EARIN Project Presentation

Aayush Gupta

Description of Project

In this project, a data set consisting of aerial images of ten different vehicles will be used to train a neural network to identify vehicles. A second (untested and unlabeled) data set will then be run through the neural network to see how well it classifies vehicles in pictures it has not seen yet.

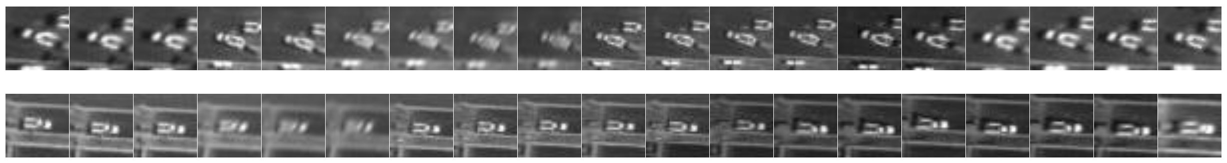
The two main difficulties that will be faced are the fact that the training set is heavily imbalanced and that, while the validation set is balanced, it is unlabeled. This will not only make the model more difficult to reliably predict, but it will also make it much more difficult to quantitatively analyze the effectiveness of the neural network's predictions of the validation set.

This project will make use of a convolution neural network along with the mini batch gradient descent method to optimize the weights and biases of each layer. After each epoch, the validation set will be run through the neural network to see how much the outputs have changed since the previous epoch. The program will stop optimizing once the validation set's outputs have converged or after a set number of epochs have passed, whichever comes first. At that point, relevant information about the final states of both the training set and validation set will be printed.

Description of Data Set

The data set consists of ten sets of black and white 32 by 32 pixel images of different vehicles. There is a significant difference in the number of images between the different sets. Half of the vehicles have fewer than 1000 images, whereas Sedans have 234,209 images. Furthermore, most of the vehicles that have fewer images have rather insignificant variations in their images. In fact, in many cases, there are only three or four different vehicles pictured in the entire set with only slight variations in the images. The fact that that many of the smaller data sets have unique images numbering in the single digits will make creating a reliable predictive model significantly more challenging.

Below are 38 images from the data set showing flatbed trucks with trailers. While the data set consists of 633 different images, all 633 images are actually slight variations of the same two vehicles.



With such tiny variations in many of the data sets, any model created will be prone to significant overfitting of the training set. The model is likely find specific patterns that very strongly correlate a set of images of some vehicle type in the training set and assume that all vehicles of that type must always have those specific patterns. As an example, if all images of busses were of the same bus parked at the same 45 degree angle, the AI model might conclude that all busses are long rectangles occupying a 45 degree angle in the image. Of course, the AI technically wouldn't be wrong—every single image of a bus it had seen up until that point satisfied that description. Even the perfect AI model is only as good as the training set that was used to develop it.

A potential solution to this problem is to artificially expand the training sets of the most poorly represented classes. This can be done by taking the existing images and altering them in one or more ways. Some common image manipulations are rotating, rescaling, changing brightness, and adding random noise. These changes not only make the gradient descent method more effective at minimization (as a result of the larger data set), but they can also reduce the chances that improper correlations are made. In this case, the goal is to ensure that predictions are based on the shape of the vehicle and not the orientation, background, or size (a picture can be taken close or far away).

General Algorithms to be Used

The neural network will consist of an input layer (a matrix of the pixel values of one image), some number of hidden layers (of some width), and an output layer (the neural network's prediction). Apart from the input layer, each layer will have a series of weights and biases as well as an activation function, which will all contribute to the values of the final output. Each layer (apart from the input) performs a convolution and, optionally, max pooling. The output layer will also have a cost function that will be used to quantify how well the current weights and biases fit the data.

Each hidden layer will have one or more nodes, a filter size (for the convolution), a stride length, an activation function, and possibly max pooling (which will have its own pool size and stride length). Each node will have a bias and set of weights arranged in an n by n matrix, where n is the filter size. Initially, the weights and biases are randomized from -10 to 10.

Generally, the raw output of each hidden node, which is equal to the inputs multiplied by the node weights with the bias added at the end, will be passed through an activation function, which modifies, filters, and/or normalizes the output. Some activation functions that will be tested are ReLU, the sigmoid function, and softmax. The former two will generally be used in hidden layers. Softmax, which normalizes the outputs so that their values are equal to the probabilities of them being the correct prediction for the given input, will be used for the output layer.

When a convolution is performed, the n by n matrix (the one with the weights) travels along the input matrix performing a dot product with the values in the input matrix that it overlaps with. After the dot product is performed, the bias is added, and the number is passed into the activation function. The result of the activation function is saved in a new matrix. The weights matrix moves along the input matrix in steps according to the stride length until it has traversed the entire matrix. If there are multiple input matrices, dot products are performed for each matrix with its associated weights matrix, and the results of all the dot products are summed.

Afterwards, if max pooling is present, a second matrix (with dimensions equal to the pooling size) traverses along the new matrix in steps according to the stride length. For every step that the pooling matrix takes, the highest value among the values in the overlapping area is taken and placed into the output (for that particular node) matrix.

Both the convolution and max pooling reduce the matrix size. In the case of convolution, patterns within the input image (represented by the weights) are being searched for, and the convolution returns the areas where those patterns can be found. This, more or less, focuses the matrix on points of interest (as opposed to the whole image), reducing its size. The max pooling reduces the resolution of the matrix, while preserving the most relevant pieces of information. This should reduce the complexity (and computation time) of subsequent calculations without significantly affecting the final outcome.

The convolution on the output layer is always performed with a filter size equal to the dimensions of the incoming matrices. As a result, the output of the convolution for each output node is a single value (which is a 1 by 1 matrix).

Since the training set is labeled, the true identities of the outputted values are known and can be compared against the predicted values. The output layer will have a cost function associated with it that will be used to numerically determine the effectiveness of the current weights and biases values at correctly predicting subsequent data points. The two cost functions that will be tested will be the mean squared error and cross entropy.

The mini batch gradient descent method will be used to optimize the values of the weights and biases. This optimization will be achieved by finding the values of weights and biases that minimize the cost function. A key feature of the mini batch gradient descent method is that it does not wait for the entire data set to be passed through the network between optimization steps. Instead, it reoptimizes after just a few data points (relative to the overall data set) have passed through. While this does increase the noise of the optimization, it greatly reduces the computation time and makes it less likely that the optimization will get stuck in a local minimum.

Some additional steps will be taken to further speed up the optimization. In order to avoid having to recalculate the weights and biases for every data point, the back propagation algorithm will be applied at the start of every new batch. Additionally, the gradient descent method will be enhanced by adding a momentum hyperparameter, which will allow it to be influenced by previous gradients (and not just the current gradient). By having a momentum, the gradient descent method should be less prone to being slowed down by large oscillations and sudden flat regions (which have a small gradient).

Since the training set is heavily imbalanced, even a batch size of 100 will never have at least one member of every class, assuming every batch contains an equal proportion of each class relative to the total class samples. This means that each mini batch will optimize only some of the classes, potentially resulting in a huge discrepancy in the qualities of the fit. Furthermore, even when the less numerous classes are represented, individually they only represent, at most, 7 percent of the mini batch. If the cost function is consistently low for the class with the most samples, but very high for some of the other classes, the overall cost will remain relatively low, which will make it appear to the neural network as if it's doing well for all the classes. As a result, it may not update its weights and biases correctly to improve the fit for the poorly performing classes.

The first solution will involve artificially increasing the representation of classes with fewer samples within each batch. While this will result in some data sets being cycled through faster than others, it will ensure that each class contributes to every optimization step. However, even with increased representation, the classes with smaller data sets will still be greatly outnumbered by those that have larger data sets.

The second solution will address this issue by applying weights to the costs of the less represented classes so that, for each batch, the impact of each class's costs on the total cost is equal to 10 percent. By doing so, whenever a class performs poorly, regardless of the number of samples it has, the neural network will much more readily adjust its weights and biases to fix the issue.

General Plan of Tests/Experiments

Preliminary testing will involve optimizing the gradient descent and the neural network to best fit the training data set. A number of different parameters will be analyzed to determine which values result in the lowest cost while maintaining a reasonable computation time.

Once a favorable set of parameters is found, the training set will temporarily be split to create a test validation set. The test validation set will have the same number of samples and distribution as the real validation set, and its samples will be chosen randomly. The neural network will then use the new training set to try and correctly predict the samples in the test validation set (which were removed from the training set).

After the neural network has made its predictions, they will be analyzed using the confusion matrix, which shows the number of true positives, true negatives, false positives, and false negatives for every class. Two matrices will be created and populated. The first will be filled in with the neural network's best guesses for each sample, while the second will be filled in using the probabilities that the neural network estimated for each sample. The most relevant pieces of information that the two confusion matrices will show will be the number of mistakes the neural network made and the number of mistakes it thinks it made.

If the neural network is found to have made too many errors on the test set, the first step will be repeated until it improves. Otherwise, the second step will be repeated, this time with a completely new test validation set (again, randomly selected). The same analysis will be performed to see how consistent the results are with different data sets.

Once the behavior of the neural network is reasonably well understood, the training data set will be returned to its original state, and the true validation set will be tested. Two sets of values will be recorded for the validation set's predictions. The first will be the total number of predictions made of each class, and the second will be a confusion matrix filled in with the probabilities that the neural network estimated for each sample. Since the validation set is unlabeled, a confusion matrix comparing the predicted values with the actual values will not be possible to create.

First, the total number of predictions made for each class will be compared with the actual number of samples of each class (77), and a total offset from all classes (total difference between the predicted and actual number of samples from each class) will be calculated. This number will be compared against a table showing the average total offset per number of errors to obtain a rough estimate of the accuracy of the predictions.

Then, using what was learned about the two confusion matrices from the test validation set, the true validation set's (sum of probabilities) confusion matrix will be analyzed. The goal will be to try and find any patterns or similarities that indicate obvious false positives, false negatives, true positives, or true negatives. If enough identifiable errors are found, a different set of neural network parameters can be tested to see if the predictions improve.

The third method of analysis of the unlabeled validation set will be through the use of tdistributed stochastic neighbor embedding on the classification layer. This will enable the classification layer to be visualized in two dimensions, which can be used to show the general locations of the validation set samples relative to the training data set. The locations of the validation set samples relative to their prediction groups can potentially reveal useful information or even trends about correct and incorrect predictions. As an example, samples that are deep inside of their predicted group are much more likely to be true positives than those near the edges.

Statistically, the lower the predictions' total offset from all classes, the lower the overall error of the predictions. As such, a reliable goal will be to get that number as low as possible without worsening the fit of the training set. and, ideally, without worsening the error of a subsequent test validation set.

Methods of Result Visualization

A series of graphs and tables will be used to show the results obtained. While optimizing the gradient descent and neural network parameters, the plots of the loss value, accuracy on train set, and accuracy on validation set (test set only) will be printed. Additionally, the total computation time will also be shown. These will be used to quickly analyze the effectiveness of every set of parameters tested.

Additionally, the TSNE method will be used to visualize the different samples as they are grouped in the classification layer. This method flattens a higher dimension layer into two dimensions while preserving the grouping and separation of groups of the original layer.

When the test validation sets are run, two confusion matrices will be displayed for each set. The first will contain the predicted values, and the second will contain the sum of all probabilities estimated by the neural network. In both cases, relevant errors (for each class and a grand total) will be calculated and displayed.

For the true validation set, only one confusion matrix will be printed (sum of probabilities). The relevant errors (for each class and a grand total) will be calculated and displayed. Additionally, the total number of predictions of each class will also be printed, as well as the total class offsets (total difference between the predicted and actual number of samples from each class).

Finally, a simulation will be run, where elements in a set of arrays representing correct and incorrect predictions will be moved randomly one at a time from correct to incorrect positions. The simulation will be run at least 1000 times simultaneously and calculate the average total class offsets (total difference between the predicted and actual number of samples from each class) for each number of incorrect predictions made. The standard deviation for each value will also be found.

Description of Implementation

The model that was implemented is a neural network which consists of two convolution layers (both having kernel sizes of 5 by 5) with 2 by 2 max pooling followed by three additional hidden linear layers, with the final hidden linear layer being connected to the output layer. Apart from the final hidden linear layer, each hidden layer (including the convolution layers) outputs through a ReLU activation function. The loss function used for optimization of the weights and biases is the cross entropy loss function. Because the cross entropy loss function already contains a softmax component and applying softmax twice can result in unexpected behavior from the neural network, the softmax activation function was not applied to the output of the final hidden layer.

The images in the dataset are first split into a training set and validation set. The split can be fractional (where a fraction of every class's images are split into the validation set) or with a constant value (where the same number of images from each class are used). In order to introduce greater variety in the training set and to encourage the NN to generalize and not memorize, the images in the training set are then augmented using transformations. These transformations include random flips, rotations, translations, scale changes, and brightness changes. Since these are overhead images of vehicles, all of these transformations make sense, and the resulting images could realistically be seen in a real world data set.

A second validation set was created by augmenting the images in the original dataset. As a result of these random augmentations, the second validation set is more difficult to correctly predict and much more closely resembles a real validation set, which would normally have images that are completely different from those in the training set.

A mini batch size of 256 was eventually chosen since that is large enough to take advantage of GPU optimizations, but not so large that overfitting of data is risked. The neural network is trained (modifying its weights and biases after every batch) until a number of images equal to the total size of the training set have been run through it. Afterwards, the performance of the neural network is checked by having it predict the images in the validation set (which it has not seen).

Since the training dataset sizes can differ greatly, the metric used to determine the stopping point for the NN is the total number of images trained on. This ensures that analyses between different tests can remain relevant.

Results and Analysis

An issue that came up almost immediately after running the program for the first time was the amount of time needed to execute each test. This was remedied (at least partially) by allowing training of the NN to be performed on the GPU instead of the CPU. Doing so reduced the computation time by ~70%.

Running the raw dataset through the neural network without any up sampling quickly resulted in the neural network classifying every image as that of a sedan. Two methods were found and implemented to remedy this problem. The first involves creating duplicate images from the smaller data sets until all data sets are equal in size. In order to reduce overfitting by the NN, the images need to be augmented with random transformations. This method solved the main problem of the unbalanced dataset, however the resulting augmented dataset ended up being massive.

In order to shorten the time between epochs, a second method was applied—weighted random sampling. This method involves choosing each image in the dataset randomly as opposed to sequentially with a stronger bias towards images belonging to underrepresented classes. This means that, over the course of an epoch, some images may be chosen several times, whereas others may not be chosen at all.

Both methods were applied in our implementation (the former through the CustomDataset class and the latter through a library). The dataset_expansion_multiplier variable is used to determine how many duplicate images of each class should be made. If it is zero, no duplicate images are made. Otherwise, augmented duplicate images are created for each class until the number of images belonging to that class is equal to dataset_expansion_multiplier multiplied by the number of images in the largest class.

Both methods were found to work quite well (either combined or on their own), however, based on the results, it appears that the weighted random sampler alone may perform slightly better.

To confirm that the augmentations and duplicate images were being created correctly (and that images in the validation set didn't somehow end up in the training set), functionality was added to enable saving the created training and validation data sets to a folder.

After performing a number of tests, it was found that higher learning rates resulted in quicker gains in accuracy, but that they also plateaued much earlier than lower learning rates. As such, a decaying learning rate was applied to take advantage of the fast gains of high learning rates early on and the steady gains of lower learning rates later.

Additionally, it was found that regardless of which method was used to overcome the unbalanced dataset problem, the NN overfit the smaller classes much more quickly than the larger classes. This actually makes sense considering that over every epoch, the NN uses images in the smaller classes many more times than images in the larger classes (due to the weighted random sampler). The weighted random sampler doesn't take this into account and, as a result, oversamples the smaller classes. To overcome this, the larger classes needed to be up-sampled, ideally to some point of equilibrium.

Initially, constant values were multiplied by the weights of the larger classes before they were passed on to the weighted random sampler. This worked, quite well even, but wasn't particularly elegant. So, we created a dynamic sampler that recalculates the weights of each class based on their performance in the current epoch (using f1 scores). Due to the possibility of undesired oscillations and possibly even divergence, a dampening factor was added, which makes the previous epoch's f1 scores partially contribute to the dynamic weights.

We found that a quadratic relationship has the potential for better results than a linear relationship if the dampening factor is low enough. However, if the dampening factor is too low, the model can start oscillating uncontrollably. We are still trying to fine tune these values as one of the last tests.

Initially, we used a balanced validation set of 100 images from each class. The results we were getting after applying a few optimizations were quite good, approaching 99% accuracy in some cases. However, we realized that this might not be the best model of the real world, where there are in fact many times more sedans than there are pickup trucks and buses. As such, we switched to an unbalanced validation set for the final analysis, where precision is much more sensitive to changes in the model. Our current goal is to maximize the macro average f1 score, which is a combined measure of precision and recall values of all classes where every class contributes equally.