**DIGIT RECOGNITION**

# **1. Introduction**

## **1.1 Purpose of this document**

This paper presents the results of handwritten digit and english character recognition on well-known image databases using state-of-the-art feature extraction and classification techniques.

## **1.2 Intended Audience**

The intended audiences for this document are all the students of the class, the faculties of the course CS6140-02 Machine Learning - Professor Kevin Small and all the teaching assistants.

## **1.3 Problem Description**

**Handwriting recognition** (HWR) is the ability of a computer to receive and interpret intelligible handwritten input from sources such as paper documents, photographs, touch-screens and other devices. This problem is importantly growing because a lot of computer devices now use pen based user input and recognition of user input is necessary for useful processing of this information.

Potential work has been done for classifying MNIST dataset using techniques like KNN , linear SVM and CNN. But we take into consideration various other datasets like Semeion,Street View Housing Number and USPS dataset to evaluate the robustness of our classification models. We also provide an analysis of the EMNIST letter dataset classified using our models.

The following are the steps we took-

* Data Collection - We collected the datasets in image and numeric formats. Not all datasets were readily available in the image format. Ex - We had to transform the SVHN dataset.
* Feature Extraction - Computed the Histogram of Oriented Gradients (HOG) from the image-based data. Numerical data was used as it is.
* Cross Validation - Used Grid Search to determine the best hyper-parameters for each classifier we ran the datasets on.
* Performance Metrics - Computed additional metrics over accuracy such as precision, recall, f1-score to evaluate the performance of our classifiers.
* Evaluation - We always used one best-tuned model and ran it against all datasets for performance comparison
* Result Comparison - Convolutional Neural Nets perform the best on our datasets with average accuracies over of 93% while MLP, K-Nearest Neighbors and Linear SVM classifier performed quite close to CNN, proving to be useful for character recognition & learning.

## **1.4 Definitions and acronyms**

### **1.4.1 Definitions**

|  |  |
| --- | --- |
| **Name** | **Definition** |
| Precision |  |
| Recall |  |
| F1 score |  |

### **1.4.2 Acronyms and abbreviations**

|  |  |
| --- | --- |
| **Abbreviation** | **Definition** |
| HOG | Histogram of Oriented Gradients |
| SVM | Support Vector Machines |
| CNN | Convolutional Neural Network |
| SVHN | Street View Housing Numbers |
| MNIST | Modified National Institute of Standards and Technology |
| USPS | US Postal Service |
| MLP | Multi-Layer Perceptron |
| KNN | K-Nearest Neighbours |
| FC | Fully Connected |

## **1.5 References**

* Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng - Reading Digits in Natural Images with Unsupervised Feature Learning NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011.
* Norhidayu binti Abdul Hamid and Nilam Nur Binti Amir Sjarif, “Handwritten Recognition Using SVM, KNN and Neural Network”
* Reza Ebrahimzadeh and Mahdi Jampour ,“Efficient Handwritten Digit Recognition based on Histogram of Oriented Gradients and SVM”
* https://yashk2810.github.io/Applying-Convolutional-Neural-Network-on-the-MNIST-dataset/
* https://gurus.pyimagesearch.com/lesson-sample-histogram-of-oriented-gradients-and-car-logo-recognition/

# **2. Problem Definition**

## **2.1 Task Definition**

The goal of this project is to analyze & compare the performance of different classifiers on the MNIST, SVHN, SEMEION and USPS datasets, namely - KNN (K-Nearest Neighbors), Linear SVM (Support Vector Machines), MLP (Multilayer Perceptron) and CNN (Convolutional Neural Network).

We would be incorporating the Histogram of Oriented Gradients feature representation scheme in our analysis to enhance our evaluation.

## **2.2 Algorithmic Specifications**

We have used 4 different machine learning algorithms for the analysis. Below is a brief description of each of the algorithms.

**2.2.1 K-Nearest Neighbours**

* The k-Nearest Neighbor classifier is by far the most simple image classification algorithm. In fact, it’s so simple that it doesn’t actually “learn” anything!. Instead, this algorithm simply relies on the distance between feature vectors.
* There are two clear parameters that we are concerned with when running the k-NN algorithm -
  + *The value of K*
  + *Distance Metric* - Euclidean (√𝚺(x - y)2), Manhattan (𝚺|x - y|), Minkowski (𝚺(|x - y|p)1/p), Chebyshev (max|x - y|).
* If K is too small (such as when k=1), then we gain efficiency, but become susceptible to noise and outlier data points. However, if k is too large, then we are at risk for over-smoothing our classification results and increasing bias.
* For the digit recognition, we will be using the Euclidean distance metric.

**2.2.2 Linear Support Vector Machine**

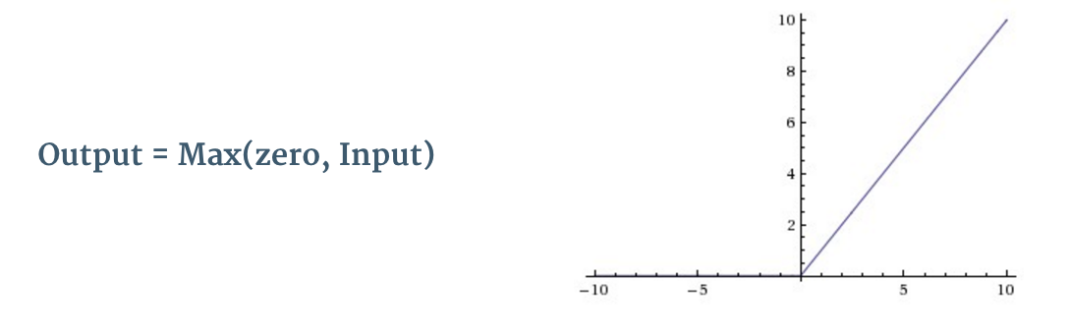
* The reason for adding a linear-kernel based SVM to our list of classifiers was mainly because it is less prone to overfitting and is theoretically easier to experiment and correlate to.
* Our datasets have large number of features in their default numeric (pixel-valued) representations (32x32x3, 28x28X3, etc.). SVMs have been proven to be good for such scenarios.
* When working with HOG, the features are lesser and number of samples remain the same (huge).
* Linear SVMs are generally better at computation speed and memory usage

**2.2.3 Convolutional Neural Net**

* Essentially, every image can be represented as a matrix of pixel values. We develop a CNN architecture which identifies an image represented as a matrix of pixel values
* There are four main operations in CNN: Convolution, Non Linearity (ReLU), Pooling or Sub Sampling and Classification (Fully Connected Layer).
* A convolution in a CNN is an element wise multiplication i.e. dot product of an image matrix and a filter.In the below example, the image is a 5 x 5 matrix and the filter going over it is a 3 x 3 matrix. A convolution operation takes place between the image and the filter and the convolved feature is generated. Each filter in a CNN, learns different characteristic of an image.



* A filter slides over the input image to produce a feature map.The CNN *learns* the values of these filters on its own during the training process (although we still need to specify parameters such as number of filters, filter size, architecture of the networketc. before the training process). The more number of filters we have, the more image features get extracted and the better our network becomes at recognizing patterns in unseen images.
* An additional operation called ReLU activation has been used after every Convolution operation. ReLU stands for Rectified Linear Unit and is a non-linear operation. Its output is given by:



ReLU is an element wise operation (applied per pixel) and replaces all negative pixel values in the feature map by zero. The purpose of ReLU is to introduce non-linearity in our ConvNet, since most of the real-world data we would want our ConvNet to learn would be non-linear

* Spatial Pooling (also called subsampling or downsampling) reduces the dimensionality of each feature map but retains the most important information. Spatial Pooling can be of different types: Max, Average, Sum etc. In case of Max Pooling, we define a spatial neighborhood (for example, a 2×2 window) and take the largest element from the rectified feature map within that window. Instead of taking the largest element we could also take the average (Average Pooling) or sum of all elements in that window. In practice, Max Pooling has been shown to work better.
* The Fully Connected layer is a traditional Multi Layer Perceptron that uses a softmax activation function in the output layer (other classifiers like SVM can also be used, but will stick to softmax in this post). The term “Fully Connected” implies that every neuron in the previous layer is connected to every neuron on the next layer
* The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the Fully Connected layer is to use these features for classifying the input image into various classes based on the training dataset.

#### **Putting it all together - Training using Backpropagation**

Input Image = A digit image in grayscale - let’s assume 8 for now

Target Vector = [0, 0, 0, 0, 0, 0, 0, 0 ,1, 0]

The overall training process of the Convolution Network may be summarized as below:

* **Step1:** We initialize all filters and parameters like input shape
* **Step2:** The network takes a training image as input, goes through the forward propagation step (convolution, ReLU and pooling operations along with forward propagation in the Fully Connected layer) and finds the output probabilities for each class.
  + Let’s say the output probabilities for the input image above are [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.2,0.1, 0.1]
  + Since weights are randomly assigned for the first training example, output probabilities are also random.
* **Step3:** Calculate the total error at the output layer (summation over all 10 classes)
* **Step4:** Use Backpropagation to calculate the *gradients* of the error with respect to all weights in the network and use *gradient descent* to update all filter values / weights and parameter values to minimize the output error.
  + The weights are adjusted in proportion to their contribution to the total error.
  + When the same image is input again, output probabilities might now be [0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1,0.2, 0.1], which is closer to the target vector.
  + This means that the network has *learnt* to classify this particular image correctly by adjusting its weights / filters such that the output error is reduced.
  + Parameters like number of filters, filter sizes, architecture of the network etc. have all been fixed before Step 1 and do not change during training process – only the values of the filter matrix and connection weights get updated.
* **Step5:** Repeat steps 2-4 with all images in the training set.

The above steps *train* the ConvNet – this essentially means that all the weights and parameters of the ConvNet have now been optimized to correctly classify images from the training set.

When a new (unseen) image is input into the ConvNet, the network would go through the forward propagation step and output a probability for each class (for a new image, the output probabilities are calculated using the weights which have been optimized to correctly classify all the previous training examples). If our training set is large enough, the network will (hopefully) generalize well to new images and classify them into correct categories.

**2.2.4 Multi-layer Perceptron**

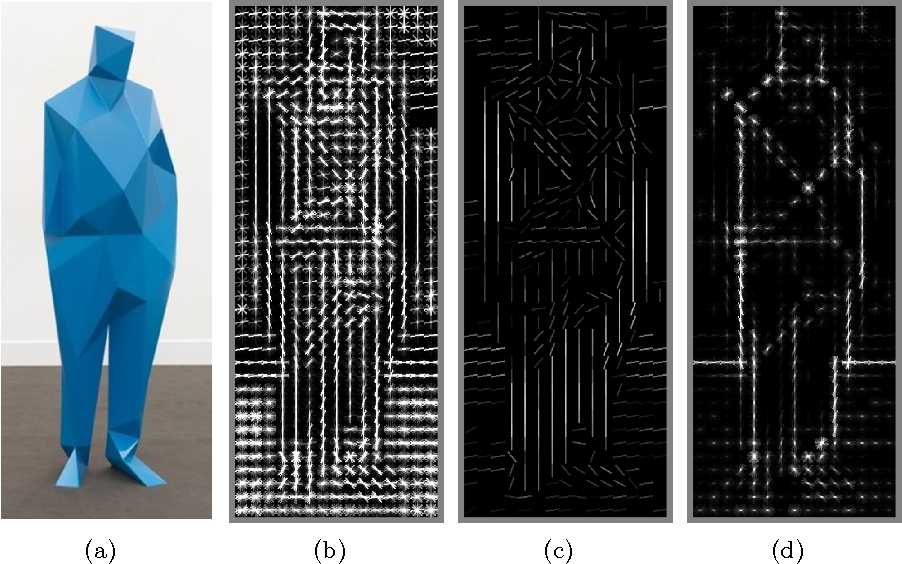
* The oldest and most simplest, oldest and most known neural network type is the multilayer perceptron (MLP).
* Input Image = A digit image in grayscale - let’s assume 8 for now. The input image is represented as a single row vector where each value is a feature. Target Vector = [0, 0, 0, 0, 0, 0, 0, 0 ,1, 0]
* The first layer we use is Keras implementation of fully connected layer i.e. dense layer with 1024 neurons. We apply the ReLu activation function just like the previous model to prevent the problem of vanishing gradients.At this point we also need to specify the input image feature dimension. For MNIST the dimension is 784(28 x 28)
* The next layer is a dropout layer. This layer prevents overfitting by randomly disabling a said percent of neurons from the previous output. This percent we used as 50%
* The third layer is another hidden layer with 1024 neurons. The use of multiple hidden layers is to boost the accuracy of prediction
* We use a dropout layer again and finally a FC layer to predict the class of the input image. For back propagation and weight adjustment we use stochastic gradient descent with learning rate 0.01 and decay rate as 1e-6.

**3. Experimental Methods and Results**

**3.1 Experimental Methodology**

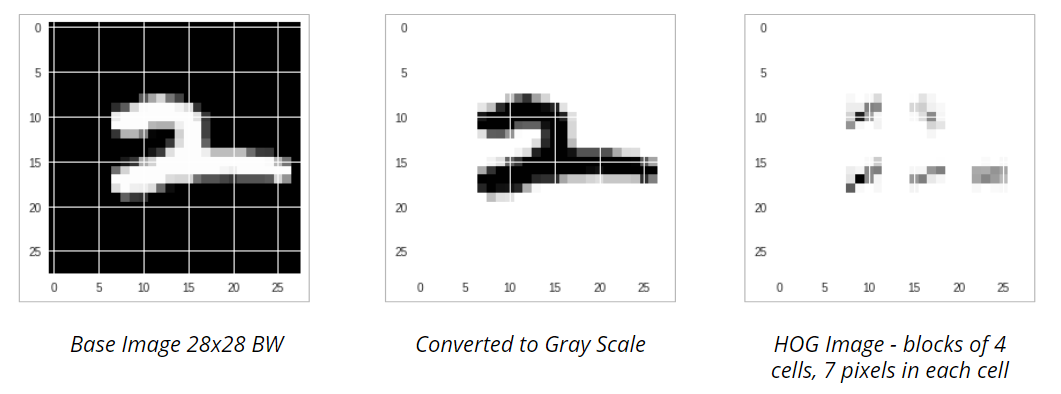
**3.1.1 Histogram of Oriented Gradients**

* An HOG descriptor is computed by calculating image gradients that capture contour and silhouette information of grayscale images. Gradient information is pooled into a 1-D histogram of orientations, thereby transforming a 2D image into a much smaller 1-D vector that forms the input for machine learning algorithms.
* HOG feature descriptors and their extensions remain one of the few options for object detection and localization that can remotely compete with the recent successes of deep neural networks (DNN)



**HOG - How is it obtained?**

* First, the gradients for each pixels (orientation and magnitude) are computed. The pixels are divided into n, mxm pixels squares.
* For each square, gradient direction histogram is computed for over p directions (our case p=9).
* Finally, the histograms are concatenated to obtain a (n\*m\*2) dimensional feature vector**.**

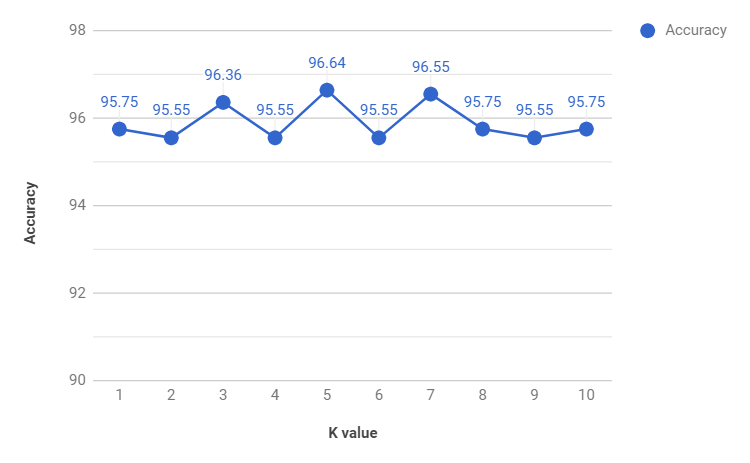
****

**3.2 Results and Analysis**

**3.2.1 K-Nearest Neighbors (KNN)**

* Using GridSearch and cross validation, we found that K=5 gives the best estimate.
* In default MNIST representation, an image is represented as 1-dimensional array of 784 (28 x 28) values where each value is a pixel of the image between 0 & 255.
* In this representation, KNN model takes approximately 9 hrs to predict the labels for the test set because of the higher dimensionality. We got an accuracy of 96.14%.
* With Histogram of Oriented Gradient (HOG) representation, the dimensionality is reduced (to 36) and the accuracy increases slightly to 96.64% (K = 5).  
  For SVHN dataset, the HOG feature representation gives an accuracy of 71.90%.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dataset** | **MNIST** | | **SVHN** | **SEMEION** | **USPS** |
| **HOG** | **Default** | **HOG** | **Default** | **Default** |
| **Accuracy** | 96.64% | 96.14% | 71.90% | 89.97% | 96.99% |
| **Precision** | 0.97 | 0.97 | 0.73 | 0.91 | 0.97 |
| **Recall** | 0.97 | 0.97 | 0.72 | 0.90 | 0.97 |
| **F1 score** | 0.97 | 0.97 | 0.72 | 0.90 | 0.97 |

****

**3.2.2 Linear SVM**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Dataset** | **MNIST** | | **SVHN** | **SEMEION** | **USPS** |
| **HOG** | **Default** | **HOG** | **Default** | **Default** |
| **Accuracy** | 94.41% | 87.67% | 78.92% | 86.64% | 94.49% |
| **Precision** | 0.95 | 0.88 | 0.79 | 0.87 | 0.94 |
| **Recall** | 0.95 | 0.88 | 0.78 | 0.87 | 0.94 |
| **F1 score** | 0.95 | 0.88 | 0.79 | 0.86 | 0.94 |

**3.2.3 Convolutional Neural Network**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Dataset** | **MNIST** | | **SVHN** | | **Semeion** | | **USPS** | |
| **Before** | **After** | **Before** | **After** | **Before** | **After** | **Before** | **After** |
| **Accuracy** | 99.14% | 99.26% | 86.87% | 85.59% | 92.73% | 94.49% | 97.78% | 97.78% |
| **Loss** | 0.028 | 0.028 | 2.043 | 2.179 | 0.195 | 0.174 | 0.079 | 0.077 |
| **Precision** | 0.991 | 0.993 | 0.871 | 0.860 | 0.933 | 0.946 | 0.978 | 0.978 |
| **Recall** | 0.991 | 0.993 | 0.869 | 0.856 | 0.927 | 0.945 | 0.978 | 0.978 |
| **F1 score** | 0.991 | 0.993 | 0.868 | 0.855 | 0.928 | 0.945 | 0.978 | 0.978 |

**3.2.4 Multi-Layer Perceptron**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Dataset** | **MNIST** | | **SVHN** | | **Semeion** | **USPS** |
| **Default** | **HOG features** | **Default** | **HOG features** |
| **Accuracy** | 98.33% | 95.61% | 85.59% | 80.21% | 91.97% | 96.27% |
| **Loss** | 0.054 | 0.134 | 0.503 | 0.668 | 0.232 | 0.132 |
| **Precision** | 0.993 | 0.956 | 0.862 | 0.808 | 0.923 | 0.963 |
| **Recall** | 0.993 | 0.956 | 0.856 | 0.802 | 0.920 | 0.963 |
| **F1 score** | 0.993 | 0.956 | 0.857 | 0.803 | 0.919 | 0.963 |

**3.3 Discussion**

* One main advantage of the k-NN algorithm is that it’s extremely simple to implement and understand. Furthermore, the classifier takes absolutely no time to train. But classifying a new testing data point requires a comparison to every single data point in our training data, which scales to **O(N)**, making working with large datasets computationally prohibitive.
* We tried to combat this problem by using HOG feature representation and achieved an improved accuracy. The other alternative we want to try further is to use Approximate Nearest Neighbor (ANN) algorithms (such as kd-trees, FLANN, and random projections, etc.), but this requires that we trade space/time complexity for the the “correctness” of our nearest neighbor algorithm.
* Finally, the k-NN algorithm is more suited for low-dimensional feature spaces. Distances in high-dimensional feature spaces are often unintuitive. We used k-NN on the datasets as a “first attempt” to obtain a baseline for classification accuracy and then try out the SVM and Neural Network models.
* The main difference between the convolutional neural network (CNN) and multi-layer Perceptron(MLP) is that CNN has layers of convolution and pooling. This means that the first layer that comes after the input, does not use all input features at the same time but rather features that are "connected".
* Although MLP can be used to learn features as well as classify data, it is not practical to apply this architecture to images. A very high number of neurons would be necessary, even in a shallow architecture, due to very large input sizes associated with images, where each pixel is a relevant variable.
* Due to the full connectivity between nodes, MLP suffers from the curse of dimensionality and thus does not scale well for high resolution images.

**4. Future Work**

* As a future scope, we could try to incorporate a touch-based interface to capture handwritten characters and analyze them in real-time with our trained ML & DNN models.

# **5. Conclusion**

* Convolutional Neural Nets perform the best on our datasets with average accuracies over of 93%.
* MLP & K-Nearest Neighbors classifiers performed quite close to CNN, proving to be useful for character recognition & learning.
* The DNN models that did well on digit recognition, performed equally well on alphabet character recognition from the EMNIST dataset. Thus, optical character recognition proves to not vary vastly from the best digit recognition models.
* HOG features are great for object / image detections, however the slight discontinuities & deformities in images causes loss in model performances.