Project 2

Statistical Patter Recognition

Girguis Sedky

1 Support Vector Machine

In this method, we use train a support vector machine (SVM) on the training FMNIST data, which consists of 60,000 images of 10 different clothing items and deploy the trained algorithm on the test data, which consists of 10,000 images. LIBSVM was used here. The dimension of the image vectors are reduced from $28 * 28 = 784$ to 50 dimensions using principal component analysis before the training procedure. We try three different Kernels, including a linear kernel, a third degree polynomial kernel, and a radial basis function kernel (RBF). Figure 1 shows a comparison of the accuracy of the SVM algorithm in classifying the test data with each kernel. All of the algorithms do relatively well. The SVM algorithm trained using the RBF kernel is the most accurate classifier.

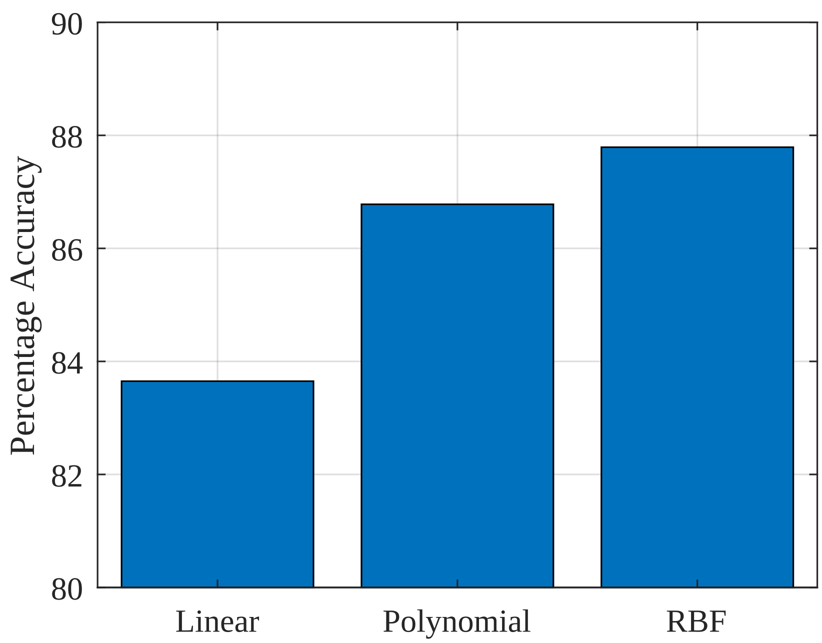

Figure 1: Bayesian classifier

2 Convolutional Neural Network

In this section, we utilize a CNN to classify the test images. No PCA is performed o the data set that is used here. The network used is a variant on LeNet that is shown in Fig. 2. It consists of 7 layers and an alternating convolution and max pooling application from one layer to the next. ReLU is introduced in the final transition layer. MatCovNet toolbox was utilized to develop, train, and implement this neural network.

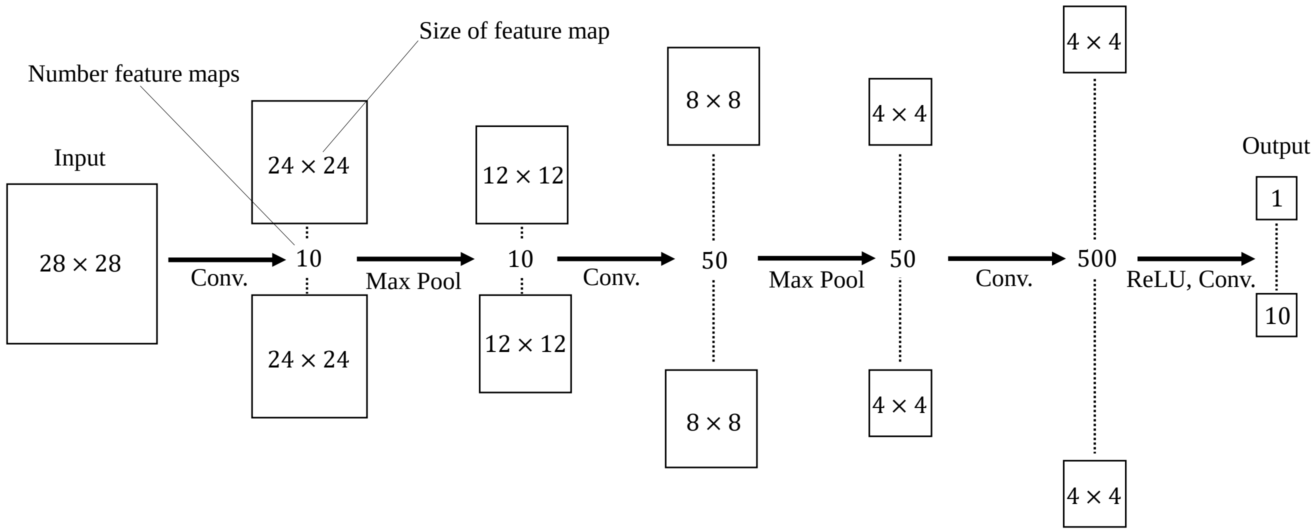

Figure 2: Convolutional Neural Network layout

The data is divided into batches of 100 images, and the network goes through the entire data set 15 times (15 epochs) during the gradient descent procedure to tune all the weights. Figure 3 shows the rate of descent of error as a function of the epoch number. The error rate has a steep drop during the first couple of passes through the data, and then its continues decreasing with a steadier pace. The trained CNN is able to classify the test images with 90.6% accuracy which is a higher accuracy compared to SVM. This improvement in accuracy can be attributed to the increased complexity of the classifier, large number of empirical weights that can be tuned in training, and non-linearity introduced by the ReLU operator.

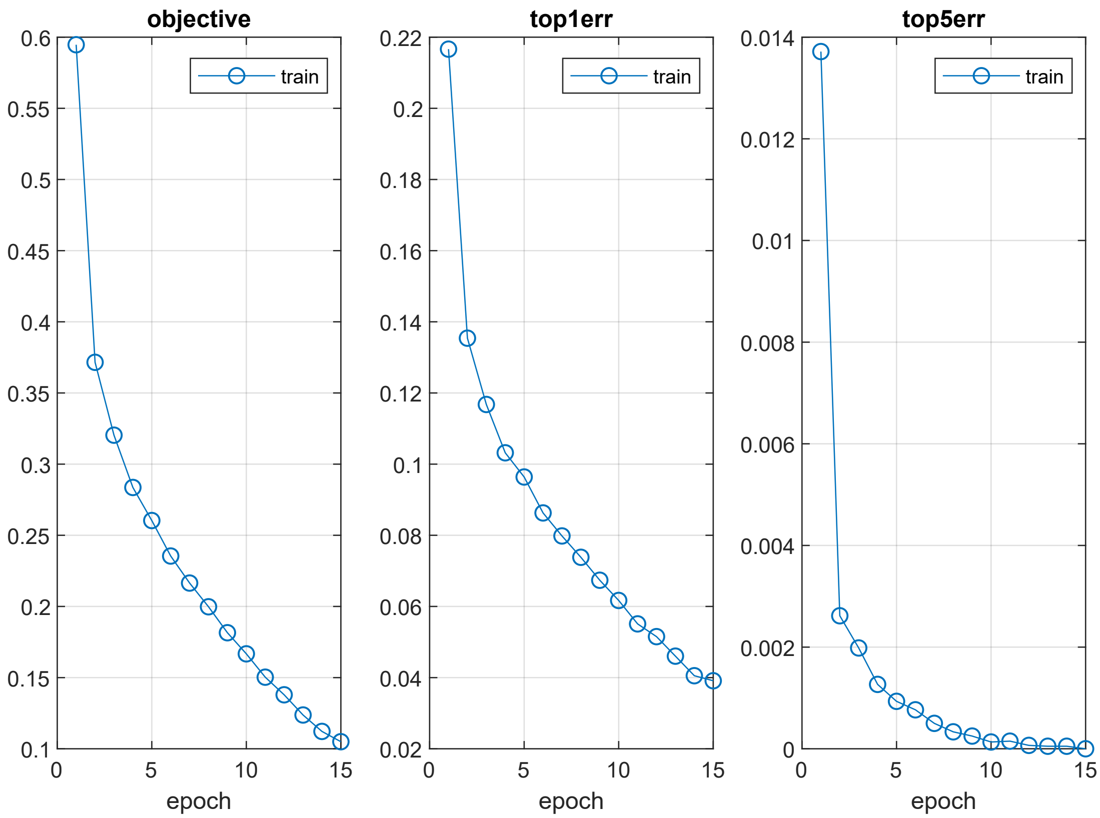

Figure 3: The descent rate of error in classifying the training data

3 Summary

Table 1 shows the classification error of the different methods utilized in project 1 and 2. In Project 1, the most accurate method of classification was to use principal component analysis to reduce the conventionality of the state vectors to 50 and then to apply Nearest-Neighbor (NN) classification to the transformed data.

In this project, both Support Vector Machines (SVM) and Convolutional Neural Networks (CNN) were superior in performance to all the methods used in project 1. CNN proves to be the best classifier. This can be attributed to the increased complexity of the classifier, large number of empirical weights that can be tuned in training, and non-linearity introduced by the ReLU operator.

Table 1: Summary of classification results

Method	Percentage Error
SVM (RBF Kernel)	12%
CNN	9.39%
Bayes	35%
NN	15%
PCA, Bayes (50)	20.5%
PCA, NN (150)	14.7%
LDA, Bayes	25.4%
LDA, NN	28%

Appendix

SVM

PCA

```
%% Reduce the dimension of the FMNIST data using PCA
clear; clc; close all;
% Girguis Sedky
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Load data
% Prototypes
images_train = loadMNISTImages('G:\My Drive\Classes&Books\StatisticalPatternRecognition\Project\Project2\Data\train-images-idx3-ubyte');
labels_train = loadMNISTLabels('G:\My Drive\Classes&Books\StatisticalPatternRecognition\Project\Project2\Data\train-labels-idx1-ubyte');
% Test data
images_test = loadMNISTImages('G:\My Drive\Classes&Books\StatisticalPatternRecognition\Project\Project2\Data\t10k-images-idx3-ubyte');
labels_test = loadMNISTLabels('G:\My Drive\Classes&Books\StatisticalPatternRecognition\Project\Project2\Data\t10k-labels-idx1-ubyte');

%% PCA
% Find the prinicipal components
coeff = pca(images_train');
% Number of dimensions kept
m = 50;

%% reduce the dimensionality of the system to m
images_train = coeff(:,1:m)'*images_train;
images_test = coeff(:,1:m)'*images_test;

%% Save the data
save('Data_PCA.mat','images_train','labels_train','images_test','labels_test');
```

Training script

```
%% Train SVM on reduced dimension FMNIST
clear; clc; close all;
% Girguis Sedky
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Load the reduced data file
load('Data_PCA.mat');

%% Train SVM using different kernels
% linear kernel
model_linear = svmtrain(labels_train, images_train', '-t 0');
% polynomial kernel
model_poly = svmtrain(labels_train, images_train', '-t 1');
% radial basis function kernel
model_rbf = svmtrain(labels_train, images_train', '-t 2');

%% Save the model
save('SVM_Model.mat','model_linear','model_poly','model_rbf');
```

Testing script

```
%% Test SVM trained model on reduced dimension FMNIST test data
clear; clc; close all;
% Girguis Sedky
% %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%% Load the reduced data file
load('Data_PCA.mat');
load('SVM_Model.mat');

%% Run SVM on test data
[predicted_label, accuracy_linear, dec_values] = svmpredict(labels_test, images_test', model_linear);
[predicted_label, accuracy_poly, dec_values] = svmpredict(labels_test, images_test', model_poly);
[predicted_label, accuracy_rbf, dec_values] = svmpredict(labels_test, images_test', model_rbf);

%% Save the accuracy of the SVM with different kernels
Performance = [accuracy_linear(1),accuracy_poly(1),accuracy_rbf(1)];
save('Performance.mat','Performance')
```

CNN

Neural network initialization

```
% Create a LeNet
function net = initialize_FMNIST_CNN()

f=1/100 ;
net.layers = {} ;
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,1,20), zeros(1, 20)}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [2 2], ...
                           'stride', 2, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(5,5,20,50),zeros(1,50)}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'pool', ...
                           'method', 'max', ...
                           'pool', [2 2], ...
                           'stride', 2, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(4,4,50,500),  zeros(1,500)}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'relu') ;
net.layers{end+1} = struct('type', 'conv', ...
                           'weights', {{f*randn(1,1,500,10), zeros(1,10)}}, ...
                           'stride', 1, ...
                           'pad', 0) ;
net.layers{end+1} = struct('type', 'softmaxloss') ;

net = vl_simplenn_tidy(net) ;
```

Main training and testing script

```
% train and test FMNIST data via deep learning
% Girguis Sedky
function CNN_FMNIST_Script(varargin)
setup;
%%
% ---------------------------------------------------------------------------
% Part 4.1: prepare the data
% ---------------------------------------------------------------------------
%%
% Load training data set
% Prototypes
images_train = loadMNISTImages('G:\My Drive\Classes&Books\StatisticalPatternRecognition\Project\Project2\Data\train-images-idx3-ubyte');
labels_train = loadMNISTLabels('G:\My Drive\Classes&Books\StatisticalPatternRecognition\Project\Project2\Data\train-labels-idx1-ubyte');
labels_train=labels_train+1;
% reshape data sp that you get a picture array
images_train = reshape(images_train, [28 28 60000]);
%%
% ---------------------------------------------------------------------------
% Part 4.2: initialize a CNN architecture
% ---------------------------------------------------------------------------
net = initialize_FMNIST_CNN() ;
vl_simplenn_display(net)

%%
% ---------------------------------------------------------------------------
% Part 4.3: train and evaluate the CNN
% ---------------------------------------------------------------------------
% Set the options for the stochastic grdaient descent back propagato
trainOpts.batchSize = 100 ;     % The number of picture that goes into
```

```
    one run of the descent
trainOpts.numEpochs = 15 ;      % Nummber of times the gradient procedure runs through all the data
trainOpts.continue = false ;     % If it were stopped, the descent continues where its left off
trainOpts.gpus = [] ;            % No GPU to be used
trainOpts.learningRate = 0.001 ; % Learning rate
trainOpts.expDir = 'NNData' ;     % directory where the options are saved
trainOpts = vl_argparse(trainOpts, varargin);

% Subtract out the mean of the image
im_mean = mean(images_train(:)) ;
images_train = images_train - im_mean ;

% Create the imdb structure that is apparently needed to run this
imdb.images.data = images_train;
imdb.images.label = labels_train';
imdb.images.id = [1:1:length(imdb.images.label)];
imdb.images.set = 1*ones(1,length(imdb.images.label));
% Call training function in MatConvNet
[net,info] = cnn_train(net, imdb, @getBatch, trainOpts);


% Save the result for later use
net.layers(end) = [] ;
net.imageMean = im_mean ;
save('NNData/CNN.mat', '-struct', 'net') ;

%%
% ------------------------------------------------------------------------
% Part 4.5: apply the model
% ------------------------------------------------------------------------

clear;
% Load the CNN learned before
net = load('NNData/CNN.mat') ;

% Load test data set
% Prototypes
images_test = loadMNISTImages('G:\My Drive\Classes&Books\StatisticalPatternRecognition\Project\Project2\Data\t10k-images-idx3-ubyte');
labels_test = loadMNISTLabels('G:\My Drive\Classes&Books\
```

```
    StatisticalPatternRecognition\Project\Project2\Data\t10k-labels-
    idx1-ubyte');
labels_test=labels_test+1;
% reshape data sp that you get a picture array
images_test = reshape(images_test, [28 28 10000]);

% subtract mean and ,multiply by that weird factor
images_test = 256 * (images_test- net.imageMean) ;

% Image to be tested
for ii=1:length(images_test)

% Apply the CNN to the test image
res = vl_simplenn(net, images_test(:,:,ii)) ;

%
scores = squeeze(gather(res(end).x)) ;
[~, best] = max(scores);
labels_est(ii) = best;
end

labels_est = labels_est';
% Test accuracy of classifying algorithm
a = find(labels_est==labels_test);
Performance = length(a)*100/length(labels_test)
save('Perfomance.mat','Performance');
% ---------------------------------------------------------------

function [im, labels] = getBatch(imdb, batch)
%% Custo, function to create the batches that go into the descent
    procedure
% ---------------------------------------------------------------
im = imdb.images.data(:,:,batch) ;
im = 256 * reshape(im, 28, 28, 1, []) ;
labels = imdb.images.label(1,batch) ;
```