Classify Hand-Drawn Pictures

Team: TVTR

Gupta, Abhijay — abhijay.gupta@mail.mcgill.ca

Ijaz, Ramsha — ramsha.ijaz@mail.mcgill.ca

Jian, Yuechun — yuechun.jiang@mail.mcgill.ca

Introduction

Image Analysis has been gaining an immense popularity in the field of Machine Learning and there is a need to develop efficient learning algorithms that could tackle any kind of images proposed. In order to delve deeper into the algorithms used for Image Analysis, we trained and tested a given dataset[1] of hand-drawn images on various Machine Learning algorithms that were able to classify and reason about an image. In order to evaluate the efficiency of every algorithm, we used a common F1-Score that evaluated the performance of all the algorithms implemented.

Feature Design

Preprocessing is the first step in any type of data analysis as the data contains unnecessary elements (noise) that drastically affects the performance of classification models. For our purpose, we implemented the following procedure in order to eliminate the noise and select features from the given images:

1. Convert the images to grayscale

This helps to separate the luminance and chrominance planes. This separation is very important as it allows to detect different features and edges of objects in an image.

2. Find contours

Finding contours in an image distinguishes objects of interest from the rest of the noise. This helped to eliminate the noise and produce an image containing only the object required for our classification.

3. Crop objects

The denoised image is cropped in order to eliminate the blank space surrounding the object of interest. These cropped images ultimately allows the classifiers to better predict the test images since the image primarily contains the object of interest now.

.

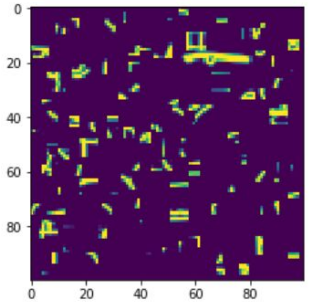

Fig 1.1.. Noisy Image of a ‘rifle’

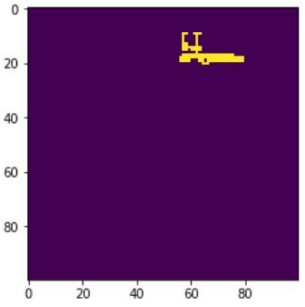

Fig 1.2. Denoised Image of a ‘rifle’

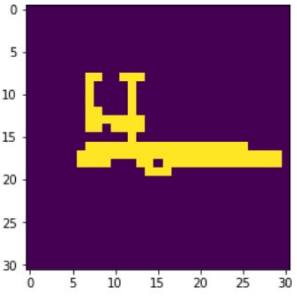

Fig 1.3. Cropped image of the denoised image of a ‘rifle’

[1] Modified subset of hand-drawn images from Google’s Quick Draw project scaled to 100x100

Algorithms

1. Support Vector Machine (SVM)

This is a discriminative classifier that categorizes the training data based on an optimal hyperplane. The main aim is to maximise the margin between the hyperplane and distinct data points.

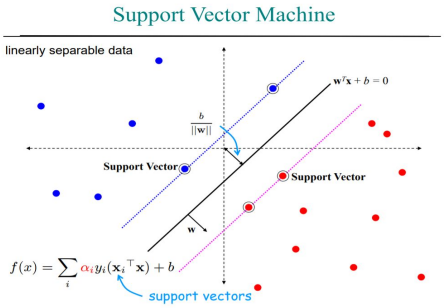

Fig 2.1 Hyperplane separating data points of two classes. [1]

The margin is maximised by minimising the hinge loss function of SVM:

$$min_w \lambda \parallel w \parallel^2 + \sum_{i=1}^{n} (1 - y_i \langle x_i, w \rangle)_+$$

The partial derivative is taken with respect to the weight in order to in order to find the gradient of the margin. This gradient is then used to update the weights of the hyperplane.

2. Logistic Regression

Logistic Regression is a Probabilistic Discriminative Model which implies that it first models the decision boundary between different classes. Based on this boundary, it is able to learn the conditional probability distribution of the training data with different classes. These probabilities are calculated using the logistic sigmoid function:

$$p(x) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

The aim is to find optimal weights for the sigmoid function that is able to yield a good estimate for the probabilities of the input data. These weights are derived by minimising the cost function which is the negative logarithm of the likelihood:

$$E(w) = -\ln p(t|w)$$

$$= -\sum_{n=1}^{N} \left\{ t^{(n)} \ln y^{(n)} + (1 - t^{(n)}) \ln (1 - y^{(n)}) \right\}$$

3. Naive Bayes

Naive Bayes is a Generative Classifier which means that it considers every feature as being independent and models the actual distribution of each class. Using individual distributions, it then computes the conditional probability distribution of the training data with different classes. These probabilities are computed by taking the maximum probability of the example belonging to a certain class, computed as follows:

$$y = argmax_y P(y) \prod_{i=1}^{n} P(x_i|y)$$

4. k-Nearest Neighbours

kNN is a non-parametric classifier that does not make any assumption on the underlying data distribution. A datapoint is classified by a majority vote of its k-nearest neighbours. The following figure represents the classification between two classes:

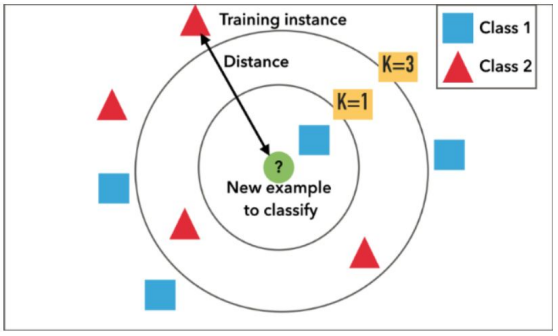

Fig 2.2. A visual of how kNN works. [2]

5. Decision Trees

Decision Trees classify the examples by sorting them down the tree from the root to a leaf node, which provides classification for the instance. Using a cost function, different features are split differently into multiple nodes. The split with the lowest cost is selected for creating the decision tree on the dataset. The Gini split can be used to compute the quality of the split:

$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

The following uses Gini Index which computes the frequency of each class at each node:

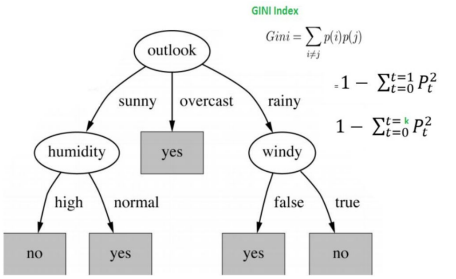

Fig 2.3. Decision Tree of weather forecast based on Gini Index [3]

6. Convolutional Neural Network

CNN is a feed-forward neural network based on multilayer perceptrons. A Perceptron is a type of linear classifier that works as shown:

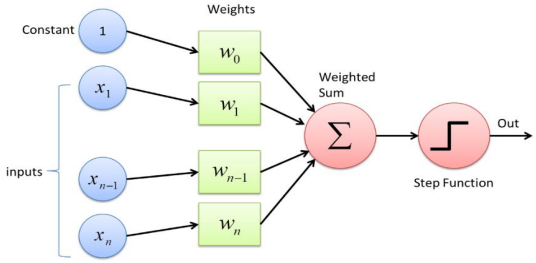

Fig 2.4. Perceptron [4]

The step function used in our model is the sigmoid function. CNN first performs a series of convolutional and pooling operations in order to learn different features of the object. The fully connected layers act as the classifier which assigns a probability to the object depicted by the image.

Methodology

- The given dataset was split into 60% for train, 20% for valid and 20% for test data. Splitting with these proportions increases the accuracy of the classifying models.

  The hyperparameters of **Naive Bayes, SVM, Logistic Regression, Decision Trees and kNN** were selected using the Grid Search Method. This method enabled different combinations of the hyperparameters for each fold of the dataset and therefore, computed optimal hyperparameters for each classifier based on the F1-score.

- **kNN** was further optimized using max-pooling. Compared to the standard kNN, this optimized version had significantly better results (increase from 19.85% to 48%).

- Many tunable parameters came into play when we tried to optimize our **CNN**. We simulated several classic cnn models such as ResNet and VGG16. We introduced max-pooling layers to reduce computational cost and overfitting to some extent. We added one regularization method Dropout to the output layer. A proportion of nodes in the layer are randomly ignored for each training sample. This forces the network to learn features in a distributed way. This technique also improves generalization and reduces the overfitting. Combining *convolutional* and *pooling layers*, CNN is able to combine local features and learn more global features of the image. Our best CNN gives 69% accuracy.

Results

1. CNN:

Our CNN has 1 input layer, 3 convolutional layers, and an output layer. After each conv layer, we apply a nonlinear layer immediately afterward. One crucial hyperparameter is the number of convolutional layers. We found 3 conv layers and 4 conv layers give similar results while more conv layers decreases the accuracy. In total, out best model has 10 layers. We experimented different number of filters and filter sizes. Generally the more filter applied, the quicker the model converges; at the same time, more computational resources are consumed. As we have limited GPU utilization on Google Cloud, we decided to increase the filter number gradually. We settled on a filter size of 64 for the first 2 layers and 128 for the rest 2 layers. 5 x 5 is the optimal filter size. From our experiments, we found that pooling size of 2 gives the best result. Therefore, each of the max-pooling layers looks at the 2 neighboring pixels and picks the maximal value.
In the end, we trained the cnn using 100 epochs with batch size being 512, we get an accuracy of 69% on the test data.

2. KNN:

With max pooling

We tuned two parameters: the *number of nearest neighbors* and the *pool size of max pooling*. For example, if we set the pool size to be 3, then each 3 x 3 matrix in the original image will be represented as 1 number in the next layer. We find 15 to be the optimal number of nearest neighbors which gives f1-score of 48% when combined with a pool size of 2. This strategy performs far better than regular kNN.

With OpenCV match shapes

We came up with a new way to compute the distance between two images. We used the matchShape function from openCV. It gives a score of the similarity between the two images between 0 and 1. We tuned the hyperparameter number of nearest neighbors in the range from 10 to 300. However, this method gives us the same accuracy as regular KNN, 16% with 15 nearest neighbors.

3. Logistic Regression:

The most important hyperparameter for Logistic Regression is *C* which is the inverse of the regularization strength.

This prevents overfitting as it applies a penalty to increasing the magnitude of the parameter values. This therefore, minimizes the error during the prediction phase as the values of the parameters will have been regularized yielding a good overall score by the model.

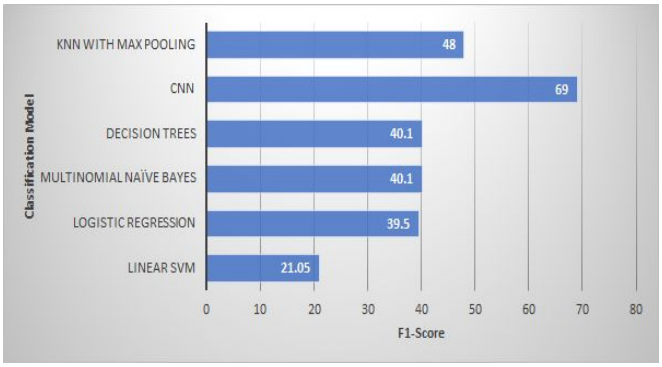

Fig 3.2. Performance graph for Classification Models used for Image Analysis

Classification Models	List of Hyper – parameters	Best Hyper-parameter(s)	F1-Score
Linear SVM	{'C': [2*i for i in range (1, 6)], 'tol': [10**(-i*2) for i in range (1, 6)]}	{'tol': 0.01, 'C': 6}	21.05%
Logistic Regression	{'C': [0.000000001, 0.0000001, 0.000001,0.00001, 0.001,0.01,0.1,1,10,100]}	{'C': 1e-06}	39.5%
Multinomial Naïve Bayes	{'alpha': [0.0001, 0.001, 0.01, 1, 10, 20, 30, 50, 65,100]}	{'alpha': 50}	40.1%
Decision Trees	{'criterion': ['gini', 'entropy'], 'max_depth': [10, 8, 6, 4, 2], 'min_samples_split': [600, 400,200, 100], 'min_samples_leaf': [600, 400, 200, 100], 'max_features': [10, 50, 100, 200, 500, 900]}	{'min_samples_leaf': 100, 'max_features': 500, 'min_samples_split': 100, 'max_depth': 8, 'criterion': 'gini'}	40.1%
kNN with Max Pooling	{'n_neighbors':10 ,15, 20, 25, 30}	{'n_neighbors': 15}	48%
CNN	{'number of layers': [3, 10, 15, 32] 'number of: [convolutional layers': [1, 2, 3, 4, 5, 16]}	{'number of layers: 10, 'number of convolutional layers': 3}	69%

Fig 3.1 Table depicting the performance of the classifiers and the hyper-parameters used for each classifier

Discussion

In terms of accuracy, CNN yielded more accurate results than any of the classification models. Decision Trees could have been optimized by pruning in order to avoid overfitting.

In order to improve our accuracy, we initially tried to train to our models on images at different rotations. However, this significantly impacted our resources and did not yield better scores than the denoised images without rotations. In order to prevent this, more research could be carried on better understanding and implementing rotations on images and then training our models on these images.

CNN yields better result with non-linear ReLU layer following each conv layer. A ReLU layer applies the function f(x) = max(0, x) to all of the values in the input volume. This layer changes all the negative activations to 0. The purpose of this layer is thus to introduce nonlinearity to a system that has just been computing linear operations during the conv layers. It also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers. From our search, we found that in the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster without making a significant difference to the accuracy.

Statement of Contribution

Abhijay:

- Preprocessing of the images
- Generating the work environment for the team on Google Cloud Platform
- Hyperparameter tuning for baseline models
- Organization of work through Github

Ramsha:

- Coding and testing Generative(Naive Bayes) and discriminative classifiers(SVM, Logistic Regression and Decision Trees)
- Drafting, editing and revising the report

Yuechun:

- Implemented and tuned hyperparameters for CNN and kNN.
- Contributed to image preprocessing.

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