CIFAR-10 Image Classification Using CNNs
COMP 562 Spring 2022 Final Project

Mohammed Alnasser, Daqi (Jennifer) Chen, Aditya Iyer, Rebecca Rozansky[1]

Abstract

Image analysis and classification are at the forefront of machine learning research. Our daily lives already make use of advanced image processing models, whether they be for identifying road signs for an autonomous vehicle or classifying malignant tumors. CIFAR-10 is a collection of low-resolution images that fall into ten different classes. In this paper, we construct a deep learning model consisting of convolutional and dense layers to accurately classify each image.

1. Introduction

1.1. The Data - CIFAR-10

The CIFAR-10 data set contains 60,000 32x32x3 images, where each image falls into one of ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The data set was created by the Canadian Institute for Advanced Research. Each image can only belong to one class and there aren't images of objects that could potentially belong to more than one class (e.g. there are no pick-up trucks in the data-set, as a pickup trucks could be both a car or a truck).

Figure 1. 10 Categories in CIFAR-10 Image Data-set

[1]Department of Computer Science, University of North Carolina at Chapel Hill.

1.2. Implications

Since each image in the CIFAR-10 data set contains only 3072 pixels, our research can be applied to compressed images. This is especially important for surveillance data, which is subject to lossy compression algorithms in order to store several hours of footage. Since we can classify compressed images in the CIFAR-10 data set, we can generalize our findings to classify objects in compressed surveillance footage.

2. Methods

2.1. Normalizing the Data

Once the CIFAR-10 data-set is loaded, it must be normalized to a scale from 0 to 1 before being used to train the model.[1] Each pixel in the data-set falls between the values 0 and 255. To accomplish normalization, we simply divide each pixel by 255.0. After normalization, an RGB value of (0,0,0) represents a black pixel and an RGB value of (1,1,1) represents a white pixel.

We trained the model on Google Colaboratory using a free GPU hardware accelerator. Our setup offered much less computational power than required for training state-of-the-art models to classify CIFAR-10 data. Therefore, our main challenge was to build and train an accurate model under the constraint of low computational power.

We drew inspirations from a state-of-the-art approach that achieves a prediction accuracy of 96% in only 26 seconds while being trained on a single GPU.[2]

Here is a brief summary of the steps we followed:

1. We loaded the data-set using: 'from keras.data-sets import cifar10'. Keras contains a handful of data-sets including CIFAR-10.
2. One hot encode target values.
3. Implement a function that can compute the accuracy of

[1]Krizhevsky, A. (2009) *The CIFAR-10 Dataset* www.cs.toronto.edu/ kriz/cifar.html

[2]94% accuracy threshold of the DAWNBench competition in 79 seconds on a single V100 GPU.

our model on either the validation set or test set when necessary.

4. Normalize training data and testing data to fit in the range, [0,1].

5. Initialize and construct a convolutional neural network (CNN) using the Tensorflow Keras Sequential Model API: 'model = Sequential()'

6. Compile the model with the Adam optimizer and the categorical cross entropy loss.

7. Fit the model with a reasonable set of hyper-parameters such as batch size, learning rate, and number of epochs.

8. Evaluate the model on the validation set and graph the cross-entropy loss and prediction accuracy.

9. Improve upon the baseline model by repeating Steps (5)-(8) with dropout, data augmentation, and batch normalization.

10. Identify the model that achieved the highest prediction accuracy on the validation set. Train the model on the unpartitioned training set and evaluate its performance on the test set.

11. Evaluate the final model and graph the loss and accuracy.

2.2. Convolutional Neural Network (CNN)

Receptive Field: the receptive field refers to size of the region in the input that are visible to a given activation (or neuron) in a convolutional neural network.

Filters: we stacked convolutional layers with small 3×3 filters followed by a max pooling layer.

Padding: used on the convolutional layers to ensure the height and width of the output feature maps matches the inputs.

2.3. Activation Functions

ReLu:

$$Relu(z) = max(0, z)$$

Softmax: Softmax multi-class single label classification

$$\sigma(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \quad for\ i = 1, 2, \ldots, K$$

2.4. Adam Optimizer

The Adam Optimizer uses exponential weighted moving averages (also known as leaky averaging) to obtain an estimate of momentum and the second moment of the gradient. Adam optimizer utilizes the hyper-parameters $\beta_1, \beta_2, \epsilon, \eta$.

β_1, β_2 are the non-negative weighting parameters:

$$\mathbf{v}_t \leftarrow \beta_1 \mathbf{v}_{t-1} + (1 - \beta_1)\mathbf{g}_t,$$

$$\mathbf{s}_t \leftarrow \beta_2 \mathbf{s}_{t-1} + (1 - \beta_2)\mathbf{g}_t^2.$$

The state variables are normalized as:

$$\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1 - \beta_1^t} \textbf{ and } \hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1 - \beta_2^t}.$$

Like in the RMSProp optimization algorithms, Adam updates the gradient as follows:

$$\mathbf{g}_t' = \frac{\eta \hat{\mathbf{v}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}.$$

$$\mathbf{x}_t \leftarrow \mathbf{x}_{t-1} - \mathbf{g}_t'.$$

3. Building The Model

3.1. The Baseline Model

We began constructing our model with three VGG (Visual Geometry Group) blocks. The architecture of a VGG is similar to that of a regular CNN. However, a VGG block implements convolutions with 3x3 kernels and a 2x2 maximum pooling.

After training for our model for 100 epochs, it achieved an accuracy of 75.64% on the validation set. While this is much better than a model that randomly assigns images to classes, our model is still quite inaccurate. The model's unusually high accuracy on the training set (≈ 100) leads us to believe that our model is over-fit.

3.2. Regularization

To improve our model, we implemented dropout, a regularization technique that randomly selects certain neurons and prevents their activations from being used in the forward pass. We added a dropout layer after each VGG block and dense layer. We followed the standard convention of increasing the dropout probability by 0.1 after each block. After implementing dropout, our model's accuracy increased to 80.93% on the validation set, which is a 5.3% improvement from the baseline model. The model's accuracy on the training set (≈ 85) is also closer to its accuracy on the validation set, leading us to believe that our model is less over fit.

3.3. Data Augmentation

Adding dropout layers enhanced our model's performance, but it still doesn't eliminate the risk of over-fitting. To further decrease the risk of over-fitting, we can augment the original data. Data augmentation involves randomly shifting, rotating, adding noise, or applying random filters to the images. This adds more variation to the data set, forcing the model to learn more sophisticated techniques for properly classifying an image.

3.3.1. Translation

We begin augmentation by defining variables, a, b, that shift the image along the horizontal and vertical direction, respectively. This leaves a blank border around the image. We resolve this by setting the values of the border pixels to the values of their nearest non-blank pixels. This ensures that the image is slightly altered while maintain much of its original content.

$$I^t_{x,y} = I_{x+a,y+b}$$

3.3.2. Rotation

To rotate an image, an angle, θ, is randomly selected and used to generate a rotation matrix, R:

$$R = \begin{bmatrix} \cos(\theta) & \text{-sin}(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}$$

We apply R to the positions of each pixel in our image (represented by the coordinates, x, y) to calculate transform them:

$$R * \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \text{x*cos}(\theta) - y * sin(\theta) \\ \text{x*sin}(\theta) + y * cos(\theta) \end{bmatrix}$$

Since the new coordinates do not always map to integer values, we must apply bilinear interpolation to preserve the content of the image:

$$t_x = floor(x * cos(\theta) - y * sin(\theta))$$
$$t_y = floor(x * sin(\theta) + y * cos(\theta))$$
$$I^R_{x,y} = t_x * (t_y * I_{((y-t_y)+1,(x-t_x)+1)}$$
$$+(1-t_y) * I_{(i2,(x-t_x)+1))} + (1-t_x) * (t_y * I1_{((y-t_y)}$$
$$+1,(x-t_x)) + (1-t_y) * I_{((y-t_y),(x-t_x))})$$

The 32x32 resulting rotated image, I^R, preserves the integrity and content of the original image.

3.3.3. Applying More Complex Filters

Keras permits us to apply various filters that help randomize and increase the variation in our data. Any filter on an image can be represented as a shift-invariant linear operator (i.e. a matrix). Common image filters include:

1. Gaussian: used to reduce noise in an image
2. Sobel: used to locate the images of an image
3. Laplacian: used to find strong borders of an image

We planned on implementing multiple image transformations to our data, but due to the computational limitations of our Google Colaboratory setup, we were forced to use only translations and reflections. Using additional filters would have required the model to be trained over at least 200 epochs. However, with our current setup, our model could only train on the GPU accelerator for approximately 150 epochs.

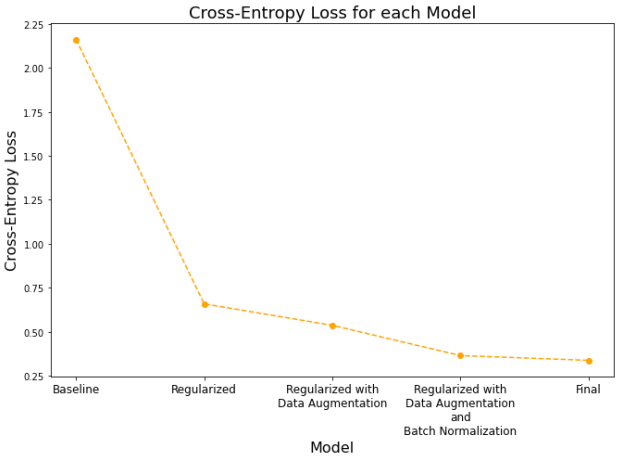
After implementing data augmentation, the prediction accuracy of our model increased to 82.99% on the validation set. This is roughly a 2% improvement upon the regularized model.

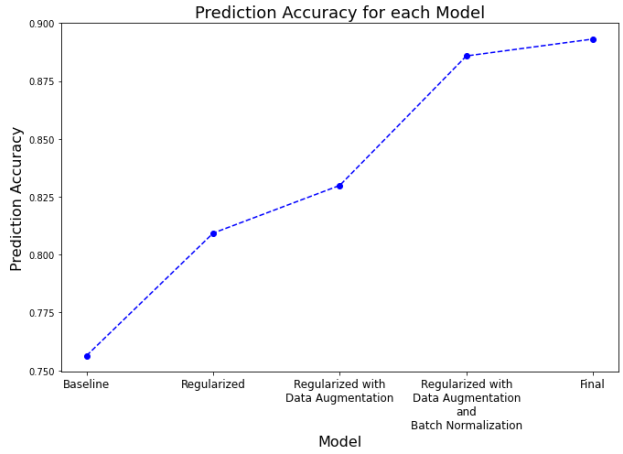
3.4. Batch Normalization

As a neural network gets deeper and more complex, it can be highly sensitive to the random initialization of weights during training. To improve upon the accuracy from the regularized model with data augmentation, we implemented a batch normalization layer after each convolutional and dense layer. Batch normalization is a method for standardizing the activation of a single layer across a batch. The model performs batch normalization a total of seven times in a single forward pass. After implementing batch normalization, our model achieved a prediction accuracy of 88.58% on the validation set. This is an improvement of approximately 5.5% upon the regularized model with data augmentation. Due to the technical limitations of our setup, we were only able train the model over 150 epochs.

4. Results

The regularized model with data augmentation and batch normalization achieved the highest prediction accuracy on the validation set.





We trained the regularized model with data augmentation and batch normalization on the complete, unpartitioned training set and evaluated its performance on the test set. The model was still trained over 150 epochs using the Adam optimizer. It achieved an accuracy of 89.31% on the test set.

While our model achieved close to a 90% prediction accuracy, we believe that, with a more computationally powerful setup, it could have achieved an accuracy well over 90%. Our free subscription of Google Colaboratory only offered as single GPU as hardware accelerator. Furthermore, we were allotted a relatively small amount of RAM and time on the hardware accelerator to train our model. This resulted in us training a relatively simple model with minimal data preprocessing over a small number of epochs.

5. Conclusion

The CIFAR-10 data set is heavily used in image classification tasks. This paper aimed to address the benefits of dropout, data augmentation, and batch normalization as techniques to avoid overfitting, especially given limited resources.

We first constructed a baseline model with three VGG blocks and built three additional models by incorporating the aforementioned techniques. Each new model we built had a higher prediction accuracy than its predecessors. The increasing prediction accuracy scores demonstrates how regularization techniques can efficiently improve model performance and prevent common pitfalls of deep neural networks.

Despite the relatively high accuracy of our final model, there is room for further research and improvement. Since image classification is a computationally demanding task, we can increase our prediction accuracy by further augmenting the CIFAR-10 data, adding more layers to our neural network, and training over a larger number of epochs. This could be made possible if we had access to more RAM and GPUs. We could also experiment with other state-of-the art techniques such as transfer learning and adding residual connections. While the final accuracy of our model may not match that of state-of-the-art models, our model demonstrates the various strategies to improve the performance of a neural network without incurring excess hardware costs.

Acknowledgements

We would like to thank Professor Jorge Silva and Teaching Assistant Kathryn Kirchoff for their instruction in COMP 562: *Introduction to Machine Learning*. We drew upon the foundational knowledge from the class to construct and train the classification models in this paper.

References

[1] Zhang, Aston and Lipton, Zachary C. and Li, Mu and Smola, Alexander J. (2021) *Dive into Deep Learning*, arXiv preprint arXiv:2106.11342.

[2] *How to Train Your ResNet 8: Bag of Tricks* (2020), Myrtle.ai.

[3] Chung, J., Gulcehre, C., Cho, K., Bengio, Y. (2014) *Empirical evaluation of gated recurrent neural networks on sequence modeling.* arXiv preprint arXiv:1412.3555.

[4] Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., ... others. (2021) *An image is worth 16x16 words: transformers for image recognition at scale.* International Conference on Learning Representations.

[5] Krizhevsky, A. (2009) *The CIFAR-10 Dataset* www.cs.toronto.edu/ kriz/cifar.html