CIFAR-10 Classification

Saadiya Allahbaksh, Ritik S Ghanshani, Anthony Goncharenko, Kevin Karnani

September 1, 2021

Abstract

We wanted to classify colored images and were curious to see how well various different deep learning models perform on the CIFAR-10 dataset. We decided to dive deep into the background research we found and any related work we came across to implement deep learning models to classify colored images into ten different categories. We wanted to see what architecture would best generalize over the CIFAR-10 dataset. Out of all the deep learning models we implemented (ANN, MLP, CNN, and GAN), we found that the Convolution Neural Network generalized the best for the CIFAR-10 dataset.

Contents

1 Introduction

Over the course of the term, we have implemented deep learning models on a very simple data set, the MNIST, which only focused on classifying binary images of digits. In order to expand on this learning, we wanted to see how well we could adapt our existing models to a new data set which is about classifying small color images which involves a higher dimensional approach to be able to train our model. For our research, we decided to design several deep learning architectures including ANN/MLPs, CNNs, and a GAN for the purpose of identifying the one that performed the best with our testing set. Along the way, we found some overfitting which turned out to be a very big issue with our learned models, so we came up with some regularization algorithms to be able to counter the overfitting we came across. This did not work so we decided to remove the dropout layer (the layer containing the regularization algorithm) from our ANN and MLP architectures.

2 Dataset

The CIFAR-10 consists of 60,000 32x32 color images for 10 different classes. There are 6,000 images per class. The dataset is divided into five training batches and one test batch (each batch has 10,000 images). Each batch file has a dictionary which contains the data, the labels, and the label names. The first 1,024 entries in the data contain the red channel values, the next 1,024 entries contain the green channel values, and the final 1,024 entries contain the blue channel values.

2.1 Class Labels

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

3 Methodology

Since we have four members, we decided to tackle the project by assigning a deep learning model to each member. Saadiya implemented the Artificial Neural Network model and Anthony worked on the Multilayer Perceptron. Ritik created a Generative Adversarial Network and Kevin implemented a Convolutional Neural Network.

4 Methods

4.1 Artificial Neural Network

An artificial neural network (ANN) attempts to simulate a network of neurons that make up the human brain so that a computer can learn things and make decisions similar to that of a human. Different layers of mathematical processing are used in artificial neural networks to understand the information that it is being fed. For the CIFAR-10 data set, we use an artificial neural network to classify images into one of ten different categories. The general architecture of an ANN is:

$$\text{Input} \rightarrow \text{Fully-Connected} \rightarrow \text{Activation} \rightarrow \text{Fully-Connected} \rightarrow \text{Activation} \rightarrow \text{Output}$$

The ANN architecture used for the CIFAR-10 dataset is:

$$\text{Input} \rightarrow \text{Fully-Connected} \rightarrow \text{Sigmoid} \rightarrow \text{Fully-Connected} \rightarrow \text{Sigmoid} \rightarrow \text{LogLoss}$$

The input layer receives the raw data which the ANN aims to learn about and standardizes the data so that one feature does not have more influence than another. Each observation is known as a node. From the input, the data is sent to a hidden layer. In the architecture above, the hidden layer is the first fully-connected layer and the first activation layer. The fully-connected layer takes input from the previous layer and uses its weighted sum to generate new outputs. It multiplies the input by a weight matrix and then adds a bias vector. An ANN learns the value of the weights and biases in the layers to best reach its goal. If we say that w_{ij} is the weight going from node i to node j, b_j is the bias, x_i is a node, and D is the total number of nodes, then

$$h_j = \sum_{i=1}^{D} = x_i w_i j + b_j$$

or if all of the weights and biases are stored as matrices, then

$$h = x^T W + b$$

The activation layer is a layer in which an element-wise function is applied to produce a new output. The function that is being applied is known as an activation function. The activation function being used in the hidden layer is the sigmoid activation function. The sigmoid function keeps values in the range of (0, +1) and is given by

$$g(z) = \frac{1}{1+e^{-z}}$$

The output of the hidden layer is sent to a fully-connected layer and the output of the fully-connected layer is sent to an objective function. For the CIFAR-10 dataset, the log loss objective function is used. This function is given by

$$J = -(y\ln(\hat{y}+\epsilon) + (1-y)\ln(1-\hat{y}+\epsilon))$$

Note: A small numeric stability constant, ϵ is added within the logs to avoid numeric instability (example: log 0 = $-\infty$).

To minimize the weights, stochastic gradient descent was implemented. Stochastic gradient descent is an iterative method use to optimize an objective function. Backpropagation is used to calculate the gradient of the loss function with respect to all of the weights in the network. Since the objective function is at the output lauer, we start the gradient there and pass the gradient back to the sigmoid layer, then the fully-connected layer and so on until the input layer is reached. The gradient of the log loss objective layer is

$$\frac{\partial J}{\partial \hat{y}} = \frac{y-\hat{y}}{\hat{y}(1-\hat{y})+\epsilon}$$

The gradient of the sigmoid activation layer is calculated as

$$\frac{\partial g}{\partial z} = g(z) \circ (1-g(z))$$

For the fully-connected layer, the gradient of its output needs to be calculated with respect to its input and the gradient of its output needs to be calculated with respect to its weights. The gradient of the fully-connected layer's output with respect to its input is calculated as

$$\frac{\partial g}{\partial h} = W^T$$

The gradient of the fully-connected layer's output with respect to its weights is calculated as

$$\frac{\partial g}{\partial W} = h^T$$

ADAM (adaptive moments) is an adaptive learning rate optimation algorithm. We used ADAM to find the individial learning rates for each parameter.

Given

- Decay rates $\rho_1 = 0.9$ and $\rho_2 = 0.999$

- Global learning rate $\eta = 0.01$
- Small constant for numeric stability $\delta = 10^{-8}$
- Accumulators $s = 0$ and $r = 0$
- Gradient g

The momentum is calculated as

$$s = \rho_1 s + (1 - \rho_1) g$$

The RMSProp term is calculated as:

$$r = \rho_2 s + (1 - \rho_2)(g \circ g)$$

Blending the above two terms gives

$$\frac{s}{\sqrt{r} + \delta}$$

Allowing each term to drop off according to its own weight over time t is given by

$$\frac{\frac{s}{1-(\rho_1)^t}}{\sqrt{\frac{r}{1-(\rho_2)^t}} + \delta}$$

The weights in the fully-connected layer are updated as

$$w_j = w_j + \eta \frac{\frac{s}{1-(\rho_1)^t}}{\sqrt{\frac{r}{1-(\rho_2)^t}} + \delta}$$

The biases in the fully connected layer are updated as

$$b_j = b_j + \eta \frac{\frac{s}{1-(\rho_1)^t}}{\sqrt{\frac{r}{1-(\rho_2)^t}} + \delta}$$

4.2 Multilayer Perceptron

The MultiLayer Perceptron (MLPs) is an extension on top of the ANNs from the previous section. Typically an MLP consists of more Hidden Layers aside from the initial Input Layer and the final Objective Function. In most cases, there are multiple Fully Connected Layers that each store the weights and biases for that layer's perceptron.

MLPs are extremely useful in classification problems because they use various techniques that would normally make classification impossible for non-linearly separable datasets. MLPs break through this restriction by using multiple complex algorithms and architectures to do proper regression. So what the MLP does after it reaches the first Hidden Layer, is that the calculated output gets pushed through an Activation function, like a ReLu or a Sigmoid. After this is calculated, the output is then pushed to the next hidden layer, by using the dot product on that output with calculated weights. After repeating the forward propagation steps a few times, the MLP will reach the Output Layer, at which point it is time to begin Backpropagation using the same method as described above in the section on ANNs.

4.3 Convolutional Neural Network

A Convolutional Neural Network (ConvNet/CNN) is an algorithm which operates by convolving over an image using randomized filters, updating through the learning process to classify images. Sometimes these filters are hand-engineered, but with enough training, CNNs have the ability to learn these filters/characteristics. The architecture of a CNN is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex, which is why the pre-processing required in a CNN is much lower as compared to other classification algorithms. [4]

Theoretically, a CNN is supposed to successfully capture the Spatial and Temporal dependencies in an image due to the use of convolutions. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better. [4]

A CNN differs slightly from other discriminative algorithms (Logistic Regression, ANN, MLP) in that it makes use of the Convolutional and Pooling Layers. The objective of the Convolutional Layer is to extract the high-level features such as edges, from the input image. Conventionally, the first ConvLayer is responsible for capturing the Low-Level features such as edges, color, gradient orientation, etc. The more layers we add, the higher the level of abstraction in terms of feature detection/abstraction. [4]

The Pooling layer, however, is responsible for reducing the spatial size of the Feature detected by the ConvLayer, making use of the dimensionality reduction to decrease computational complexity. A MaxPool will look for the brightest pixels, thus helping when we are dealing with binarized images. The converse phenomena occurs when we use a MinPool, as in, we focus on the darkest pixels. For our purposes, we decided to use a AvgPool, as this averages out the intensity of the pixel, giving us a smoothing effect over the extracted feature. After going through this process, all that is left is to flatten the final output and feed it to a regular Neural Network for classification purposes. For the sake of simplicity, the LeNet5 architecture was used:

Input $\rightarrow$ Conv $\rightarrow$ ReLU $\rightarrow$ Pool $\rightarrow$ Conv $\rightarrow$ ReLU $\rightarrow$ Pool $\rightarrow$ FCL $\rightarrow$ FCL $\rightarrow$ FCL
$\rightarrow$ Softmax $\rightarrow$ Cross Entropy [1]

4.4 Generative Adversarial Network

A Generative Adversarial Network is an algorithm that consists of 2 parts, a generator and discriminator. The generator works against the discriminator and they are trained simultaneously. The generator generates a fake image and the discriminator's job is to identify the fake images from the real images from the training set. The important thing to keep in mind when implementing the GAN is to tell the objective function that the first half of the data coming in is the real data, and the rest is the fake data. From there it will evaluate the discriminator's ability to do as its namesake, and to discriminate. Using this we will update the Fully Connected Layer's weights within the Discriminator portion of the architecture using gradient descent. Once the discriminator's weights have been updated, we will generate new fake data and pass it through the architecture, but this time without adding the real data. Evaluating only the fake data, the GAN will do gradient descent updating the weights of all Fully Connected Layers of the Generator, until it reaches the last layer to back propagate through.

However, in order to adapt this algorithm to supervised learning, a few modifications had to be made. First, the discriminator is modified and a Fully Connected layer is added at the front which feeds into the TanH activation layer. At this point, a fork occurs and this data is fed into 2 separate models: one, a normal discriminator architecture and the other is a Fully Connected layer with a Softmax activation layer, acting as a classifier [2]. This results in the classifier being trained alongside the discriminator. Usually we would discard the discriminator and use generators but in this case the discriminator and classifier are preserved. In essence what we have end up with is a Multi Layer perceptron that has been trained using the generative algorithm.

5 Evaluation

5.1 Artificial Neural Network

For multiple runs of the artificial neural network (ANN) model, the model produced accuracies that were very similar. Therefore, results from one run is included in Table 1. This table shows the accuracy for the training set and the validation set.

Dataset Type	**Accuracy**
Training Set	68.194%
Test Set	38.48%

Table 1: ANN Metrics

We got the highest accuracies with a learning rate of 0.01. The number of iterations for the gradient descent is 2,000. The model took 54 minutes and 56 seconds to train.

5.2 MultiLayer Perceptron

For the Multilayer Perceptron, a learning rate of 0.001 for 4,000 epochs gave the highest accuracy. The model took 2 hours, 14 minutes, and 44 seconds to train. The MLP model had a better accuracy for the training set compared to the ANN model. However, the ANN model had a better accuracy for the test set. We attempted to implement regularization techniques like using a Dropoff Layer and the L2 regularization technique, but our experimentation showed no favorable improvement.

Dataset Type	**Accuracy**
Training Set	76.934%
Test Set	35.96%

Table 2: MLP Metrics

5.3 Convolutional Neural Network

Initially, a big issue was that to get the CNN to run on the entire dataset using only a NumPy implementation was the sheer computational complexity. For one epoch, it initially took around 12-16 hours, which is unfeasible. To counter this, research was conducted with regards to how CNNs work under the hood in most Deep Learning APIs and libraries. It turns out most implementations take advantage of a function found in MATLAB called `im2col`, which takes an image and reconstructs each block in the image into a column vector. This is used to ease the computational complexity that occurs when performing convolutions and applying filters in the Convolutional Layer, as a naive implementation of a convolution function is $O(n^2)$. In fact, since ∼90% of both the GPU and CPU usage of the AlexNet occurs in the ConvLayer, it is sensible to see why the naive implementation of a CNN would take so long. With the implementation of this, however, it took 2-4 minutes to learn the entire training set of 50,000 images, and around 2-10 seconds to validate the learned model against the testing set of 10,000 images.

Various techniques were implemented to see what would affect the CNN's output. Xavier Initialization was added to each FCL and ConvLayer in order to help us achieve our global minima faster. Additionally, a simple gradient descent algorithm was used, then the ADAM optimization algorithm was used, and the a validation set was used in order to check against overfitting. The results using stochastic gradient descent over 20 epochs over the entire dataset are shown in the table below:

Dataset Type	Accuracy	ADAM
Training Set	68.9%	No
Training Set	74.5%	Yes
Test Set	52.3%	No
Test Set	52.6%	Yes

Table 3: CNN Metrics w/o Validation

Dataset Type	Accuracy	ADAM
Training Set	64.5%	No
Training Set	66.7%	Yes
Validation Set	65.1%	No
Validation Set	66.9%	Yes
Test Set	55.1%	No
Test Set	48.6%	Yes

Table 4: CNN Metrics w/ Validation

As shown above, adding a validation set seemed to worsen the issue of overfitting, and even reduced the accuracy on the training set itself. Batch gradient descent was attempted using a batch size of 100, which resulted in similar results across the board. While it did take slightly less time (around 5 minutes less over 20 epochs), it did have worse results by a few percent. One note to point out is that the original LeNet5 suggests the use of the `tanh` function, but due to the results being far worse than expected, a switch was made to the `ReLU`. Lastly, $\eta = 0.0001$ for the ADAM results, and $\eta = 0.005$ for the non-ADAM results.

5.4 Generative Adversarial Network

The first issue that we encountered was in adapting previously written GAN code to the SGAN architecture by adding the classification components. The resulting architecture used batch gradient descent and with a batch size of 100, the estimated time for one epoch was 30 minutes, resulting in 15 hours for 30 epochs. Having ran it once with undesirable results, training on the whole dataset was deemed unfeasible. Instead, we pivoted to training with the whole dataset in mini batches to simulate one epoch. Initially, the model had 8 percent training accuracy and 10 percent testing accuracy, which was undesirable. Experimenting with the learning rate and the number epochs yielded a 20 percent training accuracy and 13 percent testing accuracy.

6 Conclusion

6.1 Overall Analysis

The ANN and MLP models took 1-2 hours to run. We decided to combine all of the batches to push through all of the data through the architecture at one time. This increased the time the model took during training.

The CNN did worse than we thought it would in terms of accuracy, but ran much faster due to an optimization from the use of `im2col`. The use of a validation set as well as switching between batch and stochastic gradient descents seemed to have little to no effect on the yielded metrics. Overall, $\sim$60% training and $\sim$50% testing accuracies are rather undesirable, especially since it took an hour to run, but it seems to be on par with the results found with what occurs when using Deep Learning libraries. [3]

While researching semi-supervised GANs, we found that this model has a lot of potential and can be used for classifying partially labeled data. However, due to the library restrictions, scope of the project, and time constraints, building an optimized model was not possible and hence limited our approach. Given more time and optimization tools, this approach can prove to be as good as CNNs.

One last thing to point out is that the CIFAR-10 dataset is much harder to deal with than MNIST, as it has 3 color channels and the images are slightly bigger. This is clear since even with non CNN DL algorithms, we were getting 90+% accuracy. Even after conducting research, it seems that most implementations using Keras, PyTorch, or TensorFlow using the same architectures give similar results.

6.2 Future Work

The ANN and MLP models resulted in overfitting. To counter this, we added code for a dropout layer. However, when testing the models we saw that the dropout layer decreased the accuracy of the models to around 10%. Future work for these two models would include adding better regularization algorithms to counter the overfitting and figure out what other alternatives there are to deal with this issue.

In terms of the CNN, perhaps a better model could have been used. The LeNet5 is outdated, and while it works well with the MNIST dataset, this is due to MNIST images only being black and white. With respect to colored datasets like CIFAR-10 and CIFAR-100, it is not logical to use this architecture as it is far too simple.

In order to improve the performance of the Generative Adversarial Network, various approaches can be adopted. Mainly, instead of using Fully Connected layer for feature extraction, using convolution layer can yield better results. Moreover, there are other approaches to supervised GANs that can be explored. For example, separate discriminator models with shared weights and stacked discriminator models with shared weights are two different approaches that can be explored.

Along with the above improvements, optimizing the source code by using GPU processing will speed up time taken to learn by a lot and thus shorten the time taken for debugging.

References

[1] Lenet-5: Lenet-5 architecture: Introduction to lenet-5. *Analytics Vidhya*, Mar 2021. URL: https://www.analyticsvidhya.com/blog/2021/03/the-architecture-of-lenet-5/.

[2] Jason Brownlee. How to implement a semi-supervised gan (sgan) from scratch in keras. *Machine Learning Mastery*, Sep 2020. URL: https://machinelearningmastery.com/semi-supervised-generative-adversarial-network/.

[3] David Yang. Tutorial 2: 94% accuracy on cifar10 in 2 minutes. *Medium*, May 2019. URL: https://medium.com/fenwicks/tutorial-2-94-accuracy-on-cifar10-in-2-minutes-7b5aaecd9cdd.

[4] David Yang. Building a convolutional neural network: Male vs female. *Medium*, Apr 2020. URL: https://towardsdatascience.com/building-a-convolutional-neural-network-male-vs-female-50347e2fa88b.