USING BAYES ERROR RATE ESTIMATION TO ANALYZE FEATURE SPACES

A Thesis

Submitted to the Faculty

in partial fulfillment of the requirements for the

degree of

Bachelor of Arts

in

Mathematics

by

Sriram Bapatla

DARTMOUTH COLLEGE

Hanover, New Hampshire

June 2020

Abstract

Whether they be neural networks, auto-encoders, or other architectures, classifiers are commonly assessed through confusion matrices and associated statistics, or visualizations of feature spaces like t-SNE plots. These metrics method depend on the test data used to assess the performance of the network, and different sets of test data will yield different results. We need a more robust method of measuring the inherent error in a classification problem. To address this issue, this investigation studies and applies a non-parametric estimate of the Bayes Error Rate (BER) specifically to problems currently of interest to the United States Air Force related to target classification. The BER is the inherent probability of misclassification given a classifier and a data set. Its bounds are available in closed form through a specific divergence measure, but the divergence itself must be estimated. We apply two non-parametric test statistics that estimate this divergence– the Friedman-Rafsky and cross-match statistics– to bound the BER in a convolutional neural network. The first part of this study establishes the need the BER as a performance evaluation tool. The second part compares the two test statistics. The third part applies the BER estimate to an AF problem. Given that it is often difficult and expensive to survey real targets and generate SAR images, 3-D CAD (Computer aided design) models are frequently used to generate synthetic SAR images for training and testing classifiers. We use BER estimation to examine if reducing the fidelity of training and testing images affects the separability of data, and if high-fidelity images are separable when passed through networks trained on low-fidelity images. The results indicate that low-fidelity images exhibit high separability, and that networks trained on low-fidelity images perform well even on high-fidelity test data.

Preface

I'd like to thank all of my advisors in this research endeavor. Dr. Gelb, (the newly) Dr. Churchill, Mr. Zelnio, Dr. Scarnati, and Dr. Paulson: your input helped me discover new research questions, discontinue others, above all inspired me to keep experimenting. And to my friends and family, thank you so much for your support.

Contents

Chapter 1

Introduction

Neural networks currently appear in almost every facet of our daily lives. In the medical field, machine learning (ML) algorithms use neural networks to predict health conditions and assimilate different health metrics. In finance, ML has produced some of the most successful market prediction tools. Digit recognition, music recommendations, target recognition— the breadth of applications of neural networks is really quite hard to overstate. Thus, the question of a neural network's accuracy, namely it's ability to accurately predict the correct class of an input, is of the utmost importance.

The low-resolution view of an ML algorithm consists of three components and two routines. The three components in its structure consist of a set of inputs, a function, and a set of outputs. The two routines are the training and testing phases. *Training* consists of finding the optimal function for the given data. In the simple case of linear regression, this function is the set of weights and biases that minimize the mean squared error cost function. *Testing* is when the model is validated with data the network has not yet seen. This is an important step that addresses the issue of over-fitting. A good model should generalize well on test data.

For example, imagine a network trained to distinguish between the digits 0 and 1. This is a simple example of a classifier. The testing routine will pass in images of 0's and 1's that were not present in the training data to see how accurate the network is in its predictions. Once the test images are passed through the network, they reach the output layer and are then classified. The accuracy of classifier-determined classes against the true classes is traditionally presented in a **confusion matrix**, which is a matrix of true and false positives of the test data. From here, an accuracy metric is derived by dividing the number of correct classifications by the size of the test data. This is currently the main methodology for evaluating a network. Another evaluative tool is the visualization of the feature space. The **feature space** is the penultimate layer of the neural network. Although this space is usually high-dimensional, different tools can project a collection of samples in this space onto 2-D and 3-D spaces. The issue with both of these evaluation processes, however, is that different sets of test data will yield different results.

Section 1.1 Purpose

The purpose of this thesis is to explore a different approach to quantifying a classifier's performance. Instead of one estimate of accuracy that varies with a given test set, it describes a method of finding more general bounds for a trained neural network's performance. By applying the methodology presented in this thesis, a person using a neural network and a specific training data set can estimate what the best- and worst-case accuracy of the classifier will be.

This thesis will discuss the results of three experiments that convey the importance of the Bayes Error Rate (BER), which will be described in detail in Chapter 3, to improve the robustness of the accuracy measurements. The first experiment establishes the need for the BER by examining the variance of the traditional accuracy metric. The second experiment examines how the BER estimate converges. It also compares two different methods of computing the BER. Finally, the third experiment applies the BER estimate to the prototypical example of target classification for the

United States Air Force (AF). In this case, generating data for target classification is expensive, both financially and temporally. To work around the expenses of gathering real images of targets, researchers are exploring the use of simulated data, whereby computer algorithms simulate real-world imaging processes. While these methods drastically reduce financial costs of generating data, they still operate with inefficiencies. Digital models of targets like tanks and troop carriers are painstakingly crafted and edited, and many resources are currently deployed to try to perfect these models down to the smallest bolt on a door,[24]. The running hypothesis is that these minute details have important consequences on the synthetic images produced after imaging algorithms run on the digital models. This thesis reaches a different finding – the BER estimation demonstrates that low quality digital models might be able produce classifiable images.

The BER is an instrumental component of the AF application because it allows us to measure the *separability* of a feature space. Firstly, it offers a quantitative method to evaluate data sets that is more reliable than a confusion matrix or a feature space visualization. Rather than offering an accuracy measurement, it estimates bounds of irreducible error in the neural network, given a set of test data. Furthermore, it provides a structural understanding of the feature space without requiring projection onto a lower-dimensional space. Thus, with respect to the AF application, we can use the BER to evaluate the performance of data at different fidelities.

The rest of this thesis is structured as follows. Chapter 2 presents a description of how convolutional neural networks are constructed, trained, and tested. Chapter 3 defines the BER and discusses two methodologies of estimating its upper and lower bounds from a data sample. Chapter 4 then presents the experimental setup for the thesis, as well as the justification for three different experiments that we will conduct. Finally, Chapters 5 and 6 discuss the results of the experiments and look forward to further applications of the presented research.

Chapter 2

Convolutional Neural Networks

This chapter presents a more thorough description of neural networks, specifically Convolutional Neural Networks (CNNs). Due to their spatial invariance, which will be discussed later in this chapter, CNNs specifically cater to image classification. There is an abundance of literature dedicated to the study of these structures, see e.g. [8, 23, 10]. These papers also describe some of the many applications of CNNs.

We begin by discussing the structure of a neural network and and its training procedure, which is more commonly referred to as the "learning" component of machine learning. Throughout this description, one must remember the underlying picture of an ML algorithm. It is simply the optimization of a nonlinear function, albeit an immensely complicated one. We conclude with a discussion on how other parameters may affect CNN training.

Section 2.1 Structure

Before describing the structure of a CNN, it is helpful to introduce the concept of a fully-connected neural network.

2.1.1. Artificial Neural Networks

The fundamental component of an Artificial Neural Network (ANN) is the neuron. Each neuron has an associated "activation" when presented an input, which can be mathematically represented by a scalar.

Neurons are organized in structures called *layers*. The first layer is the input layer, which simply consists of a concatenated input. For instance, a grayscale 24×24 pixel image can be converted to a 576×1 vector with each component ranging from 0 to 1. This vector would then represent the input layer.

For an input vector $\vec{x} \in \mathbb{R}^d$, the ANN can then be formalized as a function:

$$\vec{y} = F(\vec{x}), \text{ where } F(\vec{x}) = f_l(f_{l-1}(...f_2(f_1(\vec{x})))).$$

Here, each f_i represents the vector function

$$f_i(\vec{z}) := g_i(W_i\vec{z} + b_i),$$

where i identifies the layer, with the 0th layer being the concatenated input. The W_i represents the matrix of learned weights for each layer and the b_i represents a layer's learned bias. The quantity within the parenthesis is a simple linear transformation. The g_i is where non-linearity comes into play, and it can be chosen to be any function, but common choices are the sigmoid, rectified linear unit (ReLu), and tanh functions, given by

$$sigmoid(z) = \frac{1}{1+e^{-z}},$$

$$relu(z) = \begin{cases} 0, & z \leq 0 \\ z, & 0 < z, \end{cases}$$

and

$$tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}},$$

respectively.

The final hidden layer, what we refer to as the feature space, is an n-dimensional space. Each input is compressed into this space. This is followed by the output layer, which is an m-dimensional space where m is the number of classes in the problem. This output layer uses a softmax function to standardize the neuron activations so that they sum to 1. Because of this step, the network outputs a set of probabilities that the input belongs to each of the m classes. The softmax function for an m dimensional input vector $\vec{z}$ with components z_k is given by

$$\alpha(\vec{z}) = \frac{e^{z_i}}{\sum_{j=1}^{m} e^{z_j}} \tag{2.1}$$

Here z_n is the n^{th} component of $\vec{z}$. Figure 2.1 presents a visual schema of a feed-forward ANN.

2.1.2. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are designed to extract features from images. Computers store images as stacks of two-dimensional matrices where elements correspond to pixel intensity at their spatial locations within the image. This thesis only focuses on grayscale images. The task of extracting features from an image requires special precautions due to the fact that certain features might occur at several distinct regions within a given image. Hence the network requires feature extraction units that are spatially invariant. An important difference to note here between the ANN described previously and the CNN is that the latter preserves the input in its original form, without flattening it to a column vector.

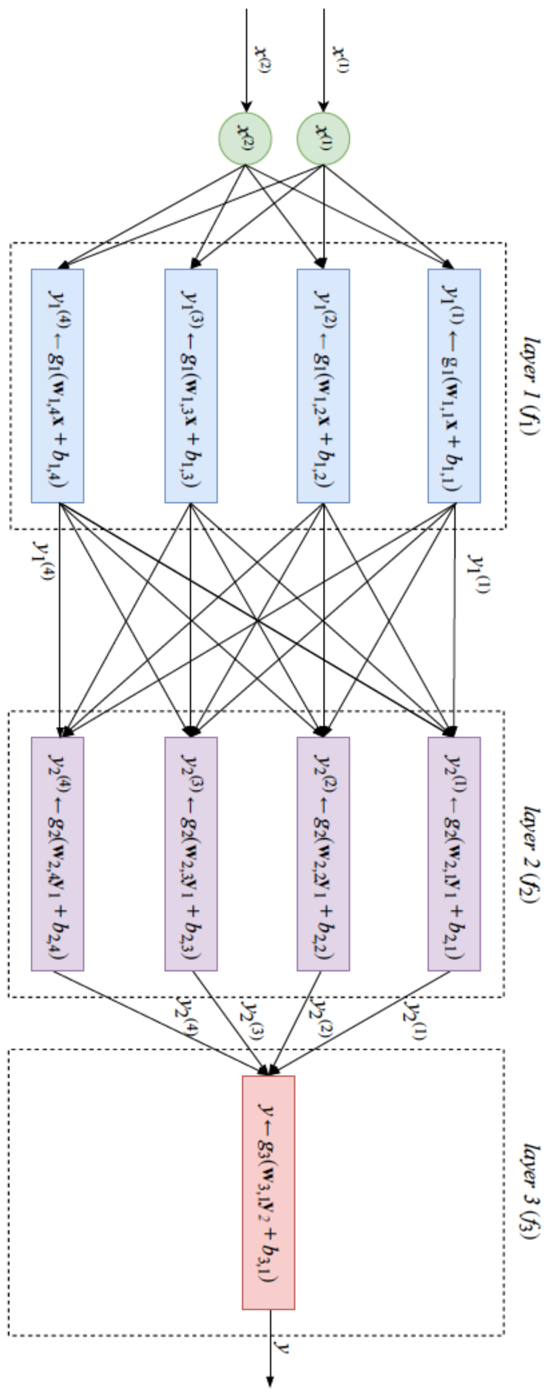

$x^{(1)}$
$x^{(2)}$
layer 1 (f_1)
$y_1^{(1)} \leftarrow g_1(\mathbf{w}_{1,1}\mathbf{x} + b_{1,1})$
$y_1^{(2)} \leftarrow g_1(\mathbf{w}_{1,2}\mathbf{x} + b_{1,2})$
$y_1^{(3)} \leftarrow g_1(\mathbf{w}_{1,3}\mathbf{x} + b_{1,3})$
$y_1^{(4)} \leftarrow g_1(\mathbf{w}_{1,4}\mathbf{x} + b_{1,4})$
$y_1^{(1)}$
$y_1^{(4)}$
layer 2 (f_2)
$y_2^{(1)} \leftarrow g_2(\mathbf{w}_{2,1}\mathbf{y}_1 + b_{2,1})$
$y_2^{(2)} \leftarrow g_2(\mathbf{w}_{2,2}\mathbf{y}_1 + b_{2,2})$
$y_2^{(3)} \leftarrow g_2(\mathbf{w}_{2,3}\mathbf{y}_1 + b_{2,3})$
$y_2^{(4)} \leftarrow g_2(\mathbf{w}_{2,4}\mathbf{y}_1 + b_{2,4})$
$y_2^{(1)}$
$y_2^{(2)}$
$y_2^{(3)}$
$y_2^{(4)}$
layer 3 (f_3)
$y \leftarrow g_3(\mathbf{w}_{3,1}\mathbf{y}_2 + b_{3,1})$
y

Figure 2.1: A description of a feed-forward neural network, taken from [3]

Before discussing the convolution process, one must define the **Hadamard product** of two matrices. Given two matrices A and B of the same dimension $m \times n$, the

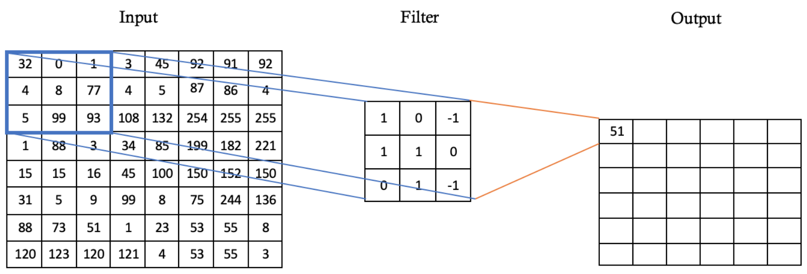


Figure 2.2: One of 36 calculations as the filter passes over and is convolved with the input image

Hadamard product is the element-wise product of each matrix:

$$A \circ B = a_{ij} * b_{ij} = \begin{pmatrix} a_{11} \cdot b_{11} & \cdots & a_{1n} \cdot b_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} \cdot b_{m1} & \cdots & a_{mn} \cdot b_{mn} \end{pmatrix}$$

The convolution of an image involves taking a filter, represented by a matrix of size $i \times i$, and sliding it across the entire image. At each stop, the Hadamard product is taken between the filter and the subset of the input of the same size as the filter. Then, the scalar sum of all the entries in the Hadamard product is outputted. The process repeats until the filter passes over the entire image. This investigation only considers square images and square filters, although this is not a fixed requirement.

Mathematically stated, for an $n \times n$ input image A with only one channel, and a filter F with size $i \times i$, the output of the convolution operation is defined as

$$C(j,k) = \vec{1}^{\mathbf{T}}(A_{j:j+i-1;k:k+i-1} \circ F)\vec{1}, \quad j,k \in [1, n-i+1], \tag{2.2}$$

where $\vec{1}$ is an i-dimensional column vector with only ones as entries. Figure 2.2 provides a simple example for the process of convolving an image with a filter.

Note that this is the simplest rendition of a convolution, and there are two additional parameters that slightly complicate the process. The first addition is *padding*,

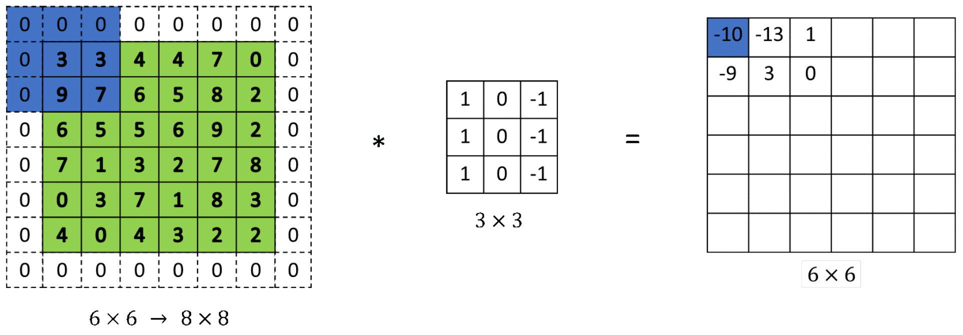


Figure 2.3: Padding an image can preserve its size after convolution

a process by which the image is surrounded by one layer of zeros. One of the main reasons architects choose to use padding is to conserve the original shape of the image. Figure 2.3 shows how this is done. The second addition is *stride size*. In Equation 2.2, it is assumed that the filter slides across the image one column or row at a time. This need not be the case. A larger stride size can increase the efficiency of the optimization by decreasing the number of products and sums in the convolution process. We are now ready to address the question: how do these convolutions fit together? To answer, we need yet another tool in our arsenal: the *max-pool* function. Earlier in this section, it was mentioned that CNNs look for translational invariance. The max-pool function does exactly this. For an $n \times n$ matrix A and a scalar s defining the size of the filter, the max-pool function outputs a matrix M with entries defined as

$$M(j,k) = max(A_{j:j+s-1;k:k+s-1}), \text{ with } j,k \in [1, n-j+1]$$

Essentially all this process does is pass through an image and take out the largest value in an $s \times s$ region of the image.

The process of convolving and pooling might seem haphazard, but they represent very intuitive processes for compressing an image. Filters look for important features in the image. Some look for edges, others denoise, and still others perform meaningful

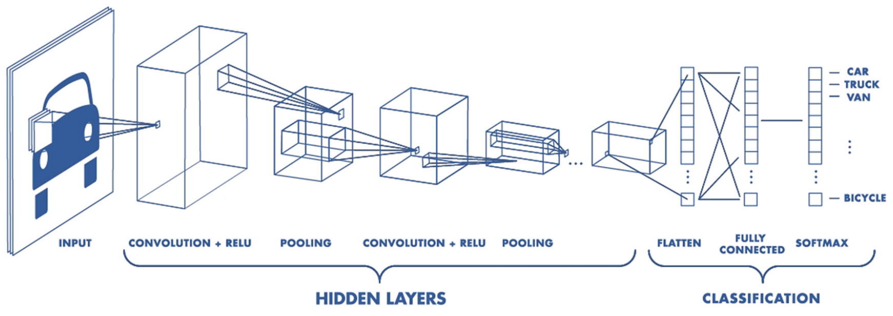


Figure 2.4: A sample CNN architecture [17]

operations that are unintelligible to an observer. The purpose of the filter is to extract and compress important information from an image. The pooling process ensures invariance because it is insensitive to the position of the highest activation in a certain block. Important information is preserved regardless of its location in the space.

A CNN, then, can be constructed by putting together these two tools. A convolutional layer is followed by a pooling layer and this is repeated for a certain number of iterations. Afterwards, the network outputs a certain matrix that contains compressed information about the input matrix. The matrix is then usually flattened into a column vector and then passed through a certain number of fully-connected layers, as discussed in the previous section. Figure 2.4 displays the steps of this process.

Section 2.2 Training

The training of an ANN is a complex optimization procedure. The two main components of this procedure are loss calculation and back-propagation.

2.2.1. Loss Calculation

Once a piece of information has propagated through a neural network it reaches its final layer, which consists of m neurons for an m-class classification problem, each with a particular degree of activation. As mentioned in the previous section, the activations are traditionally normalized in this layer to sum to one. Therefore, these activations can be seen as representing the probabilities that the input belongs to each of the possible classes.

It is at this point where loss is calculated using a particular loss function. A commonly used loss function is mean squared error (MSE), which is mostly applicable to normally distributed data. The MSE for N samples of test data, where $\hat{Y_i}$ is the predicted class of sample i and Y_i is the sample's true class is defined as

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(\hat{Y_i} - Y_i)^2.$$

However, in most classification applications, researchers implement the cross-entropy loss function, which is given for an m-class problem by

$$H(p, q) = -\sum_{i=1}^{m} p_i(x) log(q_i(x)). \tag{2.3}$$

Here $p(x)$ is the true distribution and q(x) is the predicted distribution.

2.2.2. Backpropagation

Once the cost has been calculated via a loss function, the network must adjust its vast collection of weights to yield lower cost values in following iterations. The driver behind this process is gradient descent, an iterative process looking to minimize the loss function. Once a loss function has been chosen, the cost associated with a output vector can be calculated and designated $C(\theta)$, where θ is the set of weights and biases in the network. Backpropagation uses partial derivatives to attribute the proportion of the cost for which each individual weight and bias is responsible. Because the each neuron in the output layer is chained to all of the previous neurons, the chain rule can be employed to calculate the partial derivatives $\frac{\partial C}{\partial w_i}$ and $\frac{\partial C}{\partial b_i}$, where the index i

identifies the layer of the neural network.

Section 2.3

Gradient Descent

If backpropagation is the wood for the house, gradient descent is its construction. Gradient descent is a part of nearly every optimization problem, from least-squares regression to much more complicated structures like neural networks. For parameters θ and a cost function $C(\theta)$, gradient descent is described as the iterative procedure for recovering θ as

$$\theta_{i+1} = \theta_i - \eta \nabla C(\theta_i) \tag{2.4}$$

from an initial input θ_0. Here η represents the learning rate, which will be discussed further in Section 2.4.

An important note about gradient descent is that it depends on a convex loss function. Both the MSE and cross-entropy loss functions are convex, as we want them to have a global minimum that can be reached by following the gradients along the surface.

The iterative gradient descent process occurs in the following steps:

1. Data is passed through the network.
2. The cost associated with the data is calculated.
3. Backpropagation calculates the portion of the cost that is ascribable to each weight and bias in the network.
4. Gradient descent occurs using these partial derivatives.
5. The first four steps are repeated until a convergence criterion is met or a fixed number of iterations is reached.

Gradient descent itself is a deep topic for research. There are many different algorithms that are used for neural networks, but the main ones are stochastic, batch, and mini-batch gradient descents [19]. Stochastic gradient descent updates weights after each instance of the data, and in this experimental setup it entails updating the network after each image is passed through the network. As this yields a noisy convergence path, it is sometimes better to use the averages of $\frac{\partial C}{\partial w_i}$ and $\frac{\partial C}{\partial b_i}$ across many images, or a *batch*. Once all of the batches of data in a given training set have been exhausted, meaning every training image has been seen, one *epoch* passes. This thesis implements mini-batch gradient descent with a batch size of 64 images over many epochs.

Section 2.4 Hyperparameters

There are several parameters of the model that the network architect can choose.

2.4.1. Activation Function

As discussed earlier, there are many activation functions that network architects can use in their neural networks. However the ReLu, or the rectified linear unit function, is often used for a couple of reasons. Firstly, it better-addresses the **vanishing gradient problem**. Due to the fact that the individual weights in the network are updated countless times, sigmoid or tanH functions often yield weights that go to 0. This is a byproduct of their output space, which consists only of the interval $[-1, 1]$. Once a weight is updated to 0, it no longer functions in the network. The ReLu function, on the other hand, outputs on the interval $[0, +\infty]$, lessening the occurrence of vanishing gradients. Another benefit of the ReLu function is that it's more computationally feasible, especially because the activation function operates on a significantly large number of items during a learning process.

2.4.2. Learning Rate and Momentum

As described earlier, there are many ways to modify the gradient descent algorithm, and this is a field that is rich with research, [25, 12, 15]. Many of these modifications address issues of either computational feasibility or convergence pattern. To reiterate, the goal of gradient descent is to find the global minimum of the cost function. Often times the optimization process will get stuck in a local minimum, and depending on ones choice of a learning rate, it can be quite tough to escape this minimum. The learning rate η in (2.4 specifies the magnitude with which the cost function's gradient affects a given weight. Too large a parameter can yield a very shaky convergence, as weights will experience large updates. Too small a learning rate and the function may not be able to escape a local minimum. In practice, various learning rates are tried on different networks to optimize convergence.

Momentum is another tool in the arsenal of gradient descent. Instead of only updating (2.4) based on $\eta\nabla C(\theta_i)$, one solves

$$\theta_{i+1} = \theta_i - \gamma\eta\nabla C(\theta_i),$$

where γ is meant to balance perturbations in the convergence process. With methods like stochastic gradient descent, the convergence is very jumpy. Momentum is used to reign in changes of direction while pushing the pedal on weight updates that have consistently been going in one direction.

This thesis implements the Adagrad method of gradient descent[5] Adagrad further modifies stochastic gradient descent by changing the learning rate for different weights.

2.4.3. Other Parameters

There are a plethora of other hyper-parameters that can be optimized for a neural network. The number of layers in the network, the size of filters in each convolution, strides and pooling methods, and data pre-processing are all variables that will affect the performance of a network. Many of these are chosen *ad hoc*, based on machine learning literature or through trial and error. This all extends beyond the scope of this thesis, which is focused on evaluating classifiers, not optimizing their performance.

Chapter 3

Bayes Error Rate

In Chapter 1 we discussed a neural network as a deterministic function that maps an input to a class. But when quantifying error, it helps to take a Bayesian view of the classifier. This is because the input data can be seen as an instance of a random variable with a prior distribution. The neural network then assigns values to the likelihood a sample was drawn from one of the priors. Finally, the model generates a posterior distribution, i.e. the probability that the instance belongs to each of the classes.

More formally, suppose we are given a series of n inputs $\{\vec{x_i}, y_i\}_{i=1}^{n}$, where each $\vec{x_i}$ is a realization of some random variable X, with $\vec{x_i} \in R^d$ and $y_i \in \{1, ..., m\}$, for an m-class problem. There is a series of probability functions defining the learning process. Assume we have an initial set of priors, $\{c_i\}_{i=1}^{m}$, for each class in the m-class problem, as well as a series of likelihood functions, given as densities conditional on the class the data could take: $p_i(\vec{x}) = P(X = \vec{x}|Y = i)$ for $i \in \{1, 2, ..., m\}$. We combine the prior and likelihood to obtain the posterior $h_i(\vec{x}) = P(Y = i|X = \vec{x}) \propto c_i \cdot p_i(\vec{x})$.

In most neural networks, data is assigned to a class via a maximum a-posteriori (MAP) rule. Specifically, the data is assigned to the class with the largest standardized posterior probability. The Bayes error rate is the complement of this posterior, meaning that it is the probability that a sample is identified as one class, when it

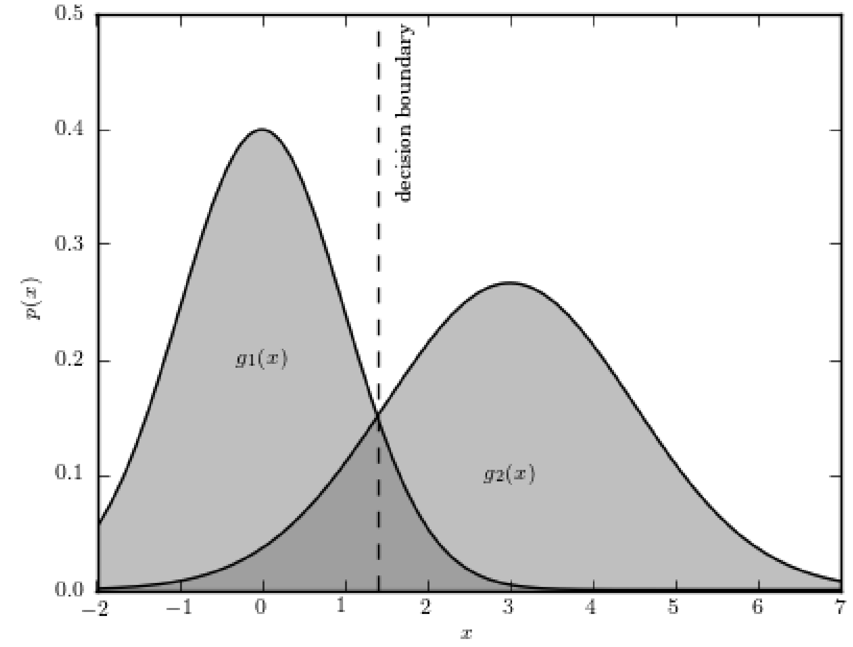


Figure 3.1: Two overlapping posterior distributions. The Bayes Error Region is darkly shaded, and it illustrates the range of x-values where misclassification can occur.

really pertains to a different class. This mathematically described as

$$e^m = 1 - \int \max\{h_1(x), h_2(x), ..., h_m(x)\}dx. \tag{3.1}$$

The integral in (3.1) is taken over the posterior probability space. Figure 3.1 depicts a sample posterior for a 2-class problem, with the Bayes error rate being the area of the shaded region.

Section 3.1

Importance of the BER

The BER is traditionally used to describe the difference between two distributions. This is an important component of hypothesis testing. In the medical field, it can be applied to testing a treatment, with the statistical distance between two distributions highlighting the difference between patient and control groups, [18]. In that study, the distributions are ideally quite distant from the onset. The whole purpose of a neural network is to maximally separate the posterior distributions of different classes so that classification is easy. The BER here is instead used to quantify this degree of separation. The higher the BER, the higher the inherent probability of misclassification. This is also called *irreducible error*, because if the true posterior distributions overlap, even the best classifier can misclassify images. This can be seen from its definition. The true BER describes the **best case performance** of a classifier given a data-set. Thus, in the machine learning sphere, it can serve as a data quality metric. In a system with known prior an posterior distributions, the BER answers the question: *what is the best-case performance of this classifier given the chosen data set?* This is an extremely valuable metric because it is the core quantity that a machine learning architect uses to evaluate her model. The issue with neural networks, however, is that the priors and posterior are not known in closed form, so the BER must be estimated.

Section 3.2
Estimating the BER

There are currently a variety of methods used to estimate and bound the BER, but none works perfectly, and there are trade offs one must consider. The bounds themselves come out of divergence measures called *f-divergences*, which estimate the difference between distributions. Given two discrete probability distributions P and Q defined on a sample space X the f-divergence between the two distributions is given by:

$$D_f(P, Q) = \sum_{x \in \mathcal{X}} Q(x) f(\frac{Q(x)}{P(x)})$$

where f is a positive-valued convex function and $f(1) = 0$, [2]. This metric determines the difference between two distributions over their shared space. One of these divergence functions, called the HP-divergence, can be modified to yield a tight bound on the BER, [22]. [21] defines the HP-divergence for a 2-class classification:

Definition 3.1. Given the prior distributions of the classes 0 and 1 denoted c_0 and c_1, as well as the conditional probabilities that a sample $\vec{x}$ was drawn from the priors, denoted $p_0(\vec{x})$ and $p_1(\vec{x})$, the HP-divergence between $p_0(\vec{x})$ and $p_1(\vec{x})$ is :

$$D(p_0(\vec{x}), p_1(\vec{x})) = \frac{1}{4c_0c_1}\left[\int_{\mathcal{R}^d} \frac{(c_0p_0(\vec{x}) - c_1p_1(\vec{x}))^2}{c_0p_0(\vec{x}) + c_1p_1(\vec{x})} \mathrm{d}\vec{x} - (c_0 - c_1)^2\right]. \tag{3.2}$$

The HP-divergence is a function of both the prior distributions of the classes and the conditional distributions. By calculating the HP-divergence, one can obtain tight BER bounds, but the issue is that the prior and conditional distributions are unknown. One solution is to assume the prior and conditional distributions are Gaussian, exponential, or other known distributions. This is a **parametric** method of estimating the BER, as one must parametrize these chosen distributions. The issue with this approach is that it requires making assumptions about unknown distributions.

Section 3.3

Non-parametric Estimation of BER

Non-parametric methods can avoid some of the bias that occurs if one assumes data follows a certain distribution, but they are also prone to having issues. Many non-parametric methods rely on probability density estimation, which can be prone to errors and does not scale very well computationally. Some alternatives to these "plug-in" estimators are graph-based estimators. This thesis implements two specific algorithms, proposed in [21, 22], which are described below.

3.3.1. The Friendman-Rafsky Statistic

This methodology uses a certain test-statistic to estimate the bounds for the BER, which are derived for the generalized HP (GHP)-divergence, first proposed in [22]. The GHP-divergence extends Definition 3.2 beyond its two-distribution input. It defines the divergence between any finite number of distributions, meaning it is quite applicable to classifiers that must classify more than two types of inputs. [22] uses an alternative form of the GHP-divergence called the *GHP-integral*, which is defined in the following manner:

Definition 3.2. Given a set of m priors $\{c_k\}_{k=1}^m$ and conditional distributions $\{p_k(\vec{x})\}_{k=1}^m$ based on a sample $\vec{x}$, first define $h(x) = \sum_{i=1}^m h_i(\vec{x})$, or the marginal posterior distribution of our random variable X. The GHP-integral between two distributions is defined as over their common sample space S^{ij} is:

$$GHP(p_i(\vec{x}), p_j(\vec{x})) = \delta_{ij} = \int_{S^{ij}} \frac{p_i(\vec{x})p_j(\vec{x})}{h(x)}$$

This integral form is related to the divergence between the two distributions in the following manner:

$$D(p_i(\vec{x}), p_j(\vec{x})) = 1 - (c_i + c_j)GHP(p_i(\vec{x}), p_j(\vec{x}))$$

Equation (3.2) can be used to bound the BER, but the issue is that the conditional distributions as well as the marginal distribution of our random variable are unknown. [22] uses the Friedman-Rafsky (FR) test statistic to estimate these bounds through the following procedure:

1. Consider an m-class classification problem. Begin with n instances of data-class pairs X $= \{x_i, y_i\}_{i=1}^n$. Then split them according to their class: $X^k =$

$\{x_i, y_i\}_{i=1,y=k}^{n}$.

2. Construct a graph G consisting of all possible pairs of points. Assign weights E to each of the edges in the graph based on the Euclidean distance between each pair. Construct a minimum spanning tree through the set of points $X^0 \cup X^2 ... \cup X^{m-1}$. A spanning tree is a tree with the minimal amount of edges that includes all the vertices of G. A minimum spanning tree is a spanning tree such that the sum of the weights in its edges is no larger than sum of weight of any other spanning tree,[20].

3. Determine the value of the FR statistic $\Re(X)_{ij}$ as such: count the number of dichotomous edges in the tree between class i and class j, meaning the number of edges that connect two distinct classes.

Once $\Re(X)_{ij}$ is calculated for each pairwise connection between classes in a set of samples X, the BER, indicated for the m-class problem as ϵ^m, can be calculated using the following theorem:

Theorem 3.3. *Let n be the size of the sample in consideration and n_i be the number of samples belonging to class i. If c_i is the prior probability distribution of class i, then as $n_i \to \infty$ and $\frac{n_i}{n} \to c_i$,*

$$\frac{\Re(X)_{ij}}{2n} \xrightarrow{a.s.} \delta_{ij}, \tag{3.3}$$

where

$$\epsilon^m \leq 2 \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \delta_{ij},$$

and

$$\epsilon^m \geq \frac{m-1}{m}[1 - (1 - \frac{2m}{m-1} \sum_{i=1}^{m-1} \sum_{j=i+1}^{m} \delta_{ij})].$$

As the sample size n increases, the ratio $\frac{\Re(X)_{ij}}{2n}$ in (3.3) converges to the true value of δ_{ij}. Thus, Theorem 3.3 can be used to computationally estimate the bounds of the

BER based on a set of sample data[22].

In order to produce produce the Euclidean graph of pairwise distances, which is the first component of the procedure, $\frac{n(n+1)}{2}$ calculations must be done, where n is the number of samples. The MST construction implements Kruskal's algorithm in $\mathcal{O}(E \log(E))$ time, where E is the number of edges in the graph, [9]. Since this graph has on the order of n^2 edges, the run-time of the BER calculation is on the order of $\mathcal{O}(n^2 \log(n))$.

One of the issues with this approach is that the variance of the FR-statistic is dimension dependent. For example, if principal component analysis (PCA) is used to reduce the dimension of the original data (while preserving its features), then the statistic, and therefore the BER estimate, would vary to an unknown degree. To remedy this problem we use what is called the *cross-match statistic*, [21], to estimate BER. It is described below.

3.3.2. Cross-Match Statistic

The cross-match statistic, a procedure proposed in [21] avoids the dimension dependent variance problem observed with the FR statistic. It can be explained as follows: Consider a 2-class problem, where $X_N = \{\vec{x_i}, y_i\}_{i=1}^N$ is a collection of N data-class pairs, with $\vec{x_i} \in R^d$ and $y_i \in \{0, 1\}$. Let m be the number of observations belonging to class 1, and n be the number of observations belonging to class 0. Also let $c_0 = P(y = 0)$ and $c_1 = P(y = 1)$. Finally, let p_0 and p_1 represent the conditional probability distributions $p_0 = p(y = 0|\vec{x_i})$ and $p_1 = p(y = 1|\vec{x_i})$. The cross-match statistic is then calculated using the following steps:

1. Construct a symmetric Euclidean distance matrix D, where each element in the matrix $D_{ij} = ||x_i - x_j||$.

2. Construct a graph G using the N data vectors as vertices, edges between the vertices, and the D matrix as weights.

3. Find the minimum complete matching for the graph. A complete matching is a set of edges such that no two edges share a common vertex and every vertex is touched by the matching. The *minimum* complete matching is a complete matching that minimizes the sum of weights between the edges used.

4. Calculate $A(X)$, the number of dichotomous edges in the minimum complete matching. This is the cross-match statistic

This statistic can then be used to calculate the BER bounds. Researchers established that the HP-divergence, a measure of the difference between two distributions, can be used to bound the BER, [22]. Designating the HP-divergence between two densities p_0 and $p_1$as $D(p_0, p_1)$, we can derive the intermediate quantity:

$$u_c(p_0, p_1) = 4c_0c_1D(p_0, p_1) + (c_0 - c_1)^2$$

Using A(X), we can then estimate $D(p_0, p_1)$ using the following theorem, [22]:

Theorem 3.4. *If N is the number of samples in consideration, and m and n are the number samples in each class, then as $m, n \to \infty$, $\frac{m}{N} \to c_0$, and $\frac{n}{N} \to c_1$:*

$$1 - \frac{N}{mn}A(X) \xrightarrow{a.s.} D(p_0, p_1)$$

The quantity $u_c(p_0, p_1)$ can then bound the BER, ϵ, in the following manner:

$$\frac{1}{2} - \frac{1}{2}\sqrt{u_c(p_0, p_1)} \leq \epsilon \leq \frac{1}{2} - \frac{1}{2}u_c(p_0, p_1)$$

Implementing Papadimitriou and Steiglitz's optimal-weighted matching algorithm, [14], a minimum-weighted matching of n samples can be found in $\mathcal{O}(n^3)$ time. A key disadvantage of using this method is that the BER bounds are only established for the 2-class problem.

Chapter 4

Methods

We now present the experimental setup and describe the data sets that will be used. We also introduce the three experiments to be conducted.

Section 4.1

General Structure of Each Experiment

There are two structural components to this endeavor. The first is the classification architecture, the function mapping individual images to a feature space. The second component is the analysis of the feature space which in this thesis is done in MATLAB and **R**[13][16]. The example process can be described by

1. Collate the testing and training data that will be used in the experiment. The MST estimate, as defined in Section 3.3.1, can be done on any m-class classification. In this case we perform a 3-class classification. In the case of the cross-match estimate, as defined in Section 3.3.2 we proceed with a 2-class classification, since it is limited to $m = 2$ classes.

2. The first CNN converts the original 64×64 MSTAR image to a 250×1 feature vector. The second CNN converts the 28×28 MNIST image to a 100×1 feature vector.

3. A set of feature vectors and their respective labels is then collected for BER analysis.

4. In the case of the FR statistic, MATLAB's built-in implementation of Kruskal's algorithm is used to construct a Euclidean minimum-spanning tree. In the case of the cross-match statistic, we use an **R**-based implementation of optimal-weighted matching, [6]. In both cases, the cross-labels are counted and BER bounds are calculated.

4.1.1. Data

This thesis uses two publicly available sets of data. The first is the MNIST data set, which consists of 60,000 training and 10,000 test samples of handwritten digits. Each sample is then converted to an 28×28 png image.The repository can be found at http://yann.lecun.com/exdb/mnist/, [11]. The second data set is the public release component of the Moving and Stationary Target Acquisition and Recognition (MSTAR) project, [1]. It consists of training and test samples of SAR images of three different vehicles, the BTR70, BMP2, and T72 tanks, from several different vantage points (angles). Each sample is read from an executable and converted into a complex image, of which only the magnitude is considered. The samples are then cropped to yield grayscale images each of dimension 64×64. The training set consists of 699 BMP2 images, 234 BTR70 images, and 692 T72 images. The test set consists of 582 BMP2 images, 195 BTR70 images, and 577 T72 images. This is not an ideal training set for a neural network because it is not equally distributed amongst targets. However, since the performance of data sets in a classifier are only compared relative to a control, their absolute performance is not extremely significant.

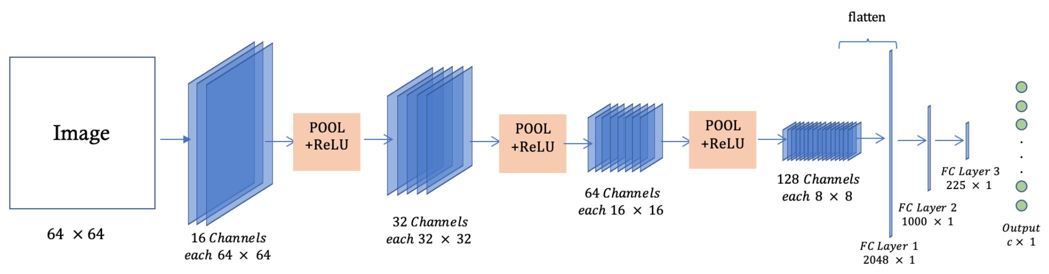


Figure 4.1: A schema of the CNN used for the MSTAR data. It takes 64 × 64 pixel inputs and outputs a 250-dimensional feature vector before its final softmax layer.

4.1.2. CNN Structure

This thesis deploys two neural network models, one for each of the data sets used. Metrics from the data sets are *not* compared with one another. This is to ensure that the differences in architectures do not interact with the results. The MNIST network consists of 3 convolutional layers followed by 4 fully-connected layers.

The MSTAR CNN has an additional convolutional layer and an equal number of fully-connected layers. Figure 4.1 depicts a schema of the MSTAR network. In both networks the learning rate for gradient descent in equation 2.4 is set to 0.01, a batch size of 64 is used, and the cost function is cross-entropy loss, given in Equation 2.3. The number of epochs over which the network is trained varies from experiment to experiment; they are chosen in an *ad hoc* manner based on the convergence of the loss function to a minimum value.

Section 4.2 Three Experiments

The following three experiments are implemented to better understand and apply BER estimation.

4.2.1. Experiment 1: Establishing Need for BER Estimation

Currently, machine learning algorithms are evaluated through test data accuracy. Once an algorithm is trained, it operates on unseen data and its performance describes how well the algorithm generalizes beyond a training set. The issue with this approach is that the accuracy metric will vary depending on the test set. The following sub-experiment helps illustrate this effect. The hypothesis was that the test data would display a non-trivial variance across different samples, and that this distribution would lie within the estimated upper and lower bounds for the BER. The experiment uses the MNIST data set.

In the experiment, a CNN was first trained on the MNIST data for 5 epochs. Then, random subsets of the testing data, which included 2500 images with 250 images from each class, were passed through the trained network. For each of these trials, the accuracy metric was collected. This process was repeated 1000 times on the same trained network, resulting in a 1000-dimensional vector of test data accuracies. Finally, the BER estimate of the test data set was estimated using the MST algorithm over three trials. The maximum upper BER bound and the minimum lower BER bound were then compared to the range of accuracies collected over 1000 trials.

4.2.2. Experiment 2: Studying Behavior of BER

The second series of experiments seek to understand the behavior of the BER estimators proposed earlier in the thesis. "Behavior" is meant to imply two evaluations:

1. The convergence of the estimate over a training process. More specifically, it asks the question: how does the estimate converge over epochs of neural network training?

2. The convergence of the estimate over the hidden layers in the neural network, leading up to the feature layer.

This experiment also compares the two different methodologies for estimating the BER. It has been noted that using the cross-match statistic yields a BER estimate whose variance does not depend on the dimension of the feature space. This sub-experiment wishes to examine this theoretical result.

The experiment begins by training a CNN for a binary classification between the BMP2 and T72 targets in the MSTAR data set. Once the network has been trained, test data is passed through the network and 1351, 250-dimensional latent vectors and their respective class labels are outputted for analysis. From here, we use principal component analysis (PCA) to realign the data along a new 250-dimensional basis. At this point, we can adjust the independent variable, namely the dimension of the latent space, by selecting a d-dimensional subset of principal components. At each increment in dimension, the cross-match and minimum-spanning-tree methodologies are used to calculate BER estimates for 20 subsamples of the data. This allows us to calculate the variance of the estimates in each of the dimension increments, of which are in the range $[2, 152]$.

4.2.3. Experiment 3: Applying BER to AF Problems

Having studied the behavior of the BER in a machine learning setting, we are now able to apply the BER estimate to a current AF problem. In general, training machine learning algorithms is one of the easier components of a ML task. The tougher part is obtaining data. Here, “tough” can mean computational intractability, high monetary cost, or the use of some other expensive resource. The AF is just one subset of the organizations that need to optimize the cost of data collection for the purpose of training algorithms.

This study deals specifically with target-recognition applications of neural networks. In this space, machine learning algorithms are trained to identify and discern different targets, such as buildings and vehicles. In order to collect a data set of training images, an aerial vehicle must be sent to take images of the target, usually through the technique of Synthetic Aperture Radar (SAR). Real-world SAR imaging, while quite accurate, is a costly procedure. In order to get these images, resources must be spent to acquire the target, position the target, and to fly an overhead vehicle for imaging. Besides the resources that must be devoted to this process, there are limits to the feasibility of this approach. There may always be targets that the AF cannot acquire.

To address the concerns of training data acquisition, organizations can opt to instead produce training data through simulation. These methods are varied and diverse, but the essence is that computers can simulate real world imaging techniques, obviating the need for resources other than computational power.

Generating simulated data, and specifically SAR data, is still quite expensive, however. Supercomputer resources must be diverted to process the simulations, and generating high quality images takes both time and money, as the maintenance of supercomputers is costly, [7]. One way to improve this process, then, would be to optimize data generation. A lever we can move is image fidelity, or the quality of images outputted from the simulator.

There are applications where the data must be as crisp as possible and high frequency components must be preserved. The question this experiment seeks to answer is **does image fidelity affect the separability of data in a classification process?** We could not test this hypothesis directly as this would entail using and altering confidential algorithms. Instead, in this experiment we use Gaussian noise as a proxy to affect the fidelity of images. In the rest of the thesis, *high fidelity* refers to the original MSTAR data set and *low fidelity* refers to data that has been blurred by a Gaussian filter.

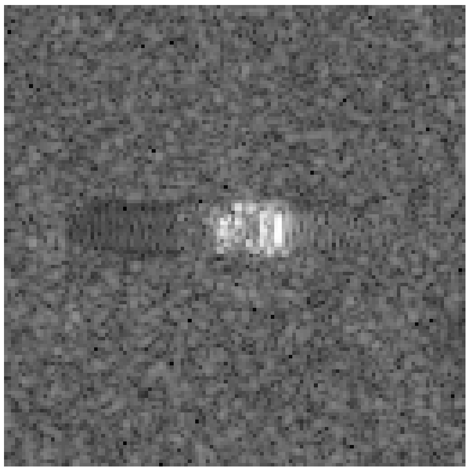
This experiment takes 3 targets, BMP2, BTR70, and T72, from the public release MSTAR data set. Each sample is then passed through a Gaussian blurring filter for the standard deviation values $\sigma \in \{0, 1, 2, 3, 4\}$. A Gaussian blurring filter is an $m \times m$ matrix whose entries approximate the two-dimensional Gaussian distribution. With a mean of zero, the distribution is given by

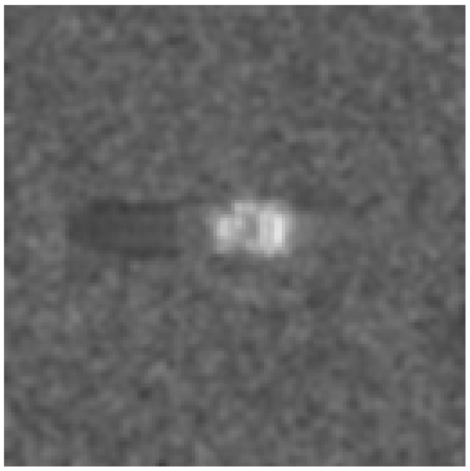
$$f(x, y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2+y^2}{2\sigma^2}}$$

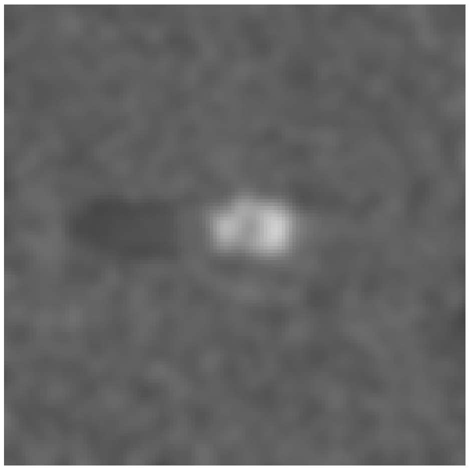
The filter is a discretized and standardized version of the two-dimensional Gaussian distribution, and it can be instantiated based on two parameters: the value of σ, or the standard deviation of the Gaussian, and filter's size, [4]. As the value of σ increases, the smoothing effect increases. In this experiment, the MSTAR train and test data are convolved with Gaussian filters having the σ values 1 through 4. Figure 4.2 shows the same image of a BMP2 tank at different values of σ.

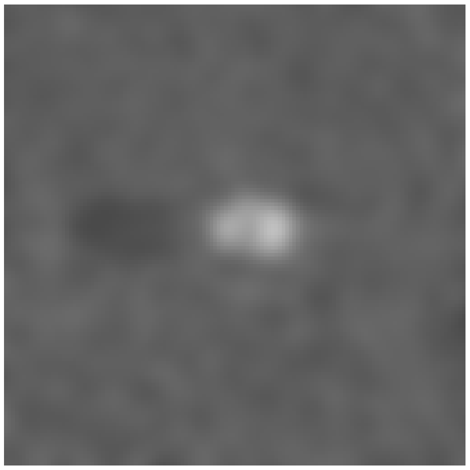
The first tested result was the difference in estimated BER across different values of sigma. This experiment addresses the separability of low-fidelity data and compares it with its high-fidelity counterpart.

The second tested result looked at the implications of training on low-fidelity data and testing on high-fidelity data. This experiment addresses the compatibility of different feature spaces and is the crux of the AF application. If high-fidelity data exhibits a significant degree of separability in lower-fidelity feature spaces, it suggests that the AF can optimize its data generation process by reducing the quality threshold for training images.


(a) $\sigma = 0$


(b) $\sigma = 1$


(c) σ =2


(d) $\sigma = 3$

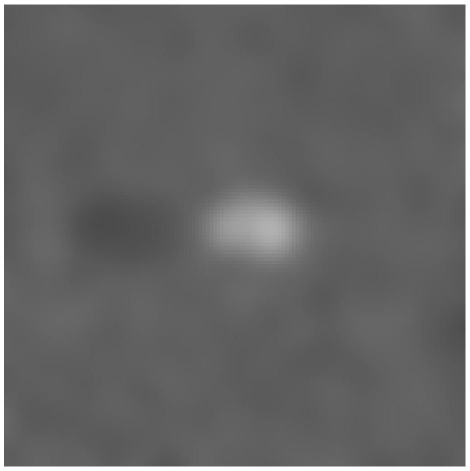

(e) $\sigma = 4$

Figure 4.2: BMP2 at five different fidelities.

Chapter 5

Discussion

Section 5.1

Experiment 1

The results we now present demonstrate that there is a non-trivial variance in the accuracy of test data samples. Over the course of 1000 trials, the mean test data accuracy was 96.76%. However, the accuracies ranged from 95.10% to 98.27% – a 3.17% difference. While this may seem small, a range of uncertainty of 3% can be intolerable in many high importance applications of neural networks, including AF, banking, and medical areas. Figure 5.1 shows the distribution of accuracies across the trials. The two vertical lines delimit the maximum upper BER and minimum lower BER estimates across 3 trials. With a relatively few number of operations, the BER calculations offer a more comprehensive view of the network's best-case and worst-case performance.

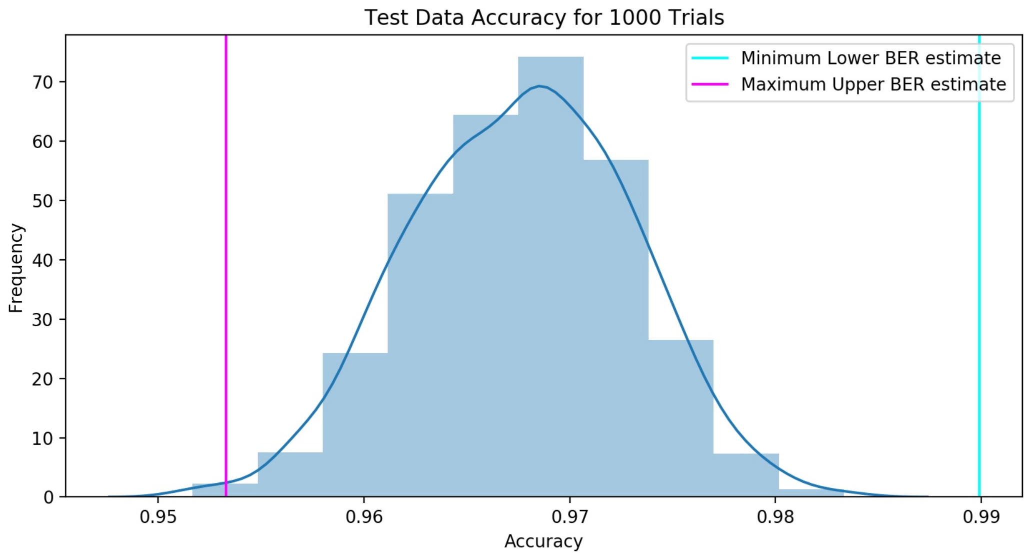


Figure 5.1: Test data accuracies across different trials

Section 5.2

Experiment 2

Figure 5.2 shows the results for Exeriment 2. First note that the BER estimate converges over a learning process. As a reminder, the unit of "time" in this experiment is represented by an epoch. Over the thirty epochs on which the CNN was trained for MSTAR images, the BER estimate decreased until convergence, meaning there is a certain threshold beyond which the BER estimate can be used reliably. The bottom graph of figure 5.2 depicts the convergence of the estimated upper and lower bounds for the BER. The three heat maps depict the number of cross-links (when samples of two different classes are connected in the MST) in the feature space. As the network is trained, the number of cross-links decreases and the samples are more clustered.

The second component of this experiment was to examine how the BER estimate changes within the different layers of a CNN. Figure 5.3 depicts the convergence of upper and lower BER bounds in four different spaces. The top pair of graphs study the second and third convolutional layers in the network, and we observe that they

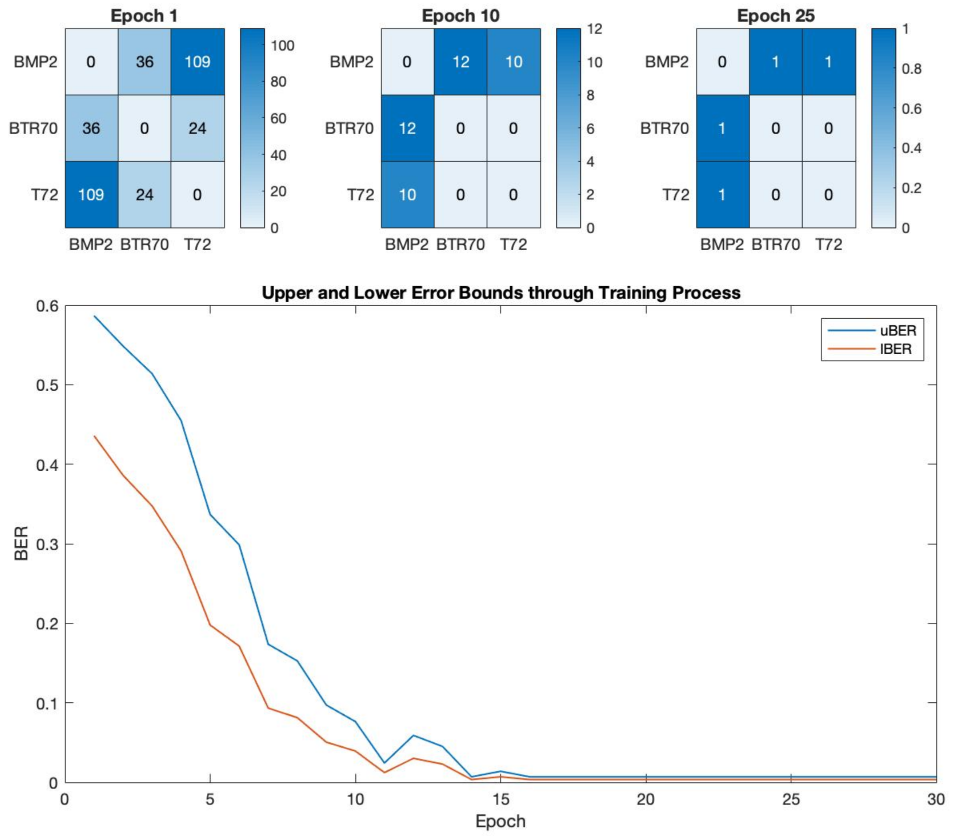


Figure 5.2: Convergence of the BER estimate 3-class problem

do not exhibit a pattern of convergence over the range of epochs. The line in the graph labeled "Mean Cross Distance/Mean Same Distance" charts the average ratio of distances between points of different classes in the MST and distances between points of the same class. We would expect a space conducive to separability to have a high ratio, implying that classes are clustered together within the space, and these clusters are far from other such clusters. This does not seem to be the case in the convolutional layers, supporting the claim that they are not optimal spaces for classification.

On the other hand, the bottom two graphs examine the two fully-connected layers that precede the output layer. In these graphs, it is clear that the BER estimates converge to low values. Moreover, the distance ratio is 8 times as large as the convolutional layers, implying that clustering is taking place. This seems to correspond with the theory behind neural networks. Each layer represents a different space and as the data feeds forward in the network, the spaces are more conducive to separability of different classes.

The final component of this experiment was to compare the cross-match and MST methodologies for estimating BER. The results show that there is a marked difference in the variance of these two measurements. These are encapsulated in Figures 5.4 and 5.5.

We can interpret these results as follows: First, as the sample size increases, the variance of the BER estimates decrease for both the MST and cross-match (CM) methods. This is in accordance with the mathematical theory behind the estimates. They converge to the true BER bounds almost surely as n approaches infinity. Accordingly, the variance decreases as well. However, Figure 5.4 shows that the variance of the CM estimate is consistently larger than the MST estimate.

Figure 5.5 depicts the mean lower and upper BER estimates across different dimensions along with error bars depicting the standard deviations of the mean. It

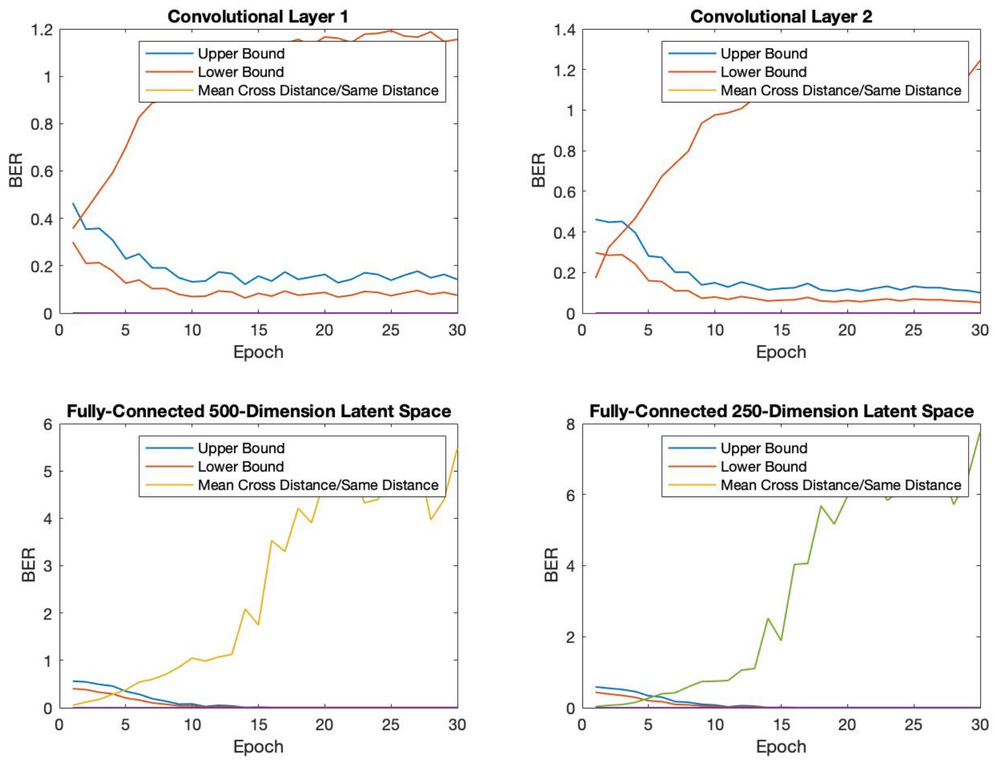


Figure 5.3: The estimated upper and lower bound for the BER, as well as a distance ratio, calculated in 4 different spaces– namely, different layers of the CNN.

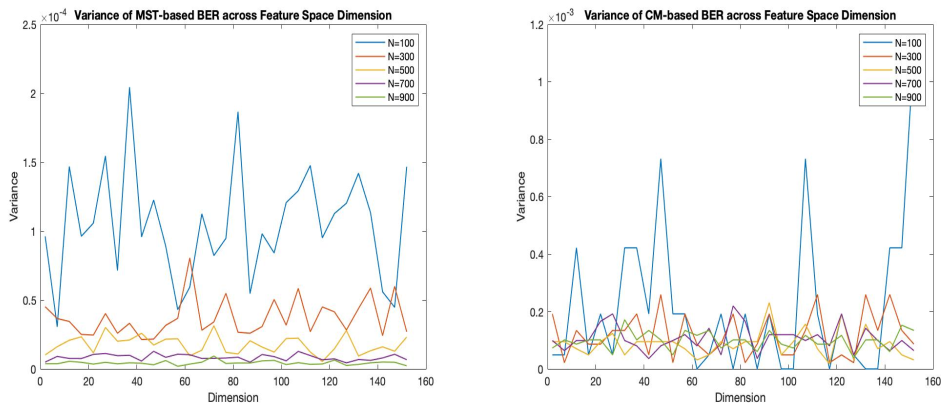


Figure 5.4: Variance of the cross-match- and MST-based BER estimates across data dimension for various sample sizes.

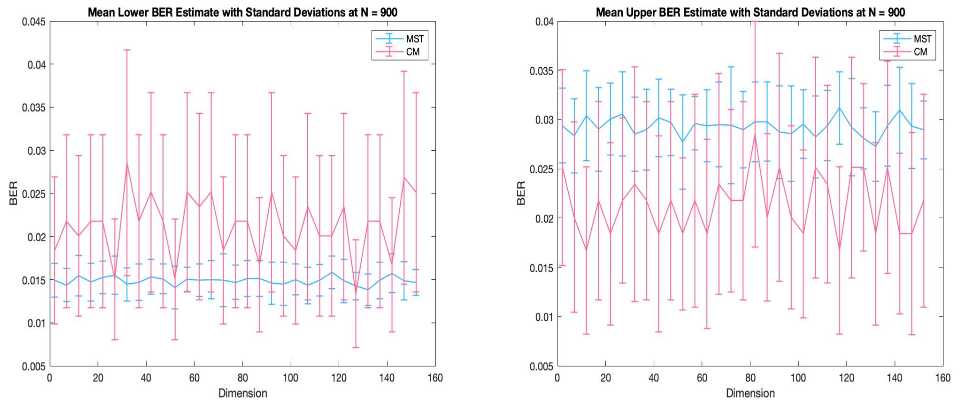


Figure 5.5: Means and standard deviations of BER estimates across different dimensions.

appears that at least for the MSTAR data set, the MST- methodology has a more consistent mean than that of the CM method. It also exhibits a lower variance than the latter across all dimensions in consideration. Finally, the BER bounds themselves are wider for the MST-based estimates, meaning this methodology yields more conservative bounds.

This result is not in line with the theory behind the cross-match statistic, whose variance should not depend on the dimension of the data. This is possibly because the data is not drawn from a normal distribution, as is done in [18].

These results seem to suggest that the MST-based bound is a stronger choice for estimating the BER based on (1) the dimensional consistency of its mean; (2) the dimensional consistency of its variance; (3) its computational efficiency; and (4) its ability to bound a non-binary classification problem. Of course, there is the question of the empirical accuracies of both methodologies, which is impossible to measure unless the posterior probability distributions are known.

Section 5.3 Experiment 3

The results of this experiment show that separability of data does not vary significantly upon the addition of Gaussian noise. This can be seen in Figure 5.6, which depicts five separate heatmaps representing the structure of the test data in the feature space. All five image fidelities exhibit a similar structure, implying that their BER estimates are quite close. It should be stated that this is not an ideal feature space structure. There is a significant amount of cross-links between the T72 and the BMP2, implying that the clusters are not very separate in the feature space. However, this experiment is only concerned with relative performance given the independent variable, that is the amount of blurring applied to the data set. With this in mind, it does not appear that the separability of the test data diminishes with the addition of Gaussian noise.

The fact that a neural network is able to classify extremely blurry images with high accuracy is quite significant, implying that there might be a learned denoising procedure within the CNN. From here, we can claim that smooth perturbations to training images do not hinder classification performance as long as the test data is convolved with the same blurring kernel as the training data.

But what if the test data exhibits a different amount of noise than the training data over which a model is formed? The second component of this experiment took high fidelity images, namely images with no Gaussian kernel applied to them, and passed them through models trained on blurred data. The question at the root of this experiment is *are the feature spaces of images of different fidelities compatible?*

The results show that the spaces are quite compatible. Figure 5.7a depicts the number of cross-links of non-blurred test data when it is projected into the feature spaces of lower-fidelity images. Figure 5.7b depicts the separability of non-blurred

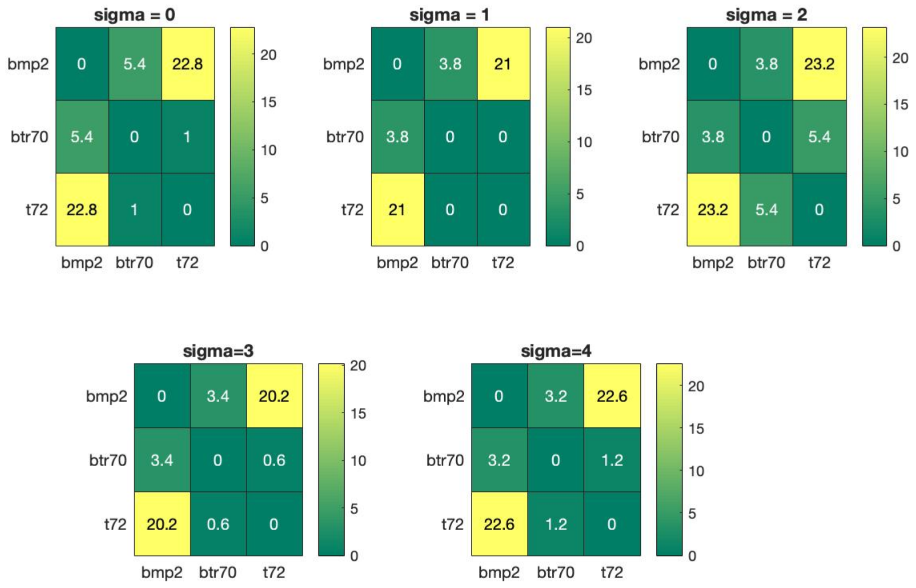
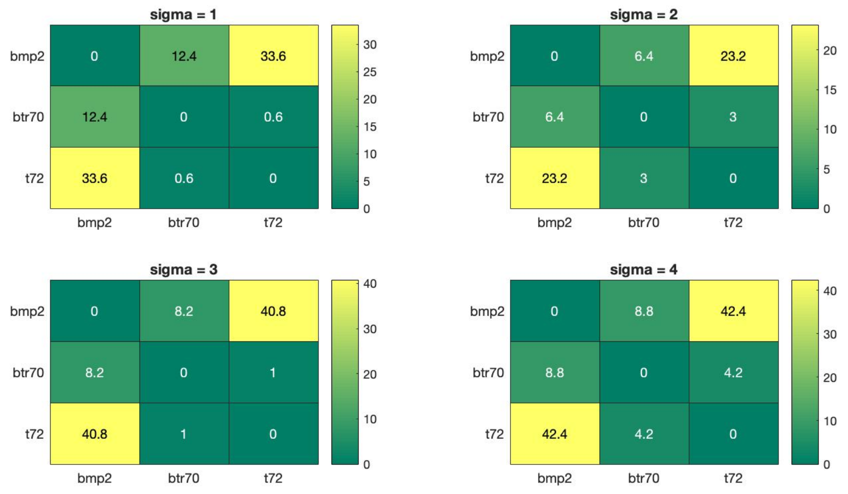


Figure 5.6: The structure of test data at different blurring thresholds in learned feature spaces trained at the same thresholds

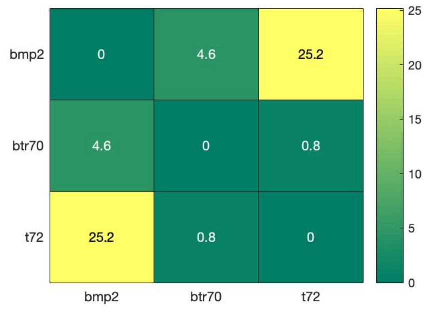
test data in its own feature space. Comparing the two figures, there is not a significant discrepancy in the structure of the cross-links. At higher values of sigma, there is an increase in the number of cross-links between the T72 and the BMP2 targets, but this could be a product of the training set, which consists of relatively few images of the BMP2.

Table 5.1 summarizes the upper and lower BER estimates for the high-fidelity data in different spaces, including its own. For the images convolved with noise with $\sigma = 1$ or $\sigma = 2$, the test data is just as separable, if not more separable, in the low-fidelity spaces. For higher values of sigma, the separability diminishes slightly, but not to a severe extent.

To better illustrate the structure of the samples in the feature spaces, we introduce a tSNE plot of the test data. **T-distributed Stochastic Neighbor Embedding** (tSNE), is a non-linear dimensional reduction algorithm. It aims to project high-dimensional data in 2 or 3 dimensions such that the distributions of data are preserved. The tSNE plots in Figure 5.8 illustrate individual samples of high-fidelity



(a)



(b)

Figure 5.7: *Top*: Cross-links of high-fidelity test data projected into feature spaces trained on low-fidelity data. *Bottom*: Cross-links of high-fidelity test data projected onto its own feature space.

data projected onto feature spaces trained using low-fidelity images. The samples show strong clustering in all of the feature spaces. However, tSNE cannot be relied on heavily, as it is prone to showing clusters even if those clusters are not truly present in the data. However, other information we have gathered, such as the few number of cross-links and the high distance ratio (as discussed in Experiment 2) support the clustering we see.

This result has a significant impact on current AF procedures. It's quite computationally intensive to generate simulated SAR data, which is an essential component of training target recognition algorithms. The speed and cost of this image genera-

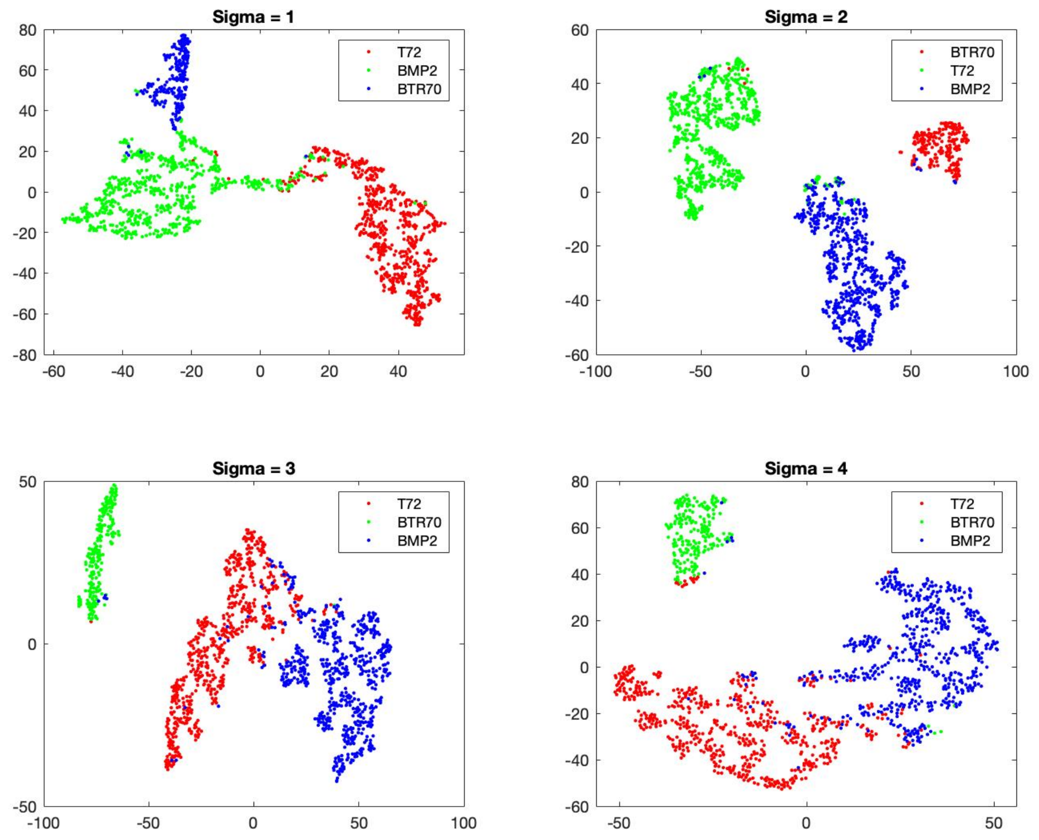


Figure 5.8: tSNE plots of high-fidelity data when projected on feature spaces of low-fidelity data.

	sigma = 1	sigma = 2	sigma = 3	sigma = 4	**sigma = 0**
uBER	0.0350	0.0260	0.0440	0.0500	**0.0340**
lBER	0.0177	0.0131	0.0224	0.0255	**.0172**

Table 5.1: Upper and lower BER estimates for high-fidelity test data projected into low-fidelity feature spaces. Bolded column shows BER bounds for non-blurred data in its own space.

tion can be greatly reduced if the required image fidelity threshold is lowered. This experiment shows that higher fidelity does not necessarily entail higher classifiability.

Chapter 6

Conclusion

This thesis takes a first hand look at applying Bayes error rate estimation to current problems in machine learning. As the world begins to rely more on machine learning models for automating procedures, architects must be presented with tools that are more reliable than accuracy metrics or precision-recall curves. The BER estimate can improve the holistic picture of how well a specific classifier can perform on a specific data set. Not only is BER estimation using the FR-statistic more computationally feasible than conducting many batches of testing, it offers worst-case performance estimates, which are crucial for quality control procedures.

Furthermore, the process of BER estimation unveils structural components in the feature space that would previously remain unknown. The number of cross-links between test data in a minimum-spanning tree unveils areas of overlap in the feature space. The distance ratio of cross-linked edges to non-cross-linked edges informs us of clustering patterns. This is all done without having to visualize the high dimensional space itself.

The final component this thesis provides is how BER estimation can be used to optimize procedures. It showed that high fidelity spatial data is not necessary to achieve strong classification performance; low fidelity data achieves similar degrees of separability in the feature space. But even more significantly, high fidelity and low fidelity feature spaces are actually compatible. Therefore, in the specific Air Force application discussed computational resources might not need to be diverted to producing the highest quality training data. Of course, further research must be conducted on this claim. Because the simulation algorithms are not publicly available, the thesis used Gaussian filters to distort image fidelity. This i.i.d. distortion most likely does not emulate the spatial distortion we would see if CAD model fidelity was altered before simulation. Nevertheless, this thesis suggests a degree of Gaussian noise invariance inherent in the CNN. Further research can examine what other invariances are present to optimize procedures.

It is important to note that the applications of this research transcend this Air Force specific use case. BER estimation can be used to evaluate any classifier-data tuple, unveiling worst- and best-case performance. A plethora of industry applications of machine learning depend on these measures, especially those in which the consequences of misclassification are dire.

There is a breadth of future research that can be done on BER estimation. The MST and optimal weighted matching is done using the l^2-norm, but different norms can be explored to yield different structures, and therefore different BER estimates. This research established that low fidelity feature spaces contain sufficient information to classify high fidelity data; future research should explore if the inverse is true. Real time SAR data often lacks pre-processing and is therefore of a lower fidelity than training data. A high degree of feature space compatibility in the inverse direction is promising for real-time target recognition. Finally, with a more industry-focused view, a BER estimation toolkit could improve existing performance evaluation processes in fields where machine learning models are used.

Bibliography

[1] *Mstar overview.*

[2] S. Amari, *α -divergence is unique, belonging to both f-divergence and bregman divergence classes*, IEEE Transactions on Information Theory **55** (2009), no. 11, 4925–4931.

[3] Andriy Burkov, *The hundred-page machine learning book*, Andriy Burkov Quebec City, Can., 2019.

[4] Frank Cabello, Julio León, Yuzo Iano, and Rangel Arthur, *Implementation of a fixed-point 2d gaussian filter for image processing based on fpga*, 2015 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), IEEE, 2015, pp. 28–33.

[5] John Duchi, Elad Hazan, and Yoram Singer, *Adaptive subgradient methods for online learning and stochastic optimization*, Journal of machine learning research **12** (2011), no. Jul, 2121–2159.

[6] Ruth Heller, Dylan Small, and Paul Rosenbaum, *crossmatch: The cross-match test*, 2012, R package version 1.3-1.

[7] C-H Hsu, W-C Feng, and Jeremy S Archuleta, *Towards efficient supercomputing: A quest for the right metric*, 19th IEEE International Parallel and Distributed Processing Symposium, IEEE, 2005, pp. 8–pp.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton, *Imagenet classification with deep convolutional neural networks*, Advances in Neural Information Processing Systems 25 (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), Curran Associates, Inc., 2012, pp. 1097–1105.

[9] Joseph B. Kruskal, *On the shortest spanning subtree of a graph and the traveling salesman problem*, Proceedings of the American Mathematical Society **7** (1956), no. 1, 48–50.

[10] Steve Lawrence, C Lee Giles, Ah Chung Tsoi, and Andrew D Back, *Face recognition: A convolutional neural-network approach*, IEEE transactions on neural networks **8** (1997), no. 1, 98–113.

[11] Yann LeCun, Corinna Cortes, and CJ Burges, *Mnist handwritten digit database*, ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist **2** (2010).

[12] Llew Mason, Jonathan Baxter, Peter L Bartlett, and Marcus R Frean, *Boosting algorithms as gradient descent*, Advances in neural information processing systems, 2000, pp. 512–518.

[13] MATLAB, *version 9.5 (r2018b)*, The MathWorks Inc., Natick, Massachusetts, 2018.

[14] Christos H Papadimitriou and Kenneth Steiglitz, *Combinatorial optimization: algorithms and complexity*, Courier Corporation, 1998.

[15] Ning Qian, *On the momentum term in gradient descent learning algorithms*, Neural networks **12** (1999), no. 1, 145–151.

[16] R Core Team, *R: A language and environment for statistical computing*, R Foundation for Statistical Computing, Vienna, Austria, 2013.

[17] Karthik Ramasubramanian and Abhishek Singh, *Deep learning using keras and tensorflow*, Machine Learning Using R, Springer, 2019, pp. 667–688.

[18] Paul R. Rosenbaum, *An exact distribution-free test comparing two multivariate distributions based on adjacency*, Journal of the Royal Statistical Society: Series B (Statistical Methodology) **67** (2005), no. 4, 515–530.

[19] Sebastian Ruder, *An overview of gradient descent optimization algorithms*, arXiv preprint arXiv:1609.04747 (2016).

[20] Robert Sedgewick, *Algorithms*, Pearson Education India, 1988.

[21] S. Y. Sekeh, B. Oselio, and A. O. Hero, *A dimension-independent discriminant between distributions*, 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2018, pp. 4419–4423.

[22] ______, *Multi-class bayes error estimation with a global minimal spanning tree*, 2018 56th Annual Allerton Conference on Communication, Control, and Computing (Allerton), 2018, pp. 676–681.

[23] Patrice Y Simard, David Steinkraus, John C Platt, et al., *Best practices for convolutional neural networks applied to visual document analysis.*, Icdar, vol. 3, 2003.

[24] Elizabeth R. Sudkamp, John W. Nehrbass, Eric Branch, and Michael Levy, *Synthetic data accuracy sensitivity to CAD model accuracy using ATR-based metrics*, Algorithms for Synthetic Aperture Radar Imagery XXVI (Edmund Zelnio and Frederick D. Garber, eds.), vol. 10987, International Society for Optics and Photonics, SPIE, 2019, pp. 55 – 62.

[25] Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J Smola, *Parallelized stochastic gradient descent*, Advances in neural information processing systems, 2010, pp. 2595–2603.