CONVOLUTIONAL NEURAL NETWORKS FOR CIFAR–10 IMAGE CLASSIFICATION

Project Report

ECE 6254: Statistical Signal Processing
Instructor: Prof. Justin Romberg
Spring 2016

Chia-Sheng Hsu (chiasheng@gatech.edu)
Jingting Yao (jingtingyaobecky@gatech.edu)
Rama Mythili Vadali (mythili.vadali@gatech.edu)
Yi-Chi Shao (ycshao@gatech.edu)

1. Project summary

In this project, convolutional neural networks with different layer structures are implemented and investigated for image classification. A convolutional neural network (CNN) is a type of feed-forward neural network that exploits local connectivity patterns between neurons of adjacent layers. A common CNN mainly consists of stacks of convolutional layers (with an activation function at the end), pooling layers and fully-connected layers [1]. A convolutional layer maps local features of an input layer into several output layers (feature maps), where all the neurons in the same feature map share the same weights and biases at different locations of the input layer. Pooling layers are usually used right after convolutional layers. What they do is down-sampling output layers from the convolutional layer in order to greatly reduce the number of parameters to be learned. The last stage of a CNN is the usual fully-connected layers as in regular neural networks.

The goal of this project is to investigate the performance of image classification with different CNN architectures based on a proven model. The baseline model we use in this project is a tiny VGGNet [2] model, a CNN architecture that achieved 7.3% error (ranked second) in the classification task of ImageNet ILSVRC-2014 [2]. The core building block of VGGNet is a stack of 2 convolutional layers followed by a maximum pooling layer and a dropout layer (regularization layer). After several stacks of such building block are fully-connected layers and a softmax classifier. We apply several different variations of VGGNet architecture to the CIFAR-10 dataset, which contains 60,000 32x32 color images in 10 classes [3]. In addition to the original VGGNet architecture, the variations we have implemented are removal of dropout layers, change of activation function from ‘ReLU’ to ‘sigmoid’ for the last activation layer, inclusion of 2 additional convolutional layers, and change of stochastic gradient descent batch size from 32 to 16. We also realized the visualization of feature maps and shared weights from the first convolutional layer so as to get an idea of what the CNN learns from layer to layer. Finally, we evaluate the performance of each classifier with prediction accuracy, confusion matrices and receiver operating characteristic (ROC) curves. It is shown that the model without the dropout layers tends to overfit the input data, resulting in higher training accuracy than that with such regularization. Moreover, we found it interesting that the animals are harder to predict than the vehicles possibly because different animals may have highly similar features such as outlines, which are important for classification. A third valuable result of this project is that using sigmoid activation in the last layer in combination with ReLU for other layers achieves better performance compared to ReLU for all activations which was implemented in the original VGGNet architecture.

2. Introduction

CNNs are specialized artificial neural network models which exploit the spatial correlation and variabilities of 2D shapes to extract features, and therefore are one of the best among all the online and offline image classification networks as to the literature search so far [2]-[5]. Based on knowledge of ANNs and their layered architectures, this project is intended to build up understanding of the architecture of CNNs and obtain experience with training on a set of labeled images and consequently to predict the category of a random image.

CNNs [6], [7] are similar to ordinary Neural Networks, but make the explicit assumption that there is spatial correlation in the input data. This makes CNN architectures well-suited learning algorithms for images and speech. Like ordinary Neural Networks, CNNs are made up of neurons that have learnable weights and biases. Each neuron receives some inputs, performs a dot product and optionally follows it with a non-linearity. The whole network still expresses a single differentiable score function: from the raw image pixels on one end to class scores at the other. And they have a loss function like SVM or Softmax on the last (fully-connected) layer.

To take advantage of the fact that the input consists of images, each layer of CNN has neurons arranged in 3 dimensions: width, height, and depth (for RGB). The neurons in a layer will only be connected to a small region of the layer before it, instead of the neurons connected in a fully-connected manner. Every layer of CNN transforms the 3D input volume to a 3D output volume of neuron activations. The final output layer for CIFAR-10 would have dimensions 1x1x10, which represents a single vector of class scores arranged along the depth dimension.
Three main types of layers make up CNN architectures: **Convolutional layer, Pooling Layer, and Fully-Connected layer**. These layers are stacked up to form a full CNN architecture.

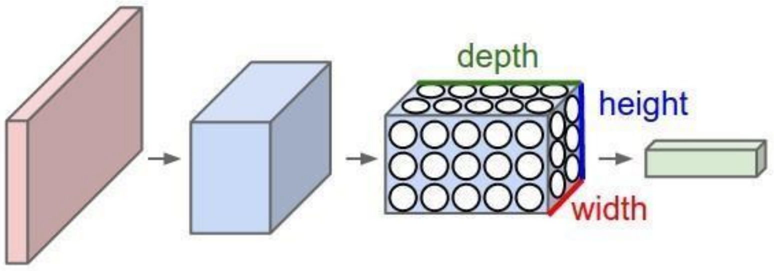


Figure 1: A CNN arranges its neurons in three dimensions, as visualized in the blue layers. The red input layer holds the image.

The Convolutional Layer is the core building block of a Convolutional Network, and its output volume can be interpreted as holding neurons arranged in a 3D volume. The output of convolutional layer depends on three hyperparameters: depth, stride and zero-padding. Depth controls the number of neurons in the layer that connect to the same region of the input volume. Stride with which we allocate depth columns around the spatial dimensions (width and height). When the stride is 1, then we will allocate a new depth column of neurons to spatial positions only 1 spatial unit apart. This will lead to heavily overlapping receptive fields between the columns, and also to large output volumes. Conversely, if we use higher strides then the receptive fields will overlap less and the resulting output volume will have smaller dimensions spatially. Note that the description of CNNs in this section has been adapted primarily from [6].

3. Methodology

Our implementation is based on an open source API TensorFlow [8], and a highly modular neural networks library, Keras [9], which is written in Python and capable of running on top of TensorFlow. Due to the limitation of computational resources, we chose CIFAR-10 dataset which has 60,000 32x32 tiny images from 10 classes, with each class having 1000 images. The 10 categories are airplane, automobile, bird, cat, deer, dog, frog, horse, ship and truck. The training set has 40,000 images, validation set has 10,000 and testing set has 10,000.

Figure 2 is an illustration of our classification system. The input is a 3D volume data. Data augmentation is applied during training to artificially increase the number of training examples by applying flipping and rotating input images. This is a more economic way to deal with the wealth of the features that already have. After going through multiple types of layers (which is discussed in the following paragraph) is the fully connected layer in which neurons have full connections to all activations to the previous layer. The last layer uses softmax to provide the classification results.

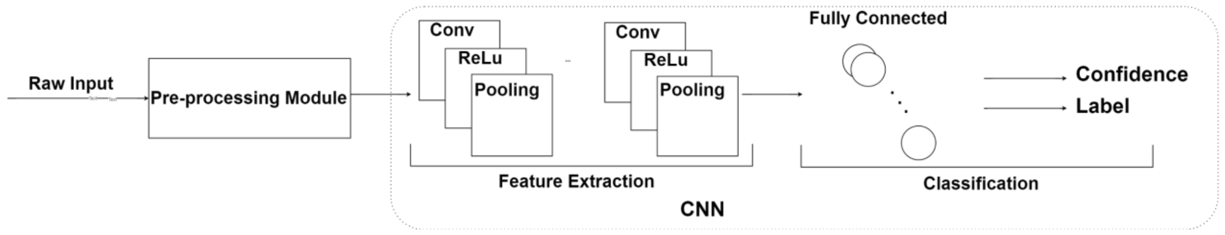


Figure 2. Diagram of the classification system

The detailed architecture of the baseline model is shown in Table I. This is a scaled down variation of the VGGNet architecture. It has four convolutional layers and two dense (fully connected) layers. Stochastic gradient descent [10] is used to find the weights of the network. The batch size, which is the number of samples per gradient update, is 32.

Table I. Architecture of the baseline model

Input Layer (32*32) RGB Images
Data Augmentation (random flip and rotate)
Conv3-32 activation-relu **Conv3-32** activation-relu maxpool Dropout
Conv3-64 activation-relu **Conv3-64** activation-relu maxpool Dropout
Flatten **Dense-512** activation-relu Dropout **Dense - 10** Activation-Softmax

Initially we ran the baseline model for 200 stochastic gradient descent epochs. The plot in Figure 3 illustrates convergence over 200 epochs for the baseline model. Based on this initial experiment, it seemed like the model converges approximately at 50 epochs. To make the best use of our computing resources, we decided to perform our training for 50 epochs. This reduced the training time from ~40 hours to ~10 hours for each experiment.

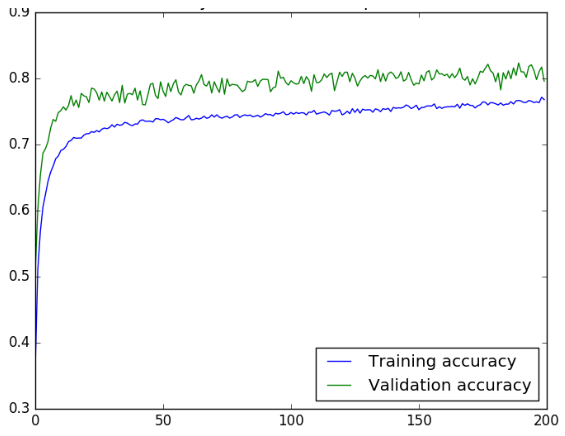


Figure 3. Convergence of the baseline model

4. Results and Discussion:

4.1. Accuracies

Testing was performed on test image set of size 10000.

Table II. A summary of training and testing results

Model variations	Training Accuracy	Validation Accuracy	Testing Precision	Testing Recall	Testing f1-score
Baseline Model	0.73	0.78	0.79	0.78	0.78
Without dropout layer	0.83	0.80	0.80	0.80	0.80
Replacing ReLU Activation with Sigmoid	0.78	0.82	0.81	0.81	0.81
Two additional layers (total 8 layers)	0.69	0.74	0.76	0.73	0.74
SGD Batch size of 16	0.47	0.58	0.60	0.58	0.58
With sample-wise preprocessing	0.64	0.13	0.32	0.13	0.08
With feature-wise preprocessing	0.69	0.21	0.41	0.21	0.15

Following are the key observations based on the results:

Absence of dropout layer and overfitting: The model without dropout layers has higher training accuracy in comparison with corresponding validation accuracy and testing metrics. This indicates that the absence of dropout leads to overfitting. The key idea of dropout is to randomly drop units (along with their connections) from the network during training. This prevents units from co-adapting too much. This is reflected in the performance metrics of the model without dropout layers [11]. The baseline model has dropout layers included, and therefore generalizes well.

ReLU vs Sigmoid for Activation: The sharp cutoff of the ReLU activation function at zero makes the output sparser and reduces likelihood of vanishing gradient [12]. The more ReLU layers exist in the model, the sparser the output representation. On the other hand, Sigmoid tends to generate more non-zero outputs and results in dense representations. Applying sigmoid function preserves more output features of the corresponding layer, and thus could potentially provide a richer set of features for training. This could be the reason that replacing one ReLU activation function to sigmoid leads to better performance.

Upon replacing all the ReLU layers with Sigmoid, the performance was found to be very bad with accuracies in the range of 0 -1. The reason is that sigmoid has the drawback of saturating and 'killing' gradients. When the neuron's activation saturates, the gradient at theses region would be near zero. In backpropagation phase, the near zero gradient value would cause almost no signal to flow through the neuron. So when the initial weights are too large, most neurons would become saturated and the network will barely learn.

Furthermore, sigmoid function has range [0,1] whereas the ReLU function has range [0,∞]. Hence sigmoid function can be used to model probability. Upon replacing the last activation layer with sigmoid instead of ReLU, we found the performance improves. The reason for this could be that the continuous range from 0 to 1 of a sigmoid gives more detailed information for classification that follows, rather than rectification operation of ReLU.

More layers and mini-batch [13]**:** For more layers and smaller batch size, the results are not comparable since both of them are not converging in 50 epochs. More number of epochs may lead to a better performance. The convergence analysis is discussed in depth in section 4.5.

Preprocessing [14]**:** It is evident from Table II that preprocessing results in worst performance. The reason for this could be how we implemented normalization in preprocessing. After normalization by subtracting means, many RGB values in the image input will be rescaled to a range that is at least partially negative. When this input goes through a ReLU activation function which eliminates negative values, considerable negative input values become zero. This information loss makes learning ineffective.

4.2. Confusion matrix

A confusion matrix [15] is a table that is often used to describe the performance of a classification model on a set of test data for which the true values are known. A confusion matrix will summarize the results of testing the algorithm for further inspection. In the confusion matrices presented below, the percentage of test data that belongs to each element of the matrix has be mapped to a color scale. Bright color along the diagonal indicate the correct prediction of true class.

Figure 4 shows the confusion matrix generated after testing the baseline model. Similar matrices were constructed for other experiments. Across different models it can be observed that performance is best for airplane, automobile, ship and truck in comparison with animal classes such as bird, cat, deer, dog, frog and horse. This could be because human-build objects have limited and more distinguishable feature sets. For instance, their curves and edges are more well-defined than animal classes. Most confusion is between dog and cat, this is because they differ primarily in facial features.

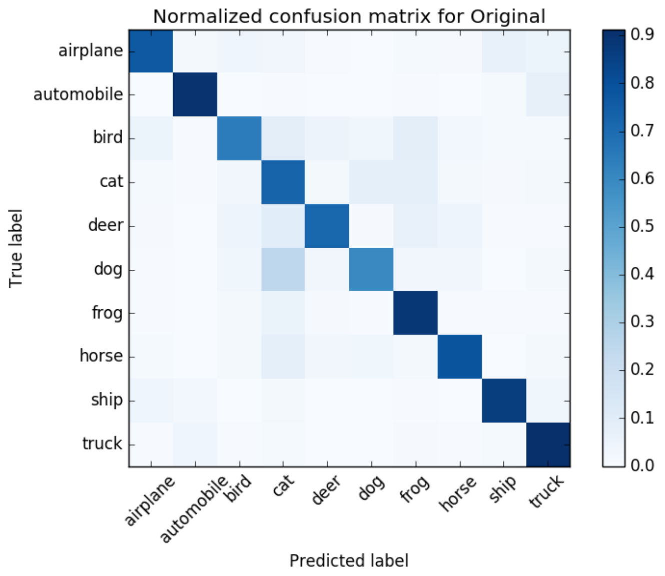


Figure 4. Confusion matrix Baseline Model:
6 layers, No preprocessing, SGD batch size 32, Dropout included, ReLU Activation

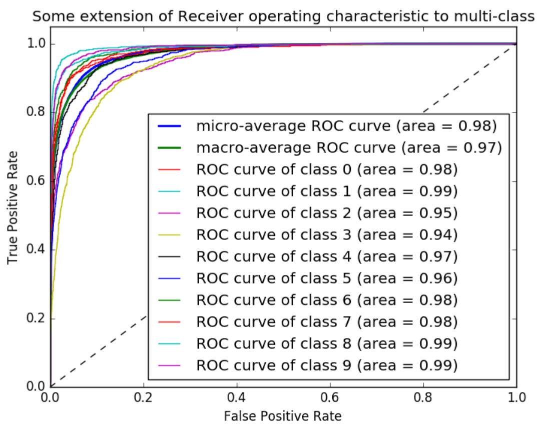
4.3. Receiver Operating Characteristic (ROC) Curve and Area under ROC Curve (AUC)

The receiver operating characteristic (ROC) curve [16] of a classification model is a two dimensional curve generated by varying the threshold set for classification. The point on a curve for a given threshold is (False Positive Rate (FPR), True Positive Rate (TPR)), where

$$TPR = TP/(TP+FN)$$
$$FPR = FP/(FP+TN)$$

For a random model that randomly performs classification the ROC curve is a straight line from (0,0) to (1,1).

Area under ROC (AUC) curve summarizes the performance of a model using a single number. The range of AUC is from 0 to 1 - the higher the better. A random model has an AUC of 0.5. The plot on the left in Fig. 5 shows the ROC curve for each class corresponding to the prediction of the baseline model. It can be seen that the performance is worst for class 3 corresponding to ‘cat’ classification, which is consistent with the results from confusion matrix. Next to it is the plot that compares the ROC and AUC for different models we trained. It can be concluded that sigmoid variation gives the best performance, next to which is the without dropout variation, followed next is the baseline model.



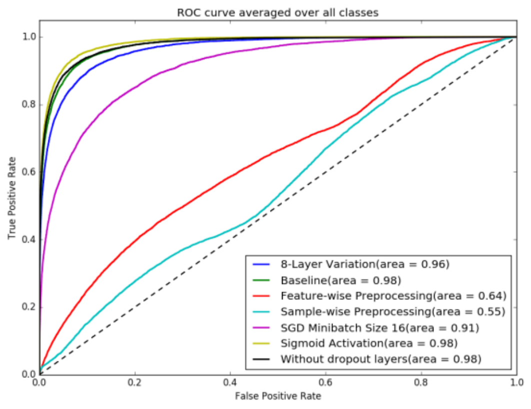
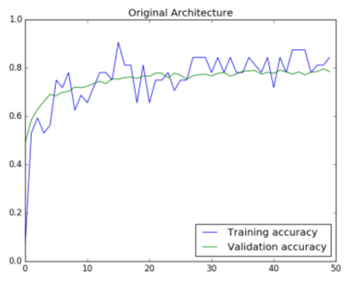


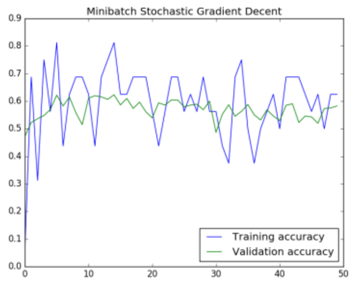
Figure 5. ROC curves

4.4. Convergence of Stochastic Gradient Descent

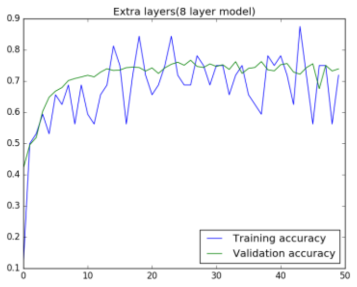
It is evident from the results that the variants of the baseline model generated by adding 2 more convolutional layers and the one with lower SGD minibatch size perform worse than the baseline model. However, it was anticipated that these scenarios would perform similar to or better than the baseline model. When we investigated the reason for this, we found that these two models did not converge within 50 epochs and would require more epochs to converge. Following plots show training and validation accuracies over the 50 SGD epochs. It can be noticed that there is a large variation of the accuracy values for each epoch implying that model has not converged [17].



(i) Convergence over 50 epochs for 6-layer model, SGD minibatch size of 32



(ii) Convergence over 50 epochs for 6-layer model, SGD minibatch size of 16

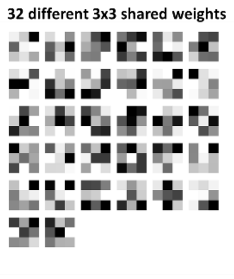


(iii) Convergence over 50 epochs for 8-layer model ,SGD minibatch size of 32

Figure 6. Convergence of variations

4.5. Layer Visualization

Shown in Fig. 6 is the visualization of the first convolutional layer in the baseline network. On the left are 32 3x3 kernels. On the right are the outputs from the 1st convolutional layer by applying each kernel to the original image; each block provides some specific feature information.



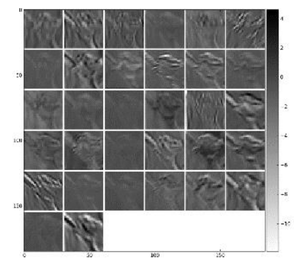


Figure 7. Visualization of first convolutional layer

5. Conclusions

In this work, we implemented convolutional neural networks (CNNs) to classify CIFAR-10 images based on their themes/categories. Firstly, our results show that for the experiments with two additional layers and batch size 16, the algorithm requires more epochs to converge and the resulting accuracy values are not a true reflection of the performance of the models and so their results are not comparable. Secondly, the model without dropout regularization tends to overfit the data, which can be justified by the higher training accuracy as compared to corresponding validation accuracy and testing precision. Thirdly, performance of the proven VGGNet architecture on the CIFAR–10 dataset can be further enhanced by replacing the last activation layer from ReLU to Sigmoid; the reason being sigmoid function preserves more output features,thus potentially provides a richer set of features for training. Finally, we found that classification of human-made objects like automobiles and airplane renders better performance in comparison to animal classes. Overall, our results show that CNN performs really well at image classification. More specifically, we demonstrated that VGGNet performs well on CIFAR-10 dataset in addition to ImageNet dataset that was implemented in the original paper.

6. Individual Contributions

Tasks	Importance	Participants*
Literature survey and learning about CNN	The task needed to implement and design the pipeline	(2)(4)
Devise architecture of CNN and its variants	Main part of the project	(1)(2)(3)
Implement Algorithms- Training the model	Main part of the project	(1)(2)(3)(4)
Testing and layer visualization	To help explain abstract concept	(3)(4)
Performance evaluation	Analyze and interpret the results	(1)(3)(4)
Project report and poster	Documentation	(1)(2)

Team members: (1) Rama Mythili Vadali (2) Jingting Yao (3) Chia-Sheng Hsu (4) Yi-Chi Shao
* All team members contributed to all the tasks, names presented here are primary contributors for each task

7. References

[1] Karpathy, Andre, (2016). CS231n Convolutional Neural Networks for Visual Recognition. Retrieved from http://cs231n.github.io/convolutional-networks/.

[2] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[3] Learning Multiple Layers of Features from Tiny Images, Alex Krizhevsky, 2009.

[4] He, Kaiming *et al.*, "Deep Residual Learning for Image Recognition." arXiv preprint arXiv:1512.03385 (2015).

[5] Szegedy, Christian. "Build a deeper understanding of images." Google Research Blog. N.p., 5 Sept. 2014.

[6] Karpathy, Andre, (2016). CS231n Convolutional Neural Networks for Visual Recognition. Retrieved from http://cs231n.github.io/convolutional-networks/.

[7] LeCun, Yann *et al.*, "Gradient-based learning applied to document recognition."*Proceedings of the IEEE* 86.11 (1998): 2278-2324.

[8] Martín Abadi *et al.*, TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[9] Chollet, François, Keras: https://github.com/fchollet/keras, 2015

[10] Bottou, Léon. "Large-scale machine learning with stochastic gradient descent."*Proceedings of COMPSTAT'2010*. Physica-Verlag HD, 2010. 177-186.

[11] Srivastava, Nitish, et al. "Dropout: A simple way to prevent neural networks from overfitting." *The Journal of Machine Learning Research* 15.1 (2014): 1929-1958.

[12] Mhaskar, Hrushikesh Narhar, and Charles A. Micchelli. "How to choose an activation function." *Advances in Neural Information Processing Systems*. 1994.

[13] Wilson, D. Randall, and Tony R. Martinez. "The general inefficiency of batch training for gradient descent learning." *Neural Networks* 16.10 (2003): 1429-1451.

[14] Lopes, Noel, and Bernardete Ribeiro. "A data pre-processing tool for neural networks (DPTNN) use in a moulding injection machine." *Proc of the Second World Manufacturing Congress, WMC'99, Durham, England*. 1999.

[15] Huang, Jin. *Performance measures of machine learning*. University of Western Ontario, 2006.

[16] Fawcett, Tom. "An introduction to ROC analysis." *Pattern recognition letters*27.8 (2006): 861-874.

[17] Bottou, Léon. "Large-scale machine learning with stochastic gradient descent."*Proceedings of COMPSTAT'2010*. Physica-Verlag HD, 2010. 177-186.