Kaggle Competition - Classification of Hand-Drawn Pictures

Mallard Team

Mohammed Shalaby - mohammed.shalaby@mail.mcgill.ca - 260607172
Samuel Hatin - samuel.hatin@mail.mcgill.ca - 260688157
Yan Yi Feng - yi.f.yan@mail.mcgill.ca - 260687858

I. INTRODUCTION

Image classification is an area of considerable importance in the field of machine learning, and has been widely studied due to its relevance in various applications. With the recent improvements in processing power, various researchers have been tackling the problem of computer image analysis for various reasons, including computer vision and natural language processing applications.

The purpose of this project is to design a machine learning algorithm that can automatically identify hand drawn images as well as reason about their appearance, thus allowing automatic classification of the hand drawn images into one of the 31 classes. The hand drawn images are an extract from Googles Quick Draw Dataset. The different images were created by randomly locating a drawing within a larger 100x100 pixel blank canvas image, where additional randomly generated noise was added within the image. Six examples of these final hand drawn images with added noise can be seen in Fig. 1. Of the 31 classes, one class is called empty and consists of only noise and no human drawings. To train the model, a dataset consisting of 10000 grayscale images of size (100,100) for the training set were provided, and the trained model will then be evaluated based on accuracy on 10000 similar instances on a Kaggle-based competition.

In this report, we begin with discussing the methodology for approaching the classification task; in particular, the preprocessing steps and machine learning models used are discussed. Then we present the results of the different models, followed by a detailed discussion of the justifications for such results. As expected, we will see that the results of the baseline classifiers were low as compared to the neural networks, while the convoluted neural network outperforms the other models.

II. METHODOLOGY

To be able to successfully train a model that can achieve the required task as accurately as possible, the training task was split into three interconnected stages; feature design, algorithms and hyper-parameter tuning. The main purpose of the feature design is to denoise the images and reduce the number of features for more efficient training. In the algorithms section, we further discuss the four algorithms used; logistic regression and linear support vector machine as our baselines, as well as a fully connected feed-forward neural network and a convoluted neural network. Lastly, the hyper-parameter tuning process is discussed in further detail.

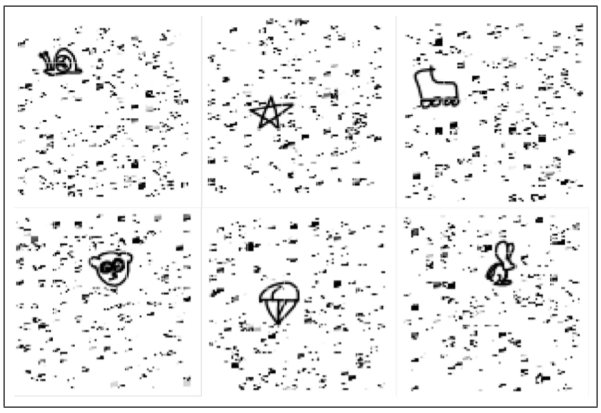

Fig. 1. Sample of hand drawn images to be classified.

During the implementation of the different algorithms, several open source libraries were used. Namely, NumPy was used for efficient linear algebra operations, and more specialized libraries were used such as scikit-learn and PyTorch for the baselines and the convoluted neural network respectively.

A. Feature Design

While it is certainly possible to perform classification over noisy bitmap thumbnails–albeit the added computational complexity and non-convergence risks–, a preferable starting point would be to preprocess the images such that a least amount of artifacts remain. To that end, the preferable approach would be to train a CNN-based auto-encoder that performs transformation operations on the input data to achieve clear outputs. That said, in the supervised learning context, it would require provided instances exemplifying what noise reduction entails, something that would either mean offsetting from Google's original *Quick, Draw!* dataset, or manual editing. Rather, a hand-tailored procedural pipeline consisting of four steps is utilized for simplicity and speed.

In the aforementioned pipeline, the first step consists of removing pixels whose intensity values fall below a certain threshold, and clipping their upper bounds to 255.

This produces a matrix containing the value zero, along with the range of values between the lower threshold and 255, effectively setting half of the initially nonzero cells to zero. Subsequently, in concurrent fashion, individually valued pixels are selected at random, and eliminated if their bounding areas defined as small squares centred over the formers constitute mostly of zero-valued cells (i.e., elimination of segregated pixels). Then, for the remaining cells whose values have not been nulled, their bounding areas are iteratively expanded starting from a one-by-one square until such action no longer yields any new valued pixels (i.e., region discovery). Such regions are removed if their areas fall below a certain limit, and finally, the largest bounding box region is returned as the hopefully clear image of interest.

Although simplistic, with cautiously chosen parameter values such as the area limits and removal thresholds, this algorithm performs spectacularly well. Measured against one hundred randomly sampled data points, the image size is reduced by on average 90 per cent, with 98 per cent of the population retaining the segment of interest (refer to *Appendix, Figure 1*). Notwithstanding the afore, this procedure does not take into consideration directly adjacent noise, and would as such leave over small, non-significant fragments, which can potentially be removed by training an auto-encoder using a selection of perfect outputs. To reduce the computational complexity of the various training algorithms, the images are further preprocessed via a bilinear aspect fit resize function to produce 55-by-55 and 28-by-28 normalized replicas, eliminating the number of features by 69.75 per cent and 92.16 per cent respectively. Additionally, to augment the amount of training data, random transform operations have been applied to generate new observations, including horizontal flipping, nonlinear scaling, off-axis sheering, and zooming amongst others.

B. Algorithms

1) Logistic Regression: Logistic regression is a probabilistic discriminative model commonly used for classification tasks. To negate the need for learning the class conditional distributions $p(\mathbf{x}|\mathrm{C_k})$, the class priors $p(\mathrm{C_k})$ and the need for implementing the Bayes' theorem, discriminative models rather use the functional form of the generalized linear model, which is defined through the conditional distribution $p(\mathrm{C_k}|\mathbf{x})$. The parameters ($\mathbf{w}$) are then determined by using maximum likelihood. This approach has two advantages over the generalized approach; there are fewer adaptive parameters to be determined (linear dependence as compared to a quadratic growth) and the predictive performance will not suffer if the class-conditional density assumptions give a poor approximation of the true distributions.

In mutli-class classification, the posterior probabilities for a large class of distributions is given by a softmax transformation of linear functions of the feature variables

$$p(C_\mathrm{k}|\phi) = y_\mathrm{k} = \frac{exp(a_\mathrm{k})}{\sum_j exp(a_\mathrm{j})} \tag{1}$$

where the activation functions a_k are given by

$$a_\mathrm{k} = \mathbf{w_k}^\mathrm{T}\phi. \tag{2}$$

Due to the non-linearity of the activation functions, the maximum likelihood solution is no longer in closed-form; it is necessary to use an iterative optimization technique to find the unique minimum of the concave error function E($\mathbf{w}$). By using the Newton-Raphson method, the update takes the form

$$\mathbf{w}^{(\mathrm{new})} = \mathbf{w}^{(\mathrm{old})} - \mathbf{H}^{-1}\nabla E(\mathbf{w}) \tag{3}$$

where $\mathbf{H}$ is the Hessian matrix whose elements comprise the second derivatives of E($\mathbf{w}$) with respect to the components of $\mathbf{w}$.

By using the maximum likelihood and the negative logarithm of the likelihood, we can obtain the cross-entropy error function for the multiclass classification problem, which is the default error function used in the scikit-learn implementation.

$$\nabla_{\mathbf{w_j}} E(\mathbf{w}_1, ..., \mathbf{w}_\mathrm{K}) = \sum_{n=1}^{N} (y_\mathrm{nj} - t_\mathrm{nj})\phi_\mathrm{n} \tag{4}$$

The M x M Hessian matrix can be obtained by applying the Newton-Raphson update to the cross-entropy error function for the logistic regression model

$$\mathbf{H} = \nabla_{\mathbf{w_k}}\nabla_{\mathbf{w_j}} E(\mathbf{w}_1, ..., \mathbf{w}_\mathrm{K}) = -\sum_{n=1}^{N} y_\mathrm{nk}(\mathbf{I}_\mathrm{kj} - y_\mathrm{nj})\phi_\mathrm{n}{\phi_\mathrm{n}}^\mathrm{T}. \tag{5}$$

Finally, the iterative reweighted least squares equation shown in equation 3 can be applied iteratively until the parameters converge to the minimized error solution, and thus learning a maximum likelihood multi-class logistic regression classification model.

2) Linear Support Vector Machine: The support vector machine, commonly referred to as just SVM, is another classifier that was implemented as a baseline algorithm. SVM is a "decision machine", thus does not provide posterior probabilities. The method involves learning a hyperplane H that maximizes the distance between the decision boundary and any of the samples; this is called the maximum margin hyperplane. Consequently, any unclassified samples can be projected into the same space as the training data, and then a class can be assigned according to the learned decision boundary.

Multiclass classification using SVM can be of various different forms. However, scikit-learn's implementation utilizes the widely used one-against-one approach which trains K(K-1)/2 different 2-class SVMs on all possible pairs of classes, where K is the total number of classes in the modelled dataset. The test samples can then be classified according to which class has the highest number of "votes".

Learning the maximum margin hyperplane is based on learning two parameters; a set of weights $\mathbf{w}$ denoting the

normal vector of the hyperplane as well as some bias b describing the position of the hyperplane. The training task comprises of learning the hyperplane H = **w** + b by learning the parameters that maximize the margin for the given training data, which is equivalent to the optimization task

$$\min_{\mathbf{w},b} \quad \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{n=1}^{N}\xi_n \tag{6}$$

where the first term is the loss function, $C > 0$ controls the trade-off between the slack variable penalty and the margin, and ξ_n is the classification error known as the slack variable. The purpose of the second term in Eq. 6 is to softly penalize points that lie on the wrong side of the margin boundary. This comes up in models dealing with training data that is not linearly separable, therefore, the model will allow overlapping class distributions at a penalty. Therefore, Eq. 6 should be solved using the following constraints

$$t_n y(\mathbf{x}_n) \geq 1 - \xi_n, \quad n = 1,...,N \tag{7}$$

$$\xi_n \geq 0 \tag{8}$$

where t_n is the true class and $y(\mathbf{x}_n)$ is the predicted class.

3) Feed Forward Neural Network: The neural network implemented in the code uses standard backpropagation with gradient descent. This implementation does not allow to add layers of varying size. The number of nodes per layer and number of layers is specified during initialization. All the underlying linear algebra is done with NumPy. All nodes use the sigmoid activation function. This allows us to store the output of the node instead of the sum. The output $\sigma(x)$ is stored for each node during the feed forward pass and then the derivative is computed from the sigmoid function value in the backpropagation phase. The option is given to train in batches instead of stochastic gradient descent. The batch size is assumed to be a multiple of the size of the data. This can effectively be considered as mini batch gradient descent without the performance increase. At each iteration, the weights are updated in a storage array and once every batch the weights are changed in an attempt to decrease the variance in the weight's update. Across multiple epochs, the learning rate is divided by 2 to simulate a decreasing learning rate and to try to find a better solution.

We tested the implementation of the neural network by trying to learn the XOR function on binary input. With a proper set of hyper-parameters, this was successfully achieved with a one-hot encoding of the classes. A one-hot encoding is also used to predict the 31 classes of the images. The input for the image classification is a flattened image of 738 pixels (28x28).

4) Convolutional Neural Network: The performance of a convolutional neural network abbreviated as CNN henceforth depends on numerous factors, including its underlying structure, the amount of filters and their associated patches, the activation functions, weight initialization protocols, the learning parameters, etc. Consequently, its design consists more of an art by trial and error rather than some static rules that can be recursed over. That said, by taking into account the spatial relationship of different features instead of uniquely considering each pixel as a standalone component, multiple advantages can be gained over regular classifiers such as support vector machines or decision trees, especially when it pertains to imagery. For instance, line strokes, gradient patches, solid zones, and the likes are now extractable, along with the notion of proximity, something that is unlikely learned by the formers. Fortunately, given the semblance of the 28-by-28 preprocessed dataset with *MNIST* and *EMNIST*, an evident starting point presents itself as reusing tried CNN models that perform relatively well on the aforementioned in terms of prediction accuracy. Indeed, the final network layout fully depicted in *Appendix, Figure 2* is a derivate of the one featured at *https://bit.ly/2Rk6RUg*, with some minor modifications, and consists primarily of convolutional layers of varying filters count connected through pooling layers for dimensionality reduction, and dropout layers for regularization.

C. Hyper-parameter Tuning

Hyper-parameter tuning was done using sklearn's GridSearchCV for both the feed forward neural network and the baselines. To combine the hand-written neural network with sklearn's search, we wrote a wrapper for the neural network that extends the BaseClassifier class so that it can use all the features of GridSearchCV. Both logistic regression and linear SVM were tested on the 100x100 original image dataset and our reduced 55x55 image dataset. The neural network has only been tested on the reduced 28x28 dataset to shorten the training time.

For the baselines, a few hyper-parameters were considered. The number of parameters tested was limited by the computation time we had available.

1) Logistic Regression: Penalty was fixed to L2. The range of regularization value C was in log space from -3 to 0 with 20 values. The best set of hyper-parameters is with $C = 0.018$ on the reduced image dataset and $C = 0.012$ on the original dataset.
2) Linear SVM: The final hyper-parameters examined are the regularization value C with a linear kernel. During testing, the rbf kernel was tested with gamma values 0.1 and 1. The results were close to random so we decided to focus on the linear kernel SVM. The range of C values tested is from -3 to -1 in log space with only 5 values. The best set of hyper parameters is with $C = 0.01$ on the reduced dataset and $C = 0.001$ on the original dataset.
3) Feed Forward Neural Network: Note that the neural network was trained on the 28x28 reduced image set with 784 input nodes since it is quite expensive to compute. The hyper-parameters tested are the batch size, number of nodes per layer, number of layer,

number of epoch and learning rate. The learning rate is decreasing with the number of epoch by a factor of 2. Since this implementation only uses a basic gradient descent, it is very difficult to obtain good results for a complex problem like image classification. Multiple values were tested separately: learning rate from 0.001 to 1000, layer size from 4 to 1000, number of layers from 1 to 3, batch size of 10 or 100. No combination produced results better than 0.04. Final cross validation results are explained below.

4) CNN: Throughout the tuning process, various observations were made, and are listed below in no particular order: 1. While reducing the depth or width of the network drastically decreases its performance down to the single digits with respect to validation accuracy due to gross overfitting, an increase would only yield marginally better results at the cost of higher computational complexity and memory usage; 2. In general, as long as the stride size is fixed at one, relatively small patch sizes in the single digits do not greatly alter accuracy; 3. The He initializer produces much better results than the default Xavier initializer, with the first epoch already attaining an accuracy in the tens, leading to faster convergence; 4. The dropout amount does not seem to greatly alter the results, although given that it helps with regularization along with the batch normalization layer, its value was left at 0.5; 5. Amongst the different optimizers tested, RMSprop with a dynamic learning rate appears to perform best in terms of speed (as compared to the default stochastic gradient descent), and fitting (versus Adam which overfits relatively quickly using the default settings); 6. Although the last fully connected layer uses a softmax activation function to output the membership probability of each class, ReLU was chosen for all other layers, as it provided much better accuracy on default settings compared to sigmoid; 7. While larger batch sizes greatly accelerate computation at the cost of memory (capped at around 256 on GeForce GTX Titan Xp), smaller batches per update seem to produce more stable results, with 64 being a good compromise value; 8. Given that there is a limited amount of examples per class, only ten per cent of the training set was used for validation, which albeit small, can nonetheless be a good proxy for distinguishing overfitting scenarios.

III. Results

A. Baseline Results

Using the best hyper-parameters obtained from cross-validation on logistic regression and linear SVM, we obtained the following results:

Both linear SVM and logistic regression with the reduced image datasets were tested on the test dataset online with the following results:

TABLE I

Performance of baseline classifiers with the best hyper-parameters during cross-validation

Performance	Original Images	Reduced Images
Logistic Regression	0.0416	0.3415
Linear SVM	0.0488	0.3897

TABLE II

Performance of baseline classifiers on the test set

Classifier	Logistic Regression	Linear SVM
Performance	0.35600	0.40833

B. Feed Forward Neural Network

The feed forward neural network did not produce any significant performance during hyper-parameter tuning phase. This is due to not using a better optimizer for the gradient descent. The network obtained a local minimum which used a frequent label to classify images instead of trying to classify each image individually. With a batch size of 100, 15 epochs, 2 layers of 1000 nodes each and a learning rate of 1000 decreasing by a factor of 2 each epoch (reaching 0.06 in the last epoch), the performance during cross validation was of 0.0272.

C. Convolutional Neural Network

The final model achieves an 84 per cent training and 78 per cent validation accuracy with losses between 0.8 and 0.9, at which point it begins to exhibit overfitting through a divergence of the formers (refer to *Appendix, Figure 3*). Although this is better than any of the linear classifiers attempted along with the neural network implementation, the fact that it is only able to accurately predict 77 per cent of the test set as per the score obtained on Kaggle hints at room for improvement. From the output metrics, the noticeably worse performers in terms of false positives and negatives were the *rifle* and *squiggle* classes, obtaining a precision slightly below 0.9, followed closely by the *moustache* class. In contrast, the best performers belonged to the *rollercoaster* and *sailboat* classes, with less than ten false predictions in total out of 9,000. This is to be expected, as given the small input dimensions, the challenge in distinguishing between different types of strokes increases, whereas more complex figures still retain their overall complex characteristics. With that said, training conducted on the 55-by-55 dataset manages an 80 per cent accuracy before overfitting. A final experiment using the latter dataset with a CNN modelled after the *VGG19* network was done to verify whether such metrics are the consequence of a simplistic layout schema that is unable to capture the additional information contained in larger inputs, with overall similar results.

IV. Discussion

A. Baselines

Logistic regression and linear SVM performed quite well under the reduced 55x55 images compared to the original

dataset. This is to be expected since the original dataset contains a lot of noise and the hand-drawn images are not centered in the frame. When trying to separate the pixel space on the original images, both algorithms have trouble determining the boundaries between classes. Once given the cleaned and reduced dataset, it is much more easier to separate the classes. This is still not good enough to make accurate prediction but it is much better than random prediction. These algorithms do not consider the shape of the objects at all, only the value of activated pixels. This may cause issues when objects have a similar shape but are not the same as the ones in the training set. This is where convolutional neural networks will be able to get the additional power to make good predictions.

B. Feed Forward Neural Network

The main problem with our implementation is that the network is stuck in a local minimum. This is mainly due to the fact that the basic gradient descent algorithm will converge to the closest minimum, without trying to look for other minima that could provide better performance. Optimizers which use momentum should alleviate that problem.

To improve the performance on the feed forward neural network on this image classifying problem, we could implement a different optimization method like adam. A more complex framework to allow adding hidden layers of custom size and with different activation functions would help with the performance and the vanishing gradient problem in deep neural nets. Current implementation is too restrictive on the structure of the neural network.

C. Convolutional Neural Network

While better weights initialization may yield concrete improvements, the fact that all three pipelines were capped at around 0.8 suggests that the problem lies elsewhere– possibly with the input feed. Admittedly, given that the noise reduction algorithm used exhibits trouble in removing artifacts adjacent to the objects of interest, thumbnails such as the rifle and moustache with close proximity noise may be interpreted as lines rather than correctly cleaned up. Improvements to this could include the use of an auto-encoder for predictive pixel elimination. Furthermore, the randomized transforms applied, while small, may bring portions of the image out-of-frame, leading to partial visuals that could potentially resemble objects of other classes. Nevertheless, aside from the shortcomings exposed by a potentially lacking preprocessing step, a 77 per cent accuracy is respectable regardless, given that merely 300 examples of each class were provided, all of which masked by considerable noise.

APPENDIX

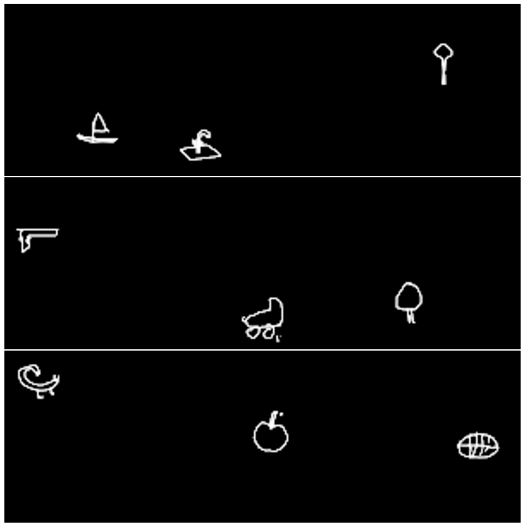

Figure 1. Randomly sampled images after noise reduction pipeline without transforms or resize.

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 64)	640
conv2d_2 (Conv2D)	(None, 24, 24, 64)	36928
max_pooling2d_1 (MaxPooling2	(None, 12, 12, 64)	0
dropout_1 (Dropout)	(None, 12, 12, 64)	0
conv2d_3 (Conv2D)	(None, 12, 12, 128)	73856
conv2d_4 (Conv2D)	(None, 12, 12, 128)	147584
max_pooling2d_2 (MaxPooling2	(None, 6, 6, 128)	0
dropout_2 (Dropout)	(None, 6, 6, 128)	0
conv2d_5 (Conv2D)	(None, 6, 6, 256)	295168
dropout_3 (Dropout)	(None, 6, 6, 256)	0
flatten_1 (Flatten)	(None, 9216)	0
dense_1 (Dense)	(None, 256)	2359552
batch_normalization_1 (Batch	(None, 256)	1024
dropout_4 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 31)	7967

Total params: 2,922,719
Trainable params: 2,922,207
Non-trainable params: 512

Figure 2. MNIST-based CNN model layout.

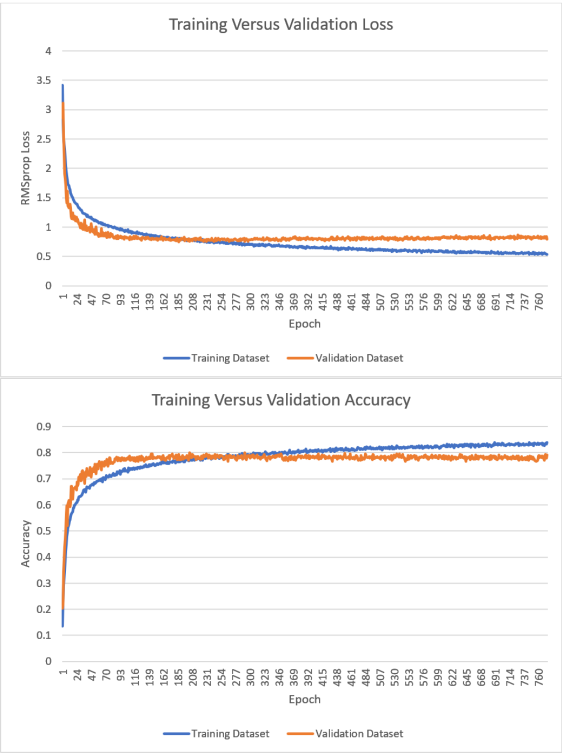

Figure 3. Default CNN metrics across training epochs.