COMP 551: Assignment 3 - Image Classification

Nikesh Muthukrishnan (nikesh.muthukrishnan@mail.mcgill.ca, 260483392)
Huey Francois Nguyen (francois.nguyen@mail.mcgill.ca, 260395728)
Zachary Warnes (zachary.warnes@mail.mcgill.ca, 260581958)
CNN_FakeNews

March 17, 2017

1 Introduction

The objective of this analysis is to classify images from a modified 40-class subset of ImageNet, using logistic regression as a baseline learner, a fully-connected feed forward neural network and SVM SIFT/SURF as well as a convoluted neural network.

2 Problem Representation

We are given training data and their respective class output for 26,344 instances. Our task is to classify 6,600 instances of test data into 40 classes. In both training and test data, each instance is represented by an array of size 64 pixels by 64 pixels, where each of its elements is an array of 3 colour (RGB) channels, ranging from 0 to 255.

The distribution of the data is as shown in Figure 1. Approximately, 60% of the data is classified as Class $0, 1, 2, 3, 4$. Furthermore, the test data follows a similar distribution.

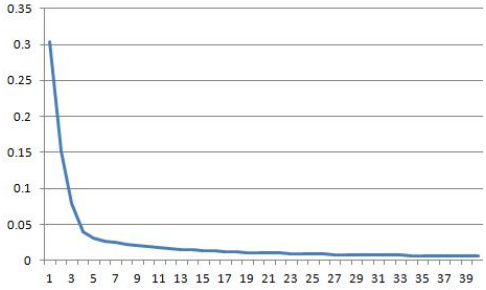

Figure 1: Distribution of Training Data

For the baseline learner as well as the convolution neural network, no filtering was performed on the images. However, for the neural network implementation, the images were converted to grayscale. The class output was converted into a 40-column binary matrix.

3 Training

3.1 Logistic Regression (classification)

3.1.1 Problem Setting

Logistic regression is a binary classification linear model. It model log-odds ratio of a binary outcome. It is a discriminative model. $\boldsymbol{x}$ is a vector of features, in this case, normalized RGB colour continuous variables, y is a binary variable that represents the class. The model estimates $P(y|\boldsymbol{x})$ using the formula:

$$P(y = 1|\boldsymbol{x}, \boldsymbol{\beta}) = \sigma(\boldsymbol{x} \cdot \boldsymbol{\beta}) = \frac{1}{1 + e^{-\boldsymbol{x} \cdot \boldsymbol{\beta}}}$$

In this analysis, we use Scikit-Learn's sklearn.linear_model to train, fit and predict the accuracy of the model.

3.1.2 Model and Hyper-Parameter Optimization

Optimization of the model and the hyper-parameters is done by the model behind the scene.

3.2 Neural Networks classification

3.2.1 Problem Setting

Neural Network is a machine learning model, that performs both linear operations and non-linear activations on multiple hidden layers composed of hidden nodes, to calculate the output for an input. Using a loss function, an error is calculated and back-propagated using Gradient Descent, to update weights. This is performed continuously over multiple epochs until the minimum loss is found.

For the tiny imagenet classification problem, the sigmoid function was selected as the activation function and the sum squared error function was selected as the loss function. These functions were selected for ease of application.

$$\sigma(w \cdot x) = \frac{1}{1 + e^{-w \cdot x}}$$
$$J(w) = \frac{1}{2}(y - h_w(x))^2$$

Finally, the tiny imagenet problem is composed of a large dimension training set. There are 26344 images, each with the size $3x64x64$. Since the input to the Neural Network needs to be flattened, the input dimension for each input becomes $12288x1$. Using an input this large was computationally expensive, therefore each image was transformed into grayscale using the following formula (NTSC standard luminance formula):

$$Gray = 0.3 * Red + 0.587 * Green + 0.114 * Blue$$

Although many important features will be lost, the neural network will be able to iterate through the data and results can be obtained. The reduced dimension size becomes $4096x1$.

3.2.2 Model Optimization

Neural Networks have several hyper-parameters that can be configured to optimize the performance of the algorithm. These parameters include the following:

- Learning Rate (α)
- Momentum Rate (β)
- Hidden Layers
- Hidden Nodes
- Epoch Iterations

3.2.3 Hyper-Parameter Optimization

Finding the optimal hyper-parameters on a self-implemented Neural Network can be time expensive, therefore, the scikit-learn implementation of NN was used to find the optimal values of the hyper-parameters and are described in Tables 1 to 4. The results shown indicate the accuracy of the corresponding NN on a validation set of size 500. Note that the grayscale images were used to obtain these results, therefore they are expected to perform poorly. Furthermore, momentum is a parameter that can speed up the Gradient Descent in a NN, however it is not necessary, and was therefore omitted from the results below. Finally, using more than one hidden layer with various hidden nodes, resulted in poorer results than a NN with 0 and 1 hidden layer, and was thus omitted from the results below as well.

		Learning Rates		
		0.001	0.01	0.1
Epochs	3	0.306	0.306	0.306
	5	0.306	0.312	0.328
	10	0.306	0.306	0.321

Table 1: No Hidden Layers

		Learning Rates		
		0.001	0.01	0.1
Epochs	3	0.13	0.306	0.306
	5	0.122	0.306	0.306
	10	0.16	0.306	0.306

Table 2: One Hidden Layer with 10

		Learning Rates		
		0.001	0.01	0.1
Epochs	3	0.314	0.306	0.306
	5	0.294	0.306	0.306
	10	0.33	0.312	0.306

Table 3: One Hidden Layer with 50

		Learning Rates		
		0.001	0.01	0.1
Epochs	3	0.306	0.306	0.306
	5	0.306	0.306	0.306
	10	0.308	0.306	0.321

Table 4: One Hidden Layer with 400

The hyper-parameters obtained from the highest validation scores from the results above, were the following:

- No hidden layer, learning rate = 0.1 and 5 epochs: Validation Accuracy of 0.328
- One hidden layer with 50 nodes, learning rate = 0.001 and 10 epochs: Validation Accuracy of 0.33

For a simpler model, the first set of hyper-parameters were selected for the NN implementation.

3.3 Support Vector Machine

3.3.1 Feature Extraction

Image data, when used in a primitive implementation has an incredible high dimensionality where each pixel is itself a feature. Performing any simple classifier will cause any particular pixel to have and extremely minute effect on the overall classification. As an added problem the sheer size of the dataset with this basic implementation causes the computation power required to fit each parameter to be vast. Initially, optimization by converting images to grayscale was the obvious first choice since this reduces the dimensionality of every image 3 fold. This may further reduce the impact and single pixel has on the overall classification by further simplifying the image. The combination of computational optimization and desire for meaningful features despite simplifications drew us to use Speeded-Up Robust Features (SURF) in our SVM implementation. This library selects those areas of each image which have

the most importance to the overall image using a hessian matrix. The result of using SURF is that each image is represented as only the key points within it. Once these key points are identified they are then orientated, so that similar key points within the same class of images do not vary by rotation. SURF also has the added bonus of reducing the dimensions of each image to further optimize computation. Once process using SURF each image becomes a collection of blobs which signify the important features of each image and can then be used in processing.

3.3.2 Implementation

The SVM was implemented using sklearn's SVM library and a one vs all classifier, which allows for simple implementations and the flexibility to try various combinations of model specifics. Again, considering the problems high dimensionality the kernel of choice was the Gaussian kernel chosen for its ability to separate this kind of high dimensional data. Other kernels were attempted but they offered little in terms of increased performance the results of which can be seen in the following table. For the original primitive grayscale implementation, three different kinds of transformations were used all with various benefits. First, a simple averaging of each of the RGB colors to produce a gray and increase performance. The second method was to select the midpoint between the highest and lowest value color of that cell in an attempt to replicate the intensity of any particular pixel. Lastly, a combination of RGB colors[3] which acts to represent those colors which our eyes are most accustomed to seeing referred to as 'luminosity'. Results varied slightly with luminosity coming out slightly ahead.

4 Convoluted Neural Network

4.1 Implementation

The RGB values of the training and test data were first converted from data type 'integer' to 'float32' to allow normalization via division by 255. This brought each value from the domain [0, 255] to [0, 1].

Then, using Keras on top of Tensorflow, we add to the model two convolution layers, each with 64 filters with receptive fields of size 3 by 3, then two hidden layers with, each with 32 filters. We had tried 128 filters; however, due to the estimated total time based on time taken to train the first epoch, we decided to reduce it to 64 filters. The layers were activated with rectified linear unit (ReLU), because under ReLU, the gradient is constant and so as x increases, it will not vanish like it would under the sigmoid activation. This allows for faster learning.

After adding the convolution layers, we add a pooling layer. This generalizes the feature maps generated from the previous convolution layers. We set the receptive fields of size 2 by 2.

To reduce overfitting, we use the dropout technique, set at 20%, after each convolution and hidden layer, but we could have set the rate up to 50% (Srivastava, 2014).

Finally, the output layer was activated with softmax.

To compile the model, we set the learning rate at 0.01 and the number of epochs at 10 and we use stochastic gradient descent over the loss function defined by categorical cross-entropy.

The training accuracy for the model at 2 different settings, at epoch 3, 5 and 10, are shown below:

		Learning Rate
		0.01
Epochs	3	0.3228
	5	0.3602
	10	0.4228

Table 5: Training accuracy, double convolutional layers with receptive fields size 32 by 32

		Learning Rate
		0.01
Epochs	3	0.3250
	5	0.3635
	10	0.4320

Table 6: Training accuracy, double convolutional layers with receptive fields size 64 by 64

5 Results

5.1 Logistic Regression Results

The implemented default logistic model with no special settings gives a training accuracy 29.202%. The implementation on the test data yields a 29.818% accuracy.

5.2 Neural Network Results

The implemented Neural Network had the following specifications:

- Learning Rate = 0.1
- Number of Epochs = 5
- No Hidden Layers

Note: These optimal hyper-parameters were derived for grayscale images as discussed in Section 3.2.3.

Using this implementation on the provided test data resulted in a 0.30030% accuracy. Since approximately 30% of the training and test data was classified as class 0, the NN predicted class 0 for all test samples.

5.3 Support Vector Machine

For each run, the one-vs-all classification was used with a K- fold cross validation of 5 as a standard split. The highest accuracy observed for any one trail was 32%. The one vs all implementation had a strong negative effect on the overall accuracy, the high number of distinct classes caused most models to predict the first class which makes up roughly 30% of the training data. Since each run of the SVM resulted in poor accuracy the focus was shifted to how each feature extraction technique compared to one another. In addition, multiple parameters for the SVM were modified to inspect their effect to the overall outcome of the classifier. Below we can see the comparisons of the different kernels, different grayscales and use the SURF feature extractor.

Results in the following table are all in comparison to a SVM with a polynomial kernel with degree 3 and average grayscaling. Positive percentages indicate an accuracy increase of that percent.

		Gray Scale Compression		
		Avg.	Lum.	Light.
Kernels	Cubic Poly.	0.0	6.00%	4.22%
	Gaussian	19.48%	23.80%	20.73%

Table 7: Without SURF

Now the comparison of same combinations of parameters but with SURF feature extraction applied. Percentages show the comparison to the previous table.

		Gray Scale Compression		
		Avg.	Lum.	Light.
Kernels	Cubic Poly.	27.14%	13.19%	20.82%
	Gaussian	6.11%	3.90%	1.13%

Table 8: Using SURF feature Extraction

The significant increase in performance for the cubic polynomial kernels when using feature extraction should be seen more as an improvement to a previously abysmal performance. Originally, somewhat better than random guessing after applying SURF some structure of the data can start to be utilized.

5.4 Convolutional Neural Network

After running for 10 epochs, using this implementation of the convolutional neural networks yields an accuracy of 43.909% and 43.485% for the model with double convolutional layers with 32 filters and 64 filters, respectively. The difference between the accuracy on the test of the two models is 0.424%.

Given that about 30% of the training data belongs to one dominant class, these two models perform most likely correctly classified the images of that dominant class and perform only marginally better than the logistic model on the rest of the images.

6 Discussion

6.1 Logistic Regression

For the logistic regression, the baseline learner gave a fair accuracy on Kaggle, if we assume that it accurately predicts all 40 classes correctly around 30% of the time. However, that percentage figure is very close to the percentage of the dominant class in the training data. In light of the poor performance of our other models, we suspect that this model is very good at predicting only one class: the one that was over-represented in the training data.

6.2 Neural Network

As in Logistic Regression, the Neural Network prediction is respectable in comparison to a random predictor. The model, however, simply classifies all outputs as zero, which is not be favorable. One significant improvement that can be made is to use colour information of the images, as well as a higher model complexity. Finally, since we are aware of the distribution of the data, we can change the error function such that lower frequency classes will be penalized more for incorrect classification.

6.3 Support Vector Machines

Ultimately the SVM despite using a rich feature selection performed much poorer than the neural networks. Something to consider is the larger number of classes present, which would have led to skewed positive and negative classifications for any one class. Also, considering the number of classes, any one class did not have a particularly large set to train on. Given more data and a tremendous increase in computation power, one should expect to see a rise in accuracy. Considering that some classes were similar (low pixel count with similar round shapes) extracted features may be very similar in both the positive and negative classes of those cases, leading to a reduced average accuracy overall. Although a relatively robust classifier the SVM struggles to accurately represent so many diverse classes, and understandably the more complex models performed better overall. We can see here the limitations of this sort of model in the realm of more complex scenarios despite the attempt to provide the features in a manner of different ways.

6.4 Convolutional Neural Network

For the convolutional neural network, we ran to completion the code with initially a double convolutional

layers, each with 32 filters with receptive fields of size 3 by 3, and then we repeated the same code, this time instead with a double convolutional layers, each with 64 filters with receptive fields of size 3 by 3. Each was run only once. The result was that on Kaggle, the model with the double convolutional layers each with 32 filters slightly outperformed by 0.5% the model with the double convolutional layers with 64 filters.

We have a few hypotheses: it can be that the models did not train through enough epochs to really magnify the difference between 32 and 64 filters, as they only both ran through 10 epochs. As a comparison, AlexNet ran through 90 epochs (Krizhevsky, 2012). Similarly, along the same line of reasoning, if the learning rate was too small, then there was little chance of escaping the local optima. It is also very likely that the over-representation of certain output classes, for example up to 30% of the training data for one class, and thus, the under-representation of other output classes, skewered the model towards predicting the dominant class in the test data. Or, it can be that there is no significant difference between 32 and 64 filters and that the tiny difference in performance can be attributed to the stochastic gradient descent.

7 Statement of Contributions

Nikesh Muthukrishnan worked the code for the fully implemented Neural Network and the Scikit-Learn implementation:
Huey Francois Nguyen worked on the code for the baseline learner using logistic regression with Scikit-Learn and for the convolution neural network with Keras on Tensorflow.:
Zachary Warnes completed the code for the Support Vector Machines using Surf feature extractor.:

We hereby state that all the work presented in this report is that of the authors.

8 References

1. Srivastava, N. *et al.* (2014) Dropout: A Simple Way to Prevent Neural Networks from Overfitting. http://jmlr.org/papers/v15/srivastava14a.html

2. Krizhevsky, A. *et al.* (2012) ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems 25 (NIPS 2012). https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks

3. John D. Cook. Algorithms to convert color to grayscale. https://www.johndcook.com/blog/2009/08/24/algorithms-convert-color-grayscale/

4. Feature extraction using opencv library and SURF. http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_surf_intro/py_surf_intro.ht