see that the accuracy increases while batch size increases but number of epochs decreases. To be able to evaluate relation of accuracy with one individual hyper-parameter, we also post the accuracy with 128 batch size but 10 epochs, which is 0.7056000232696533.

Optimizer	Loss function	Learning Rate	Batch Size	Regularization	Number of Epochs	Test Accuracy	Test Loss
Adam	SCC	0.001	128	none	25	0.7028999925	2.011043787
Adam	SCC	0.001	128	none	10	0.7095999718	1.076698184
Adam	SCC	0.001	256	none	10	0.7124000192	0.9631000161
Adam	SCC	0.001	512	none	10	0.6837000251	0.9629266858

Table 2: baseline vs. highest accuracy structure

Number of Epochs We can see that with the same batch size, a smaller number of epochs results in a better accuracy - how does this happen? We can take a look at the test loss of both networks (table 2), with 10 epochs, the test loss is smaller than with 25 epochs, this means we already passed the point that test loss is minimized and more epochs result in loss is growing back. Number of epochs is not a simple larger or smaller is better scenario, it is highly determined by the size of the model and the variation in the dataset. In our experiment, out of $1, 5, 10, 15, 25$, we found that number of epochs = 10 gives the highest accuracy while other hyper-parameters are fixed, so we will fix the number of epochs = 10 for all the rest of experiments.

Batch Size Now with a fixed number of epochs to be 10, we see from table 2 that a batch size 256 has better accuracy than batch size 128. Does this mean larger the batch size higher the test accuracy for the neural network? The answer is also a NO. Similar to the number of epochs, there is a 'turning point' that if when we approach this batch size, the accuracy got better, but going behind or beyond this point will both result in decreasing in accuracy. Too large of a batch size will lead to poor generalization. An example is from table 2 as the batch size continuous increasing from 256 to 512, the test accuracy drops.

Loss function As listed in figure 3, in total we tried $3^5 = 243$ combinations of hyper-parameters.Out of all 243 combinations, we have 36 combinations that achieved test accuracy above 0.2. And surprisingly, all such 36 runs used sparsecategoricalcrossentropy as the Loss function. Sparse Categorical Cross-entropy can be considered when we have more than 2 classifications and all classes can be represented by integers. This fits our experiment - the dataset $CIFAR-10$ contains 10 classifications and although we know each of them indicates an image type, but the dataset does use numerical labels from 0 to 9. While for Poisson loss function, it is more ideal when the data is from a Poisson distribution, but $CIFAR-10$ is not. Similar excuse for Kullback-Leibler Divergence(KLDivergence) Loss function - KLDivergence is designed for measuring distance for continuous distributions. From our experiment results, we can see only the Sparse Categorical Cross-entropy loss function gives relatively good accuracy for the dataset.

Learning Rate Now based on above observations, we fix the loss function to be sparsecategoricalcrossentropy and apply no regularization to dense layers and only take a look at how varies of learning rate affects the test accuracy.

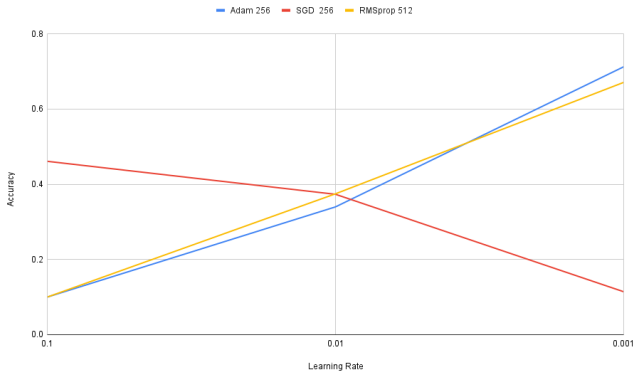

Fig. 4: Leaning Rate = 0.1, 0.01, 0.001 with fixed loss function = sparsecategoricalcrossentropy and regularization = none, number of epochs = 10

We picked 3 most representative sets of testing accuracy and plot figure 4, each line represents one set of accuracies. The blue line shows accuracy at learning rate $= 0.1, 0.01, 0.001$ with Adam optimizer at batch size $= 256$, the red line is generated with SGD optimizer and a batch size $= 128$, the yellow is generated with RMSprop optimizer and a batch size $= 512$.

Some interesting phenomenons can be observed from this figure 4: While the learning rate is getting smaller and smaller, the test accuracy gets better for both Adam and RMSprop optimizer, but worse for SGD optimizer. 10^{-3} is the

learning rate recommended in the Adam paper[9], a large learning rate with results in faster initial learning before the rate is updated. Same for RMSprop, the creator of RMSprop optimizer, Hinton also claimed that the best learning rate for RMSprop is 0.001[10]. For the SGD optimizer, the accuracy drops when accuracy becomes small, our thought is that while finding the gradient, too small the learning rate results in too little a step toward the minimum. This means our experiment does not wait for it to converge.

Optimizer Again we will fix loss function = sparse-categoricalcrossentropy, regularization = none and number of epochs = 10 for best comparable results.

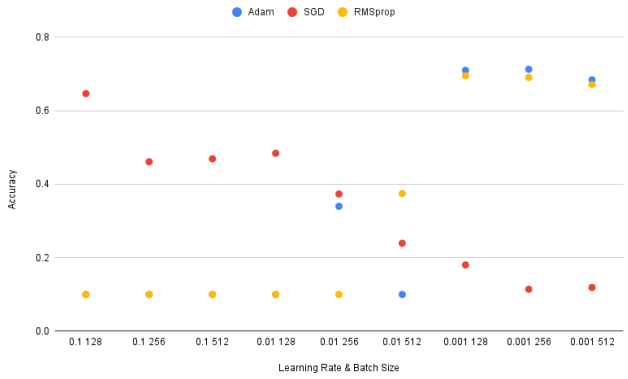

Fig. 5: Optimizer = Adam, SGD, RMSprop with fixed loss function = sparsecategoricalcrossentropy and regularization = none, number of epochs = 10

From figure 5, we see that Adam and RMSprop had similar performance and reach around 70% accuracy with a learning rate $= 0.001$. While from the last paragraph we know that SGD optimizer works better with a larger learning rate like 0.1, it only reaches 0.646300017833709 as the highest accuracy.

Regularization We also applied different regularization to both of our dense layers. We had three options on applying the regularization - no regularization, l1 regularization to both dense layers with a dropout$= 0.2$, and l2 regularization to both dense layers with a dropout$= 0.2$. From figure3 we see that for each option we had $3^4 = 81$ outputs. Out of 81 results, 21 combinations without applying any regularization achieves an accuracy above 0.4, none of the combinations with l1 regularization achieves an accuracy above 0.4, and only 1 combination with l2 regularization achieves an accuracy above 0.4.

4.5. Comparison with state-of-Art results

Author(s)	Key point	Accuracy
Tan, Le[11]	Model Scaling - EfficientNets	98.90%
DeVries, Taylor[12]	Regularization - Cutout	97.44%
Huang, Liu, Maaten et al[13]	Fully Connection - DenseNet	96.54%
Zeiler, Fergus[14]	Stochastic Pooling	84.87%
McDonnell, Vladusich[15]	Classifier weight	75.86%

vs

Our Accuracy
71.24%

Table 3: 5 state-of-the-art works that use CNN and CIFAR-10

5. CONCLUSION

In exploring all combinations of hyper-parameters to our baseline CNN architecture, we achieved the highest test accuracy of 71.24% with Adam optimizer, sparse categorical cross-entropy loss function, 0.001 learning rate, 128 batch size, and no regularization. We observe that with 10 epochs the network output the best accuracy; Batch size is also not the larger the better - sometimes a batch size 512 will give worse accuracy compared to 256; Sparse Categorical Cross-entropy is the best loss function to fit $CIFAR-10$ dataset; learning rate is based on the optimizer - both Adam and RMSprop optimizer gives better accuracy when the learning rate is 0.001, but SGD optimizer prefers larger learning rate like 0.1; Applying regularization seems to be a bad idea for our network structure and the dataset we use.

6. REFERENCES

[1] Alex Krizhevsky, Geoffrey Hinton, et al., "Learning multiple layers of features from tiny images," 2009.

[2] K. Balaji and K. Lavanya, "Chapter 5 - medical image analysis with deep neural networks," in *Deep Learning and Parallel Computing Environment for Bioengineering Systems*, Arun Kumar Sangaiah, Ed., pp. 75–97. Academic Press, 2019.

[3] Alexander Kolesnikov, Lucas Beyer, Xiaohua Zhai, Joan Puigcerver, Jessica Yung, Sylvain Gelly, and Neil Houlsby, "Big transfer (bit): General visual representation learning," 2020.

[4] Ekin D. Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V. Le, "Autoaugment: Learning augmentation policies from data," 2019.

[5] Nguyen Huu Phong and Bernardete Ribeiro, "Rethinking recurrent neural networks and other improvements for image classification," 2021.

[6] Yoshihiro Yamada, Masakazu Iwamura, Takuya Akiba, and Koichi Kise, "Shakedrop regularization for deep residual learning," *IEEE Access*, vol. 7, pp. 186126–186136, 2019.

[7] Senwei Liang, Yuehaw Khoo, and Haizhao Yang, "Drop-activation: Implicit parameter reduction and harmonic regularization," 2020.

[8] S. Canavan, "Project 1.pdf," University of South Florida, 2022, pp. 1–4.

[9] Diederik P. Kingma and Jimmy Ba, "Adam: A method for stochastic optimization," 2017.

[10] Geoffrey Hinton, "Neural networks for machine learning lecture 6a," .

[11] Mingxing Tan and Quoc V. Le, "Efficientnet: Rethinking model scaling for convolutional neural networks," 2020.

[12] Terrance DeVries and Graham W. Taylor, "Improved regularization of convolutional neural networks with cutout," 2017.

[13] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger, "Densely connected convolutional networks," 2018.

[14] Matthew D. Zeiler and Rob Fergus, "Stochastic pooling for regularization of deep convolutional neural networks," 2013.

[15] Mark D. McDonnell and Tony Vladusich, "Enhanced image classification with a fast-learning shallow convolutional neural network," 2015.