WORCESTER POLYTECHNIC INSTITUTE

Kernel Coherence Encoder

by

Fangzheng Sun

A thesis

Submitted to the Faculty

of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Master of Science

in

Data Science

April 2018

APPROVED:

Professor Randy C. Paffenroth, Adviser:

_____

Professor Jian Zou, Reader:

_____

Contents

WORCESTER POLYTECHNIC INSTITUTE

Abstract

Data Science

Master of Science

by Fangzheng Sun

In this thesis, we introduce a novel model based on idea of autoencoders. However, different from a classic autoencoder reconstructing its own inputs through a neural network, our model takes advantage of Kernel Canonical Correlation Analysis (KCCA) and reconstructs input data from another data set, where these two data sets should have correlation. Our model extends traditional KCCA in a way that the non-linearity of the data is learned through an optimal kernel function within a neural network. In one of the novel parts of this thesis, we do not optimize our kernel based upon some prediction error metric, as is classic in autoencoders. Rather, we optimize our kernel to maximize the "coherence" of the underlying low-dimensional hidden layers. This idea makes our method faithful to the classic interpretation of linear Canonical Correlation Analysis (CCA). As far we are aware, our method, which we call a *Kernel Coherence Encoder (KCE)*, is the only extent approach that uses the flexibility of a neural network, while maintaining the theoretical properties of classic KCCA. Note, there are several computational and robustness issues in the implementation of a coherence based deep learning KCCA. Accordingly, in another one of the novelties of our approach, we leverage a modified version of classic coherence which is far more stable in the presence of high-dimensional data.

Chapter 1

Introduction

In the recent years, the concept of deep learning is becoming a hot topic for divergent industries as well as multiple fields of study [9, 14, 18, 40]. Deep learning is a subfield of machine learning. There are many sub-categories of deep learning. Herein, we focus on artificial neural network (ANN) [24, 25] and autoencoders [2–4] that allow us to discover non-linear low-dimensional manifold. Such methods are frequently use for unsupervised learning and for producing low-dimensional representations of data. Typically, an autoencoder consists two parts, an encoding phase (encoder) $E(\cdot)$ and an decoding phase (decoder) $D(\cdot)$. The encoder $E(\cdot)$ takes the input data and maps it to a low-dimensional representation. The decoder $D(\cdot)$ then takes this low-dimensional representation and attempt to reconstruct the original high-dimensional data.

Note, such an autoencoder is closely related to Principal Component Analysis (PCA) [6, 7, 17, 27]. In fact, an autoencoder is identical to PCA when each layer is assumed to be linear and the reconstruction cost function is Euclidean distance. In particular, many people are attracted by the nonlinear aspect of autoencoders, which makes it more powerful than a typical PCA, and effective in solving more real-life problems.

Here, we increase the performance of CCA [1, 5, 13] in the same way by leveraging the non-linear capabilities of autoencoders. We propose a model that extends the traditional CCA from linear to non-linear in the ideas of non-linear autoencoders. CCA and PCA differ from that PCA attempts to reconstruct a set of data from itself, while CCA has two data sets, and it tries to reconstruct one data set from the other. It is this added complexity of having two data sets that makes our proposed Kernel Coherence Encoder (KCE) non-trivial to develop.

At a high level, our method proceed in two phases. First we train an "encoder" to take our input data and map it to a space called Reproducing Kernel Hilbert Space (RKHS) [10] by a

particular kernel function, where the coherence between the input data is maximized. Note, this kernel function is predefined with some parameters that trained through the "encoder". With this well-trained kernel function, represented by the encoder, we can maximize the coherence of two potentially correlated datasets by mapping them to RKHS where their correlations become linear and much easier to be explored. Second, we fix the "encoder", and then train the "decoder" comprised of neural networks to reconstruct our original two data sets.

1.1   Motivation

Let us begin the story of our research with some interesting questions that everyone may have: at the age of 20, are you curious what you may look like at the age of 40? Or, when you looks at a 40-year-old person, can you imagine what he or she looks like at the age of 20? Human beings, the super intelligent creature, has very complicated and powerful brain that learns and calculates fast. In your mind, you may have brief answers to the above questions, based on your observations of other people's looking and their changes over a 2-decade period. If you are a talented artist, you can even depict your answer on the canvas. Your brain learns and trains from these observations of faces and then starts to collect the underlying rules for your final answers. Then you more or less find a "mapping" between a 20-year-old face and a 40-year-old face. Consequently, given a "testing" face or face image, your answer becomes sophisticated instead of a random guess.

We want our model to mimic this learning process of a human brain, while maintaining its own trick to extract the "correlation" between data sets. Our purpose to design the KCE model is to reconstruct two "potentially correlated datasets" from one to another. An autoencoder focuses on one input data set. Given two correlated data sets, we introduce KCCA to dig out the correlation between them.

The idea of applying kernel methods to autoencoders initially comes from Yan Pei's work *Auto-encoder Using Kernel Methods* [9]. In his model, he replaces the encoder part of an autoencoder with Kernel PCA (KPCA) [27] and the decoder with kernel-based linear regression. It is a model that is designed to work similarly as an autoencoder, but remains faithful to the "kernel trick". The "kernel trick", which will be discussed in detail in the next chapter, is a classic technique for non-linearly projecting features to a high-dimensional space, gaining the benefit of complex non-linear feature, without incurring the cost of working directly in a high-

dimensional space. Although it is reasonable, and perhaps even obvious, to try to connect the kernel trick and deep learning, it is notable that, as far as we are aware, the currently methods such as [9], do not leverage the full power of autoencoders.

1.2 Contribution

We gain the initial vision of CCA through *Empirical canonical correlation analysis in subspaces* [5]. This paper provides detailed explanations of CCA, including the calculations for canonical variables of data pairs. Meanwhile, we are inspired by the authors' idea to maximize their coherence through CCA. However, in most real-life problems, the correlations of dataset are not necessarily linear, thus calculating their linear coherence may not be the best solution to dig their correlations. We take a further step to its kernel version, KCCA. With the help of deep learning, we initialized a kernel function with some hyper parameters and train the kernel function to maximize the coherence of two data sets in the RKHS. Then we extract the canoincal variables for data reconstruction.

In our model, we solve two problems caused by high dimensionality of data. First, the training process is very slow. To fix this issue, we apply PCA to the original dataset to decrease the dimension to an acceptable level, where the principal components carry most of information of original data (explain $> 90\%$ variance). Second, CCA is very sensitive to high dimensionality because the high-dimensional Gramian matrix [11] containing the inner products is easy to become singular (not invertible) [12] in the new space. So it becomes impossible to calculate its determinant. In our model, we slightly modify the formula provided by [5] to avoid this issue at a cost of losing some computational efficiency. The modifications will be explained later.

In the next step, we accopmlish the reconstruction by using both linear and non-linear neural networks and PCA reconstruction to map the principle components back to the original space of the another data set.

To demonstrate the effectiveness of KCCA in our model, we use MNIST digit image data as an experimental example and compare the performance of our model to two other well-designed comparison models.

Chapter 2

Background

In this chapter, we will introduce necessary backgrounds that form our ideas and algorithm. We are going to introduce the dimension reduction algorithm PCA, the correlation extraction methods CCA and KCCA and the deep learning frameworks in the Python deep learning package *tensorflow* where we leverage the full power of the autoencoders. This chapter will be divided to seven sections. In the first section of this chapter, we introduce the main idea of PCA. This is the process we use to do dimension reduction for computational efficiency. Then in the next 3 sections, we give detailed explanation of the CCA, the kernel methods (the kernel trick) and the KCCA. This part contains definitions used in our model and their fundamental theoretical backgrounds and theorems. The last three sections of this chapter list training methods in the literature, including deep learning, autoencoders and the open source software library TensorFlow [14–16] in Python. These ideas are used in our model to train the kernel function as well as the linear and non-linear mappings.

2.1 Principal Component Analysis

Principal Component Analysis (PCA) is a popular unsupervised learning approach for a low-dimensional set of features from a large set of variables. For high-dimensional data, principal components allow us to summarize this set with a smaller number of representative variables that collectively explain most of the variability in the original set. PCA refers to the process by which principal components are computed, and the subsequent use of these components in understanding the data. The first principal component of a set of features $X = (X_1, X_2, ..., X_p)$

is the normalized linear combination of the features

$$Z_1 = \phi_{11}X_1 + \phi_{21}X_2 + ... + \phi_{p1}X_p \tag{2.1}$$

that has the largest variance. After the first principal component Z_1 of the features has been determined, we can find the second principal component Z_2. The second principal component is the linear combination of $X = (X_1, X_2, ..., X_p)$ that has maximal variance out of all linear combinations that are uncorrelated with Z_1. The third principal component is the linear combination of $X = (X_1, X_2, ..., X_p)$ that has maximal variance out of all linear combinations that are uncorrelated with Z_1 and Z_2. So on and so forth [17].

Finally we will have totally p unique principal components $Z_1, Z_2, ..., Z_p$. Each unique principal component Z_i explains a specific percentage of variance to the original dataset, a positive quantity, ranging from 0 to 1. And the percentages of all p principal components sum to 1. In most programming PCA tools, the principal components appear with a descending order of the percentages by variance they explain. These percentages are cumulative, indicating the cumulative variance explained to the original dataset by their corresponding principal components. Thus, to understand the data and execute proceeding analysis after PCA, we typically extract first k principal components $Z_1, Z_2, ..., Z_k, k < p$. These k principal components usually explain most of the variance of the original data, above 90% or 95%.

2.2 Canonical Correlation Analysis

In statistics, Canonical Correlation Analysis (CCA) is a way of inferring information from cross-covariance matrices. If we have two vectors $X = (X_1, ..., X_n)$ and $Y = (Y_1, ..., Y_m)$ of random variables, and there are correlations among the variables, then CCA will find linear combinations of the X_i and Y_j which have maximum correlation with each other [13]. From [5], we derive the following formulas to explain this algorithm.

Consider two random vectors $x \in \mathbb{R}^m$ and $y \in \mathbb{R}^n$. Let $z = \begin{bmatrix} x^T & y^T \end{bmatrix}^T \in \mathbb{R}^{m+n}$. We assume that x and y have zero means and share the nonsingular composite covariance matrix

$$R_{zz} = E[zz^T] = \begin{bmatrix} R_{xx} & R_{xy} \\ R_{xy} & R_{yy} \end{bmatrix} \tag{2.2}$$

where the elements of the cross-covariance matrix R_{xy} are inner products in Hilbert space of second-order random variables. $R_{xy}[i, j] = E[x_i y_j]$ is the inner product between random variables x_i and y_j in the Hilbert space.

The coherence matrix of x and y is defined as

$$C = E[(R_{xx}^{-1/2}x)(R_{yy}^{-1/2}y)^T] = R_{xx}^{-1/2} R_{xy} R_{yy}^{-T/2} \tag{2.3}$$

A singular value decomposition (SVD) for the coherence matrix C may be written as

$$C = R_{xx}^{-1/2} R_{xy} R_{yy}^{-T/2} = F\Sigma G^T,$$
$$F^T C G = F^T R_{xx}^{-1/2} R_{xy} R_{yy}^{-T/2} G = \Sigma \tag{2.4}$$

where $F \in \mathbb{R}^{m \times m}$ and $G \in \mathbb{R}^{n \times n}$ are orthogonal matrices and the matrix Σ is a diagonal singular value matrix with $\Sigma(m) = diag[\sigma_1, \sigma_2, ..., \sigma_m]$; $\sigma_1 \geq \sigma_2 \geq ... \geq \sigma_m > 0$. Let the elements of $u \in \mathbb{R}^m$ and $v \in \mathbb{R}^n$ be the canonical coordinates of x and y.

The diagonal matrix

$$\Sigma = F^T C G = F^T R_{xx}^{-1/2} R_{xy} R_{yy}^{-T/2} G = E[(F^T R_{xx}^{-1/2}x)(G^T R_{yy}^{-T/2}y)^T] = E[uv^T] \tag{2.5}$$

is the canonical correlation matrix. Each σ_i is the correlation between pairs of canonical coordinates (u_i, v_i).

From the above equation, we extract the following expression for canonical variables u and v:

$$u = F^T R_{xx}^{-1/2}x, \quad v = G^T R_{yy}^{-T/2}y \tag{2.6}$$

The standard measure of linear dependence for the composite vector is the Hadamard ratio where the ratio takes the value 0 if and only if there is linear dependence among the elements of the composite vector and takes the value 1 if and only if elements of the composite vector are mutually uncorrelated.

In the paper, the author made a decomposition of the Hadamard ratio and derived the following term

$$L = det(I - \Sigma\Sigma^T) = \prod_{i=1}^{m}(1 - \sigma_i^2); \quad 0 \leq L \leq 1 \tag{2.7}$$

which measures the linear dependence between x and y. Correspondingly,

$$H = 1 - L = 1 - det(I - \Sigma\Sigma^T) = 1 - \prod_{i=1}^{m}(1 - \sigma_i^2); \quad 0 \leq H \leq 1 \qquad (2.8)$$

measures the linear coherence between x and y.

2.3 Kernel Methods

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick" [18]. Given x_i and y_j, the function $K(x_i, y_j) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is often referred to a kernel function.
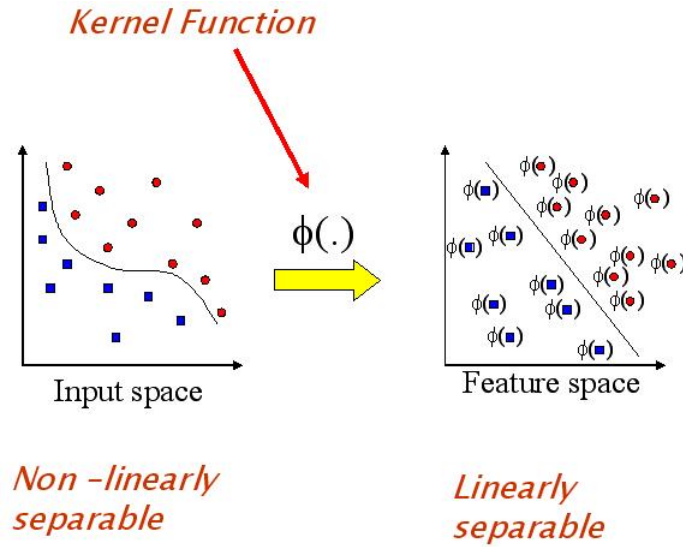

Figure 2.1: A visual demonstration of kernel function. A kernel function maps data from original feature space to a higher dimensional space and avoids the explicit mapping that is needed to get linear learning algorithms to learn a nonlinear function or decision boundary [42].

2.3.1 Positive Definite Kernel

A positive definite kernel is a generalization of a positive definite function or a positive definite matrix. It makes sure there is a corresponding inner product space for x and y. Its greatest advantage is that positive definite kernels can be defined on every set. Therefore it can be applied to data of any type.

Certain problems in machine learning have additional structure than an arbitrary weighting function K. The computation is made much simpler if the kernel can be written in the form of a "feature map" $\varphi : \mathcal{X} \to \mathcal{V}$ which satisfies $K(x, x') = \langle \varphi(x), \varphi(x') \rangle \mathcal{V}$. The key restriction is that $\langle \cdot, \cdot \rangle \mathcal{V}$ must be a proper inner product. On the other hand, an explicit representation for φ is not necessary, as long as $\mathcal{V}$ is an inner product space [19].

In KPCA or KCCA, a kernel function maps data from original space to an inner product space RKHS to ensure both the existence of an inner product and the evaluation of every function in this space at every point in the domain. From the definition of RKHS, the corresponding kernel function should be both symmetric and positive definite [10].

2.3.2 Mercer's Theorem

To explore a kernel function K without the explicit representation for φ, we apply Mercer's theorem: an implicitly defined function φ exists whenever the space $\mathcal{X}$ can be equipped with a suitable measure ensuring the function K satisfies Mercer's condition.

In mathematics, specifically functional analysis, Mercer's theorem is a representation of a symmetric positive definite function on a square as a sum of a convergent sequence of product functions. This theorem, presented in (Mercer 1909), is one of the most notable results of the work of James Mercer [20]. To explain Mercer's theorem, we first consider an important special case; see below for a more general formulation. A kernel, in this context, is a symmetric continuous function

$$K : [a, b] \times [a, b] \to \mathbb{R} \tag{2.9}$$

where symmetric means that $K(x, s) = K(s, x)$. K is said to be non-negative definite (or positive semidefinite) if and only if

$$\sum_{i=1}^{n} \sum_{j=1}^{n} K(x_i, x_j) c_i c_j \geq 0 \tag{2.10}$$

2.3.3 Schoenberg's Theroem

Suppose a kernel function K is not positive definite, it may not represent an inner product in any Hilbert space. Here is one way to see. A kernel K is positive definite if and only if for all samples of n points, its corresponding kernel matrix $\mathcal{K}$ is a positive definite matrix. With a positive definite $\mathcal{K}$, by Cholesky decompose $\mathcal{K} = LL^T$, each row of L as one mapped point in

the inner product space. If K is not positive definite, the matrix $\mathcal{K}$ may also not be positive definite. Consequently, Cholesky does not work and there is no corresponding inner product space.

Definition. A function f with domain $(0, \infty)$ is said to be completely monotonic, if it possesses derivatives $f^{(n)}(x)$ for all $n = 0, 1, 2, 3, ...$ and if $(-1)^n f^{(n)}(x) \geq 0$ [21].

Schoenberg [22] provides a method to ensure the kernel function to be positive definite: If $f(t)$ is a completely monotonic function, then the radial kernel

$$K(x, y) = f(\|x - y\|^2) \tag{2.11}$$

is positive definite on any Hilbert space. Since the radial kernel ia already symmetric, the above Schoenberg's theroem provides us a direct way to get a symmetric positive definite kernel function satisfying RKHS definition for our model.

2.4   KCCA

In section 2.2, we explain CCA step by step. For two random vectors $x \in R^m$ and $y \in R^n$, let $z = \begin{bmatrix} x^T & y^T \end{bmatrix}^T \in R^{m+n}$ and assume that x and y have zero means. formula 2.2 is the covariance matrix of x and y. The elements of the cross-covariance matrix R_{xy} are inner products in Hilbert space of second-order random variables. $R_{xy}[i, j] = E[x_i y_j]$ is the inner product between random variables x_i and y_j in the Hilbert space.

Now we apply the kernel method to obtain the covariance matrix of x and y in RKHS: here we define

$$\begin{aligned} R_{xx}[i, j] &= K(x_i, x_j) \\ R_{xy}[i, j] &= K(x_i, y_j) \\ R_{yx}[i, j] &= K(y_i, x_j) \\ R_{yy}[i, j] &= K(y_i, y_j) \end{aligned} \tag{2.12}$$

where $K(x_i, y_j) : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ is a positive definite kernel function. Thus the covariance matrix becomes

$$R_{zz} = \begin{bmatrix} R_{xx} & R_{xy} \\ R_{xy} & R_{yy} \end{bmatrix} =$$

$$\begin{bmatrix} \begin{pmatrix} K(x_1, x_1) & \ldots & K(x_1, x_m) \\ \vdots & \ddots & \\ K(x_m, x_1) & & K(x_m, x_m) \end{pmatrix} & \begin{pmatrix} K(x_1, y_1) & \ldots & K(x_1, y_n) \\ \vdots & \ddots & \\ K(x_m, y_1) & & K(x_m, y_n) \end{pmatrix} \\ \begin{pmatrix} K(y_1, x_1) & \ldots & K(y_1, x_m) \\ \vdots & \ddots & \\ K(y_n, x_1) & & K(y_n, x_m) \end{pmatrix} & \begin{pmatrix} K(y_1, y_1) & \ldots & K(y_1, y_n) \\ \vdots & \ddots & \\ K(y_n, y_n) & & K(y_n, y_n) \end{pmatrix} \end{bmatrix} \quad (2.13)$$

where the elements of the cross-covariance matrix R_{xy} are inner products in the RKHS of random variables x and y. $R_{xy}[i, j] = K(x_i, y_j)$ is the inner product between random variables x_i and y_j in the RKHS.

We mimic all the remaining formulas from 2.3 to 2.8, except that, with kernel methods, L and H now measure the non-linear dependence and non-linear coherence between x and y respectively.

2.5   Deep Learning

Deep learning is a class of machine learning techniques that exploit many layers of non-linear information processing for supervised algorithms, such as pattern analysis and classification. It is a sub-field within machine learning that is based on algorithms for learning multiple levels of representation in order to model complex relationships among data. An observation (e.g., an image) can be represented in many ways (e.g., a vector of pixels), but some representations make it easier to learn tasks of interest (e.g., is this the image of a human face?) from examples, and research in this area attempts to define what makes better representations and how to learn them [23]. In this section, we will focus on a narrower, but now commercially important, subfield of Deep Learning, artificial neural networks (ANN).

A standard ANN consists of many simple, connected processors called neurons, each producing a sequence of real-valued activations. Input neurons get activated through sensors perceiving the environment, other neurons get activated through weighted connections from previously active neurons [24]. An ANN consists of an input layer of neurons (or nodes, units), one or two

(or even three) hidden layers of neurons, and a final layer of output neurons. Figure 2.2 shows a typical architecture of a neural network, where lines connecting neurons are also shown. Each connection is associated with a numeric number called weight. The output, h_i, of neuron i in the hidden layer is

$$h_i = \sigma(\sum_{j=1}^{N} V_{ij}x_j + T_i^{hid}) \tag{2.14}$$

where $\sigma()$ is activation function, N the number of input neurons, V_{ij} the weights, x_j inputs to the input neurons, and T_i^{hid} the threshold terms of the hidden neurons [25]. We usually write the above formula in the form of

$$h = \sigma(Wx + b) \tag{2.15}$$

where $\sigma()$ is the activation function, W the weight matrix and b the bias vector.
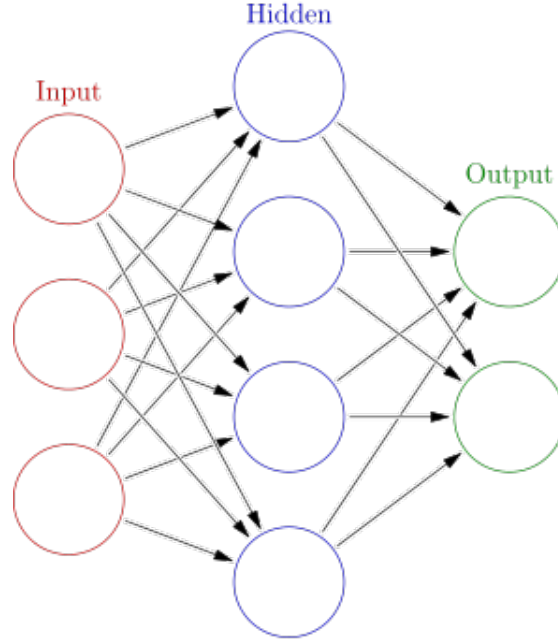

Figure 2.2: An artificial neural network is an interconnected group of nodes, akin to the vast network of neurons in a brain. Here, each circular node represents an artificial neuron and an arrow represents a connection from the output of one artificial neuron to the input of another [43].

2.5.1 Weight Matrix and Bias Vector

In the inner formula of each neuron $h = \sigma(Wx + b)$, we have two network parameters W and b inside the activation function $\sigma()$. They are the weight matrix and bias vector respectively. Each layer (neuron) has its own weight matrix and bias vector. For a neural networks with

n layers, we have totally n pairs of such parameters, $[W^1, W^2, ..., W^n]$ and $[b^1, b^2, ..., b^n]$. The dimensions of each W^i and b^i are determined by the dimensions of the input and output values in each layer. For example, suppose the input $x \in \mathbb{R}^d$ and the layer maps it to $h \in \mathbb{R}^p$. In this case, the weight matrix W^i functions to map data from $\mathbb{R}^d$ to $\mathbb{R}^p$, so it is a $p \times d$ matrix. The bias vector b^i is a vector with length p.

2.5.2 Non-linear Activation Function

If we apply the two linear projection functions $h = W^1 x + b^1$ and $z' = W^2 x + b^2$ and the cost function is Euclidean distance $\|x - x'\|_2$, this two-layer neural network is identical to the PCA. However, most people are attracted by the nonlinear aspect of the autoencoders, which makes it effective in solving more real-life problems.

To introduce non-linearity into ANN, we apply element-wise non-linear functions such as a sigmoid function or a rectified linear unit after linear projections. The non-linear function in each neuron is $h = f(Wx + b)$ where the non-linear mapping f acts as the activation function. The commonest activation function is sigmoid function $f(t) = \dfrac{1}{1 + e^{-t}}$. We use $\sigma()$ to represent this sigmoid function. Thus each layer within an ANN consists a function $h = \sigma(Wx + b)$, taking input $x \in \mathbb{R}^d$ and mapping it to $z \in \mathbb{R}^p$.
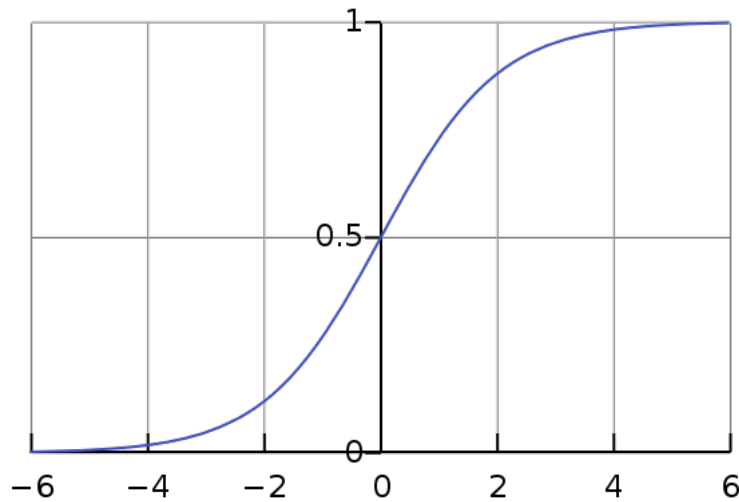

Figure 2.3: Logistic curve $f(t) = \dfrac{1}{1 + e^{-t}}$ [44].

2.6   Autoencoders

An autoencoder is an artificial neural network used for unsupervised learning of efficient codings. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for the purpose of dimensionality reduction. Autoencoders are simple learning circuits which aim to transform inputs into outputs with the least possible amount of distortion. While conceptually simple, they play an important role in machine learning. Recently, autoencoders have taken center stage again in the deep architecture approach and this concept has become more widely used for learning generative models of data [2, 4].
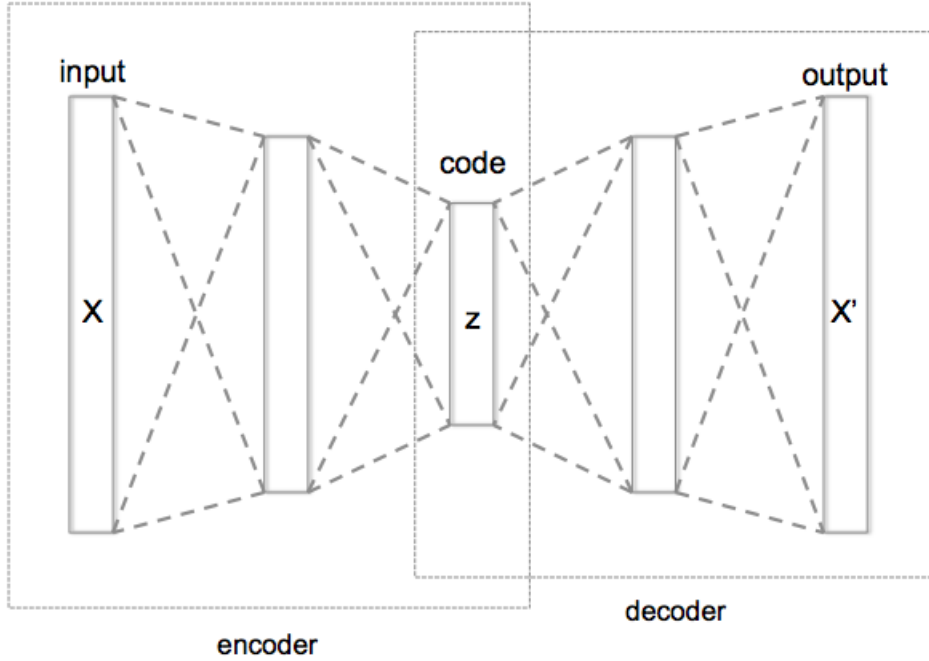

Figure 2.4: Schematic structure of an autoencoder with 3 fully-connected hidden layers [45].

Typically, an autoencoder consists two parts, an encoding phase (encoder) $E(\cdot)$ and an decoding phase (decoder) $D(\cdot)$. The encoder $E(\cdot)$ receives input data and outputs mid-layer-data, which acts as the input data for the decoder $D(\cdot)$. The decoder maps the mid-layer-data back to the original feature space. The encoder and decoder can be comprised of single layer or multiple layers, each layer with a function $z = \sigma(Wx+b)$ mapping $x \in \mathbb{R}^d$ to $z \in \mathbb{R}^p$. σ is an element-wise activation function such as a sigmoid function or a rectified linear unit. A typical autoencoder structure is shown as following:

In the simplest case, where there is one hidden layer, the encoder stage of an autoencoder maps $x \in \mathbb{R}^d = \mathcal{X}$ to $z \in \mathbb{R}^p = \mathcal{F}$ (z is hidden layer, for this simplest case, it is also the mid-layer of the whole autoencoder). The encoder $E(\cdot)$ and the decoder $D(\cdot)$ can be defined as transitions ϕ and ψ,

$$
\begin{aligned}
\phi &: \mathcal{X} \to \mathcal{F} \\
\psi &: \mathcal{F} \to \mathcal{X} \\
\phi, \psi &= \underset{\phi, \psi}{\arg\min} \, \|X - (\phi \circ \psi)X\|^2
\end{aligned}
\tag{2.16}
$$

This can be written as

$$
z = \sigma(Wx + b) \tag{2.17}
$$

The image z is usually referred to as code, latent variables, or latent representation. Here, $\sigma()$ is an element-wise activation function such as a sigmoid function or a rectified linear unit. W is a weight matrix and b is a bias vector. The decoder stage of the autoencoder maps z to the reconstruction x' of the same shape and in the same dimension with original data x:

$$
x' = \sigma'(W'z + b') \tag{2.18}
$$

where σ', W' and b' for the decoder may differ in general from the corresponding σ, W and b for the encoder, depending on the design of the autoencoder.

In the training process, autoencoders are usually trained to minimize reconstruction errors (such as squared errors) between original x and the reconstructed x':

$$
\mathcal{L}(x, x') = \|x - x'\|^2 = \|x - \sigma'(W'(\sigma(Wx + b)) + b')\|^2 \tag{2.19}
$$

where x is usually averaged over some input training set [2].

2.7 TensorFlow

TensorFlow is an open source software library for numerical computation using data flow graphs, a machine learning system that operates at large scale and in heterogeneous environments. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. In TensorFlow, devel-

opers have the options among deploying computations to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but recently this system becomes open to all developers around the world and broadens its range to a wider variety of domains [15].

TensorFlow uses dataflow graphs to represent computation, shared state, and the operations that mutate that state. It maps the nodes of a dataflow graph across many machines in a cluster, and within a machine across multiple computational devices, including multicore CPUs, generalpurpose GPUs, and custom-designed ASICs known as Tensor Processing Units (TPUs). This architecture gives flexibility to the application developer: whereas in previous parameter server designs the management of shared state is built into the system, TensorFlow enables developers to experiment with novel optimizations and training algorithms [14].
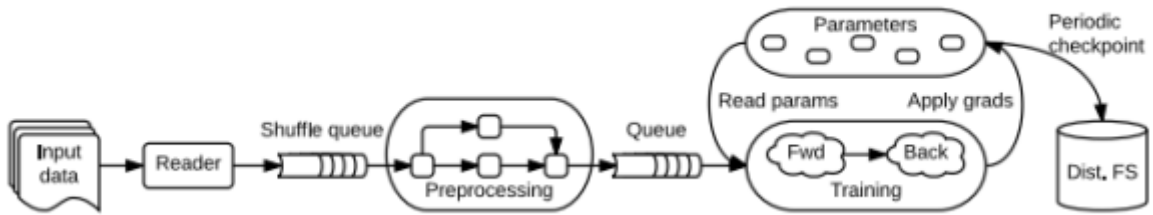

Figure 2.5: A schematic TensorFlow dataflow graph for a training pipeline, containing subgraphs for reading input data, preprocessing, training, and checkpointing state [46].

Based on its great flexibility and programming efficiency, we choose TensorFlow as the tool to implement all training sessions in our model.

Chapter 3

Kernel Coherence Encoder

In section 2.5 and 2.6, we described how autoencoders learn through training and finally reconstruct data x back to its original space, with minimal reconstruction errors. Herein we try to modify this task that given input dataset x, we want the model to reconstruct another "potentially correlated dataset" y. For this purpose, the autoencoders do not work well. The main reason is that an autoencoder does not have training process on extracting the correlation between the two datasets x and y. Instead, it merely explores "direct mapping" from one cluster of patterns to another one. This leads to devastating issues in reconstruction results. In next chapter, we will use some image example to show this issue. To avoid such issues and accomplish the new task, we introduce KCCA to dig out the correlations between the pair of input data sets and create a novel model KCE.

In this chapter, we will describe our model KCE in details, including the step-wise explanations, training algorithms and evaluation metrics of reconstruction results. Finally we will apply Model Comparison [26] methods and use two comparison models for control experiment.

3.1 Model Design

This section is a general introduction of our model KCE. The flow chart in figure 3.1 provides a step-by-step description of the model itself. For each step (rectangle in yellow) shown in the flow chart (PCA, KCCA, linear ANNs, PCA reconstruction and non-linear ANNs), we are going to explain their functionalities and implementation details one by one.
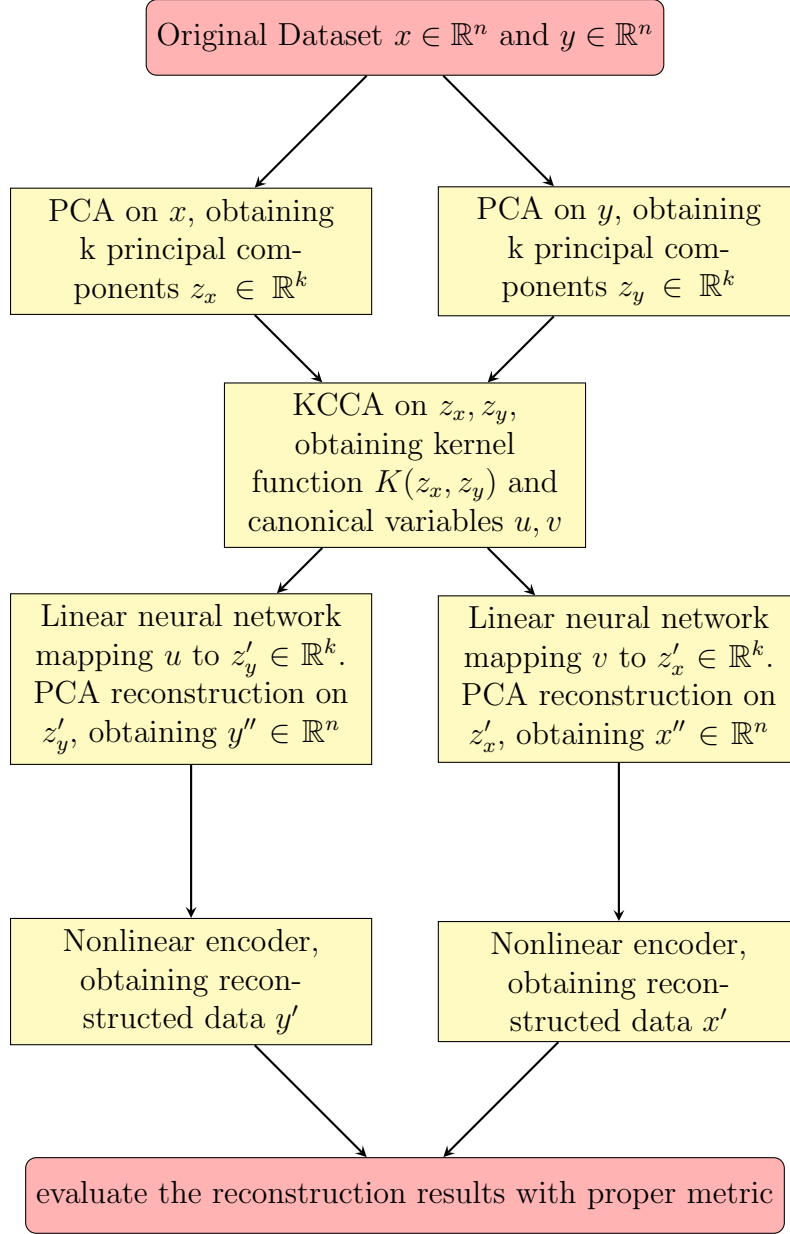
Figure 3.1: Flow chart of our KCE model

This model design, though having very different structure with an autoencoder, maintains its major properties: dimension reduction, non-linear mapping and reconstruction error minimization. Consequently, we divide this model into the encoder phase and decoder phase. The PCA and the KCCA steps work as the encoder phase. The encoder phase reduces the dimension of input data, extracts their mutual information and "stores" the information into canonical variables. The linear ANNs, the PCA reconstruction and the non-linear ANNs work as the decoder phase. The decoder phase, with the help of mutual information within the canonical variables, maps data back to their original spaces and finish the reconstruction. The canonical variables u and v, the mid-layer between the two phases, are the keys for reconstruction.

3.1.1 Encoder Phase

3.1.1.1 PCA

In real-life problems, data $x \in \mathbb{R}^n$ and $y \in \mathbb{R}^n$ are usually high-dimensional data. We have two concerns with this issue: First, directly using the high-dimensional data as input data will slow down the ANN, and moreover, cause over-fitting [17], making our model neither efficient nor effective. Second, for the subsequent step KCCA, we need to calculate the determinant of the kernel covariance matrix (Gramian matrix) $R_{zz} \in \mathbb{R}^{2n} \times \mathbb{R}^{2n}$. From experimental trials, we find that high-dimensional Gramian matrix is easy to become singular (not invertible) in the new space thus impossible to calculate its determinant.

So we apply PCA to reduce the dimensions of original data before using KCCA. As we described in chapter 2, PCA is a dimension reduction algorithm that allows us to summarize high-dimensional data set with a smaller number of representative variables that collectively explain most of the variability in the original set. PCA refers to the process by which principal components are computed, and the subsequent use of these components in understanding the data.

To reach a trade-off between model accuracy and the above two concerns, we take the first k principal components $Z_1, Z_2, ..., Z_k$ that explain over 90% variance of the original data, where k is a much smaller integer compared with the original dimension n. As a result, the input data for the KCCA step become $z_x \in \mathbb{R}^k$ and $z_y \in \mathbb{R}^k$. Meanwhile, we keep the eigenvectors and means of x and y generated in this step for PCA reconstruction.

3.1.1.2 KCCA

In this step, we are supposed to train an optimal kernel function that maximizes the coherence between the pair of principal components z_x and z_y. Meanwhile, we also obtain the canonical variables u and v, corresponding to z_x and z_y.

To ensure that for any scalar $z_{x,i}$ in $z_x = (z_{x,1}, z_{x,2}, ..., z_{x,k})$ and $z_{y,j}$ in $z_y = (z_{y,1}, z_{y,2}, ..., z_{y,k})$, the kernel function $K(z_{x,i}, z_{y,j})$ always represents an inner product in any Hilbert space, we need to make sure it is positive definite. One way to satisfy this constraint is to apply Schoenberg's theorem: if $f(t)$ is a completely monotonic function, then the radial kernel $K(z_{x,i}, z_{y,j}) = f(\|z_{x,i} - z_{y,j}\|^2)$ is positive definite on any Hilbert space. We use one of the simplest completely

monotonic function

$$f(t) = e^{-\alpha t + \beta}, \alpha > 0 \tag{3.1}$$

$$K(z_{x,i}, z_{y,j}) = e^{-\alpha \|z_{x,i} - z_{y,j}\|^2 + \beta} \tag{3.2}$$

with two parameters α and β.

To maximize the non-linear coherence of the input pair of data, we set non-linear dependence as the cost function in the training process. Summarizing [5], we extract the following formulas in chapter 2. L and H represent nonlinear dependence and coherence between z_x and z_y.

$$L = det(I - \Sigma\Sigma^T) = \prod_{i=1}^{m}(1 - \sigma_i^2); 0 \geq L \geq 1 \tag{3.3}$$

$$L = \frac{det(R_{zz})}{det(R_{z_x z_x})det(R_{z_y z_y})} = \frac{det\left(\begin{bmatrix} R_{z_x z_x} & R_{z_x z_y} \\ R_{z_y z_x} & R_{z_y z_y} \end{bmatrix}\right)}{det(R_{z_x z_x})det(R_{z_y z_y})} \tag{3.4}$$

$$H = 1 - L = 1 - \frac{det\left(\begin{bmatrix} R_{z_x z_x} & R_{z_x z_y} \\ R_{z_y z_x} & R_{z_y z_y} \end{bmatrix}\right)}{det(R_{z_x z_x})det(R_{z_y z_y})} \tag{3.5}$$

One problem of the above formulas is that they are sensitive to high-dimensional space. If one σ_i is large, the result of dependence formula tends to 0 and coherence tends to 1. As a result, the coherence of z_x and z_y tends to small if any z_x is predictable for any z_y or any z_y is predictable for any z_x. On the other hand, as we mentioned, if a high-dimensional matrix is singular, we cannot calculate its determinant. To fix these issues, we modify the above formulas.

$$L_{i,j} = \frac{det\left(\begin{bmatrix} R_{z_{x,i} z_{x,i}} & R_{z_{x,i} z_{y,j}} \\ R_{z_{y,j} z_{x,i}} & R_{z_{y,j} z_{y,j}} \end{bmatrix}\right)}{det(R_{z_x z_x})det(R_{z_y z_y})} = \frac{R_{z_{x,i} z_{x,i}} R_{z_{y,j} z_{y,j}} - R_{z_{x,i} z_{y,j}} R_{z_{y,j} z_{x,i}}}{R_{z_x z_x} R_{z_y z_y}} \tag{3.6}$$

Here $C_{i,j}$ represents the predictability between this specific pair of points, $z_{x,i}$ and $z_{y,j}$. Now we have

$$L = \frac{\sum_{i,j=1}^{k}(L_{i,j})^2}{n^2} \tag{3.7}$$

$$C = 1 - L = 1 - \frac{\sum_{i,j=1}^{k}(L_{i,j})^2}{n^2} \tag{3.8}$$

By our new formula, the novel coherence C is small if and only if all z_x is predictable from z_y and all z_y is predictable from z_x. The connection between X and Y comes from coherence of z_x and z_y in the reproducing kernel Hilbert space. We define κ as the kernel function training process, then we have

$$\kappa = \underset{\alpha,\beta}{\operatorname{argmin}} L \tag{3.9}$$

Once the cost converges to a small value, we obtain the parameters α and β and the corresponding kernel function $K(z_{x,i}, z_{y,j}) = e^{-\alpha\|z_{x,i}-z_{y,j}\|^2+\beta}$.

We get the covariance matrix R_{zz} by 2.13 and get the canonical variables u and v by 2.3 - 2.6.

3.1.2 Decoder Phase

3.1.2.1 Linear ANNs

The first step for the decoder phase is to transfer the canonical variables u and v to the spaces of principle components z_y and z_x respectively. Here we use 2 two-layer linear ANNs to map u and v to $z_y{'}$ and $z_x{'}$. The two ANNs are namely γ_1 and γ_2:

$$
\begin{aligned}
z_y^{'} &= W_{1,2}(W_{1,1}u + b_{1,1}) + b_{1,2} \\
z_x^{'} &= W_{2,2}(W_{2,1}v + b_{2,1}) + b_{2,2} \\
\gamma_1(W_{1,1}, W_{1,2}, b_{1,1}, b_{1,2}) &= \underset{W_{1,1},W_{1,2},b_{1,1},b_{1,2}}{\operatorname{argmin}} \left\| z_y - z_y^{'} \right\|^2 \\
\gamma_2(W_{2,1}, W_{2,2}, b_{2,1}, b_{2,2}) &= \underset{W_{2,1},W_{2,2},b_{2,1},b_{2,2}}{\operatorname{argmin}} \left\| z_x - z_x^{'} \right\|^2
\end{aligned}
\tag{3.10}
$$

γ_1 and γ_2 provide us $z_y^{'}$ and $z_x^{'}$ which have minimal difference with z_y and z_x respectively.

3.1.2.2 PCA Reconstruction

Then we need an operation that maps the principal components z_x and z_y back to the original feature space of y and x. Our solution is PCA reconstruction, which can be thought as the "reverse" PCA that maps principal components to high-dimensional feature space.

$$PCA\ reconstruction = PC\ scores \cdot Eigenvectors^T + Mean \tag{3.11}$$

Eigenvectors map the principal components $z \in \mathbb{R}^k$ back to the original feature space $x^{'} \in \mathbb{R}^n$. We make use of the eigenvectors and means of y and x that we obtain in the first PCA step

and follow 3.11 to get the PCA reconstruction.

$$y'' = z'_y \cdot Eigenvectors_y^T + Mean_y$$
$$x'' = z'_x \cdot Eigenvectors_x^T + Mean_x$$

(3.12)

3.1.2.3 Non-linear ANNs

From experimental results, we found that the results of PCA reconstruction, y'' and x'', though in the feature space of y and x, are not precise enough to be the final results. On the one hand, training errors are unavoidable in ANNs. The previous training processes are all executed to principal components in the lower-dimensional spaces $\mathbb{R}^k$. The last mappings from $\mathbb{R}^k$ to $\mathbb{R}^n$ amplify the dimensions of data as well as the reconstruction errors. On the other hand, we mentioned that the k principal components explain over 90% variances. This means that during PCA and PCA reconstruction, a small part of the information from the original data gets lost. For these two reasons, we need one more training step where the errors are computed in the feature spaces of original data sets to enhance the performance of our model.

This training process for our model is 2 two-layer non-linear encoders mapping y'' to y' and x'' to x' respectively. It minimizes the error between y' and original data y and error between x' and original data x. Slightly different from typical autoencoders that are usually trained to minimize reconstruction errors (such as squared errors) between input data and its own reconstructed data, our encoders are trained to minimize reconstruction error between y and y' and reconstruction error between x and x' (note that the input data of this step are not y and x, but y'' and x''). We define the two ANNs as $\mathcal{E}_1$ and $\mathcal{E}_2$:

$$y' = W_{1,3}(W_{1,4}y'' + b_{1,3}) + b_{1,4}$$
$$x' = W_{2,3}(W_{2,4}x'' + b_{2,3}) + b_{2,4}$$
$$\mathcal{E}_1(W_{1,3}, W_{1,4}, b_{1,3}, b_{1,4}) = \underset{W_{1,3},W_{1,4},b_{1,3},b_{1,4}}{\text{argmin}} \|y - y'\|^2$$
$$\mathcal{E}_2(W_{2,3}, W_{2,4}, b_{2,3}, b_{2,4}) = \underset{W_{2,3},W_{2,4},b_{2,3},b_{2,4}}{\text{argmin}} \|x - x'\|^2$$

(3.13)

This is the last step of the whole model. Once the training errors converge, we obtain reconstruction y' and x' and evaluate the model performance.

3.2 Training Algorithm

In section 3.1, we mentioned three deep learning training processes, a kernel function optimization training κ (3.1.2), canonical variable to principal component mappings γ (3.1.3) and the encoders $\mathcal{E}$ (3.1.4). We use gradient descent optimization [29] as our training algorithm in all these three training processes. Gradient descent is one of the most popular algorithms to perform optimization and by far the most common way to optimize artificial neural networks. Although this algorithm is often used as a black-box optimizer for developers, we briefly introduce one of its variants, batch gradient descent which is used by TensorFlow class *tf.train.GradientDescentOptimizer* [28], to help explain our training algorithm. In code, batch gradient descent looks something like this:

$$
\begin{aligned}
&for\ i\ in\ range(\#\ epochs): \\
&\quad params\_grad = evaluate\_gradient(cost\_function, data, params) \\
&\quad params = params - learning\_rate * params\_grad
\end{aligned}
\tag{3.14}
$$

For example, for the following optimization ϕ with parameters W and b,

$$
\begin{aligned}
h &= f(Wx + b) \\
cost &= \|x - x'\|_2 \\
\phi &= \underset{W,b}{\operatorname{argmin}}\ cost
\end{aligned}
\tag{3.15}
$$

In each step (epoch) i, batch gradient descent computes the gradient of the cost function with respect to the parameters for the entire training dataset:

$$
\begin{aligned}
G(W_i) &= \frac{\partial\ cost}{\partial\ h}\ \frac{\partial\ h}{\partial f(W_i)}\ \frac{\partial f(W_i)}{dW_i} \\
G(b_i) &= \frac{\partial\ cost}{\partial\ h}\ \frac{\partial\ h}{\partial f(b_i)}\ \frac{\partial f(b_i)}{db_i} \\
W_{i+1} &= W_i - \alpha G(W_i) \\
b_{i+1} &= b_i - \alpha G(b_i)
\end{aligned}
\tag{3.16}
$$

where α refers to learning rate and $G()$ refers to the gradient of a parameter. In TensorFlow implementation, we use a one-line code

optimizer = tf.train.GradientDescentOptimizer(learning_rate).minimize(cost)

By this class, we only need to define the learning rate and cost function for a training process. We set learning rate to 0.01 for all three training processes and the cost functions are described in the previous section.

3.3   Evaluation Metrics

To evaluate the reconstruction results, we need to specify one or multiple metrics. For image data, for example, the most direct way to check reconstruction performance is to convert the output data back to original canvas size and print the image with reshaped data. Then we can evaluate the reconstruction visually by comparing the reconstruction image with original image. Usually this is a very effective approach. However, it is not guaranteed that human eyes are able to identify all trivial differences between two images, and it is hard to claim which model works better if two models output similar reconstruction images, or demonstrate visually indistinguishable levels of performance. Thus it is necessary to introduce some quantitative measures.

Our purpose here is to compare the reconstructed image data and original image data by quantifying the error, the distance or the similarity between them. This is very equivalent to the Image Similarity Analysis [30, 31, 34]. We decide to apply 5 different metrics: L^2-norm, Pearson Correlation score, cross-correlation, Bhattacharyya distance and Fast Fourier Transform (FFT) rank.

3.3.1 L^2-Norm

For real vectors $x = [x_1, x_2, ...x_n]$, L^2-Norm [32] is given by

$$\|x\|_2 = \sqrt{\sum_{i=1}^{n} x_i^2} \tag{3.17}$$

The L^2-Norm is also known as the Euclidean norm because it measures the distance in Euclidean space. For output data vector x' and y', we can measure their reconstruction error by calculating

L^2-Norm of $x - x'$ and $y - y'$ respectively.

$$\|x - x'\|_2 = \sqrt{\sum_{i=1}^{n}(x_i - x'_i)^2}$$
$$\tag{3.18}$$
$$\|y - y'\|_2 = \sqrt{\sum_{i=1}^{n}(y_i - y'_i)^2}$$

Note that L^2-Norm works for data as vectors. After reshaping the data to matrix form, the measure turns to Frobenius norm [32]. But the result will not change.

3.3.2 Pearson Correlation Score

Pearson Correlation Coefficient [30, 33] is a measure of the linear correlation between two variables x and y. It has a value between +1 and 1, where 1 is total positive linear correlation, 0 is no linear correlation, and 1 is total negative linear correlation. This metric measures how highly correlated are two variables and is measured from -1 to +1. A Pearson Correlation Coefficient of 1 indicates that the data objects are perfectly correlated but in this case, a score of -1 means that the data objects are not correlated. In other words, the Pearson Correlation score quantifies how well two data objects fit a line.

In essence, the Pearson Correlation score finds the ratio between the covariance and the standard deviation of both objects. In our case, we calculate the Pearson Correlation score between x and x', y and y' to demonstrate how correlated they are:

$$Pearson(x, x') = \frac{\sum xx' - \frac{\sum x \sum x'}{n}}{\sqrt{(\sum x^2 - \frac{(\sum x)^2}{n})(\sum x'^2 - \frac{(\sum x')^2}{n})}}$$
$$\tag{3.19}$$
$$Pearson(y, y') = \frac{\sum yy' - \frac{\sum y \sum y'}{n}}{\sqrt{(\sum y^2 - \frac{(\sum y)^2}{n})(\sum y'^2 - \frac{(\sum y')^2}{n})}}$$

A larger Pearson Correlation score means a larger correlation between original data and reconstructed data. Since the model generates data in the same scale with input data, a larger Pearson Correlation score here refers to greater similarity, thus better reconstruction result.

3.3.3 Cross-correlation

In signal processing, cross-correlation [34] is a measure of similarity of two series as a function of the displacement of one relative to the other. This measure has been used in medical image registration. For 1-dimensional data, this approach is essentially same as Pearson correlation coefficient score method:

$$CC(x, x') = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(x'_i - \overline{x'})}{n\sqrt{var(x)var(x')}}$$
$$CC(y, y') = \frac{\sum_{i=1}^{n}(y_i - \overline{y})(y'_i - \overline{y'})}{n\sqrt{var(y)var(y')}}$$

(3.20)

However, for 2-dimensional data, this measure has one advantage: it takes care of the data shifting just like cross-correlation method takes care of the time-delay between signals in signal processing. In our case, let $c2d()$ refers to 2-dimensional cross-correlation computation, for the reshaping data $x, y, x', y' \in \mathbb{R}^{m \times n}$,

$$CC2d_x = c2d(\frac{x - \overline{x}}{\sqrt{var(x)}}, \frac{x' - \overline{x'}}{\sqrt{var(x')}})/(m \cdot n)$$
$$CC2d_y = c2d(\frac{y - \overline{y}}{\sqrt{var(y)}}, \frac{y' - \overline{y'}}{\sqrt{var(y')}})/(m \cdot n)$$

(3.21)

Here $CC2d_x$ and $CC2d_y$ are two arrays containing cross-correlation coefficients (range -1 to 1) in different phases. We select the maximum values of them to represent the similarity between x and x' and between y and y' respectively.

3.3.4 Bhattacharyya Distance

In statistics, the Bhattacharyya distance [31, 35, 36] measures the similarity of two discrete or continuous probability distributions. In image processing, this measure can be used to determine the relative closeness of the two samples being considered. In our case [31],

$$d(x, x') = \sqrt{1 - \frac{1}{\sqrt{xx'n^2}}\sum_{i=1}^{n}\sqrt{x_i x'_i}}$$
$$d(y, y') = \sqrt{1 - \frac{1}{\sqrt{yy'n^2}}\sum_{i=1}^{n}\sqrt{y_i y'_i}}$$

(3.22)

A zero Bhattacharyya distance means that two data are exactly the same. Larger Bhattacharyya distance refers to greater gap or difference between them.

3.3.5 FFT Rank

Fourier transform [37] is an important image processing tool which is used to transform an image from spatial domain into frequency domain. A Fast Fourier transform (FFT) [38] reduces the complexity of Fourier transform from N^2 to $NlogN$. This major improvement of computational makes FFT practically possible in many applications. FFT is an algorithm that samples a signal (or signal-like data) over a period of time (or space) and transforms it into its frequency domain. In the frequency domain, each point represents a particular frequency contained in the spatial domain. The points in frequency domain have both real and imaginary parts, which represent magnitude and phase respectively. In our evaluation of reconstruction, we use only the real part (magnitude) of the FFT results, as it contains most of the information of original data in spatial domain.

In our case, we use the following rank formula as our metric for comparison. Here x, y, x', y' refer to intensity values of data (data in spatial domain), and $\overline{F}_x, \overline{F}_y, \overline{F}_{x'}, \overline{F}_{y'}$ represent the average frequency values of them (in frequency domain) [31].

$$
\begin{aligned}
rank(x, x') &= real(\frac{(\sum_{i=1}^n x_i x_i' - n\overline{F}_x \overline{F}_{x'})^2}{(\sum_{i=1}^n |x_i|^2 - n\overline{F}_x^2)(\sum_{i=1}^n |x_i'|^2 - n\overline{F}_{x'}^2)}) \\
rank(y, y') &= real(\frac{(\sum_{i=1}^n y_i y_i' - n\overline{F}_y \overline{F}_{y'})^2}{(\sum_{i=1}^n |y_i|^2 - n\overline{F}_y^2)(\sum_{i=1}^n |y_i'|^2 - n\overline{F}_{y'}^2)})
\end{aligned}
\tag{3.23}
$$

A rank ranges from -1 to 1, where 1 is obtained if two datasets are exactly the same and -1 if they are fully independent from each other. The higher the rank, the more similarity shared by the two datasets.

3.4 Comparison Models

We set the below training parameters for all three training processes in our model:

hidden layer number = 100,

learning rate = 0.01,

training epochs = 2000,

batch size = 100

From earlier experiments, we found these parameters would ensure that all the steps can be well trained. For fair comparison, in the following models designed to measure the performance of our model, we fix all these parameters in their trainings processes.

3.4.1   Comparison Model 1 - Coherence Encoder

The difference between KCE and this "Coherence Encoder (CE)" is that our KCE use KCCA to analyze mutual information between two data sets while CE use CCA instead. The purpose for this comparison is to test the effectiveness of kernel methods in the KCCA part in extracting the correlation between two data sets and the level of improvement achieved by Kernel tricks. Note that we derived the CCA calculations as well as linear coherence formula in chapter 2. This comparison model is built on them.
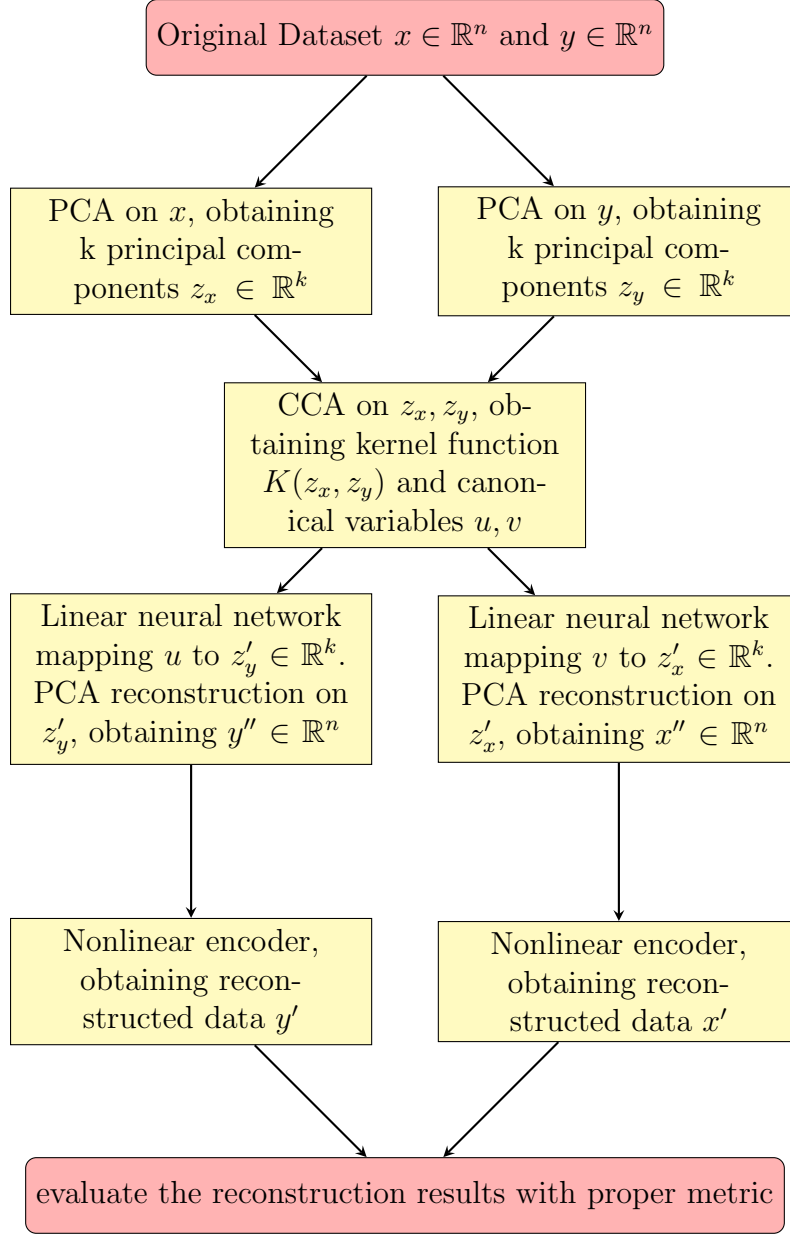
Figure 3.2: Flow chart of comparison model 1: Coherence Encoder

3.4.2 Comparison Model 2 - Encoder

This comparison model "Encoder" applies neither KCCA nor CCA. In other words, it directly maps z_x to z_y' and z_y to z_x' in the linear neural network training process, without the usage of either kernel function or canonical variables. As a result, the KCCA training procedure is eliminated from the model. The purpose for this comparison is to test the effectiveness of KCCA in extracting the correlation between two data sets and the level of improvement achieved by using canonical variables as input features.
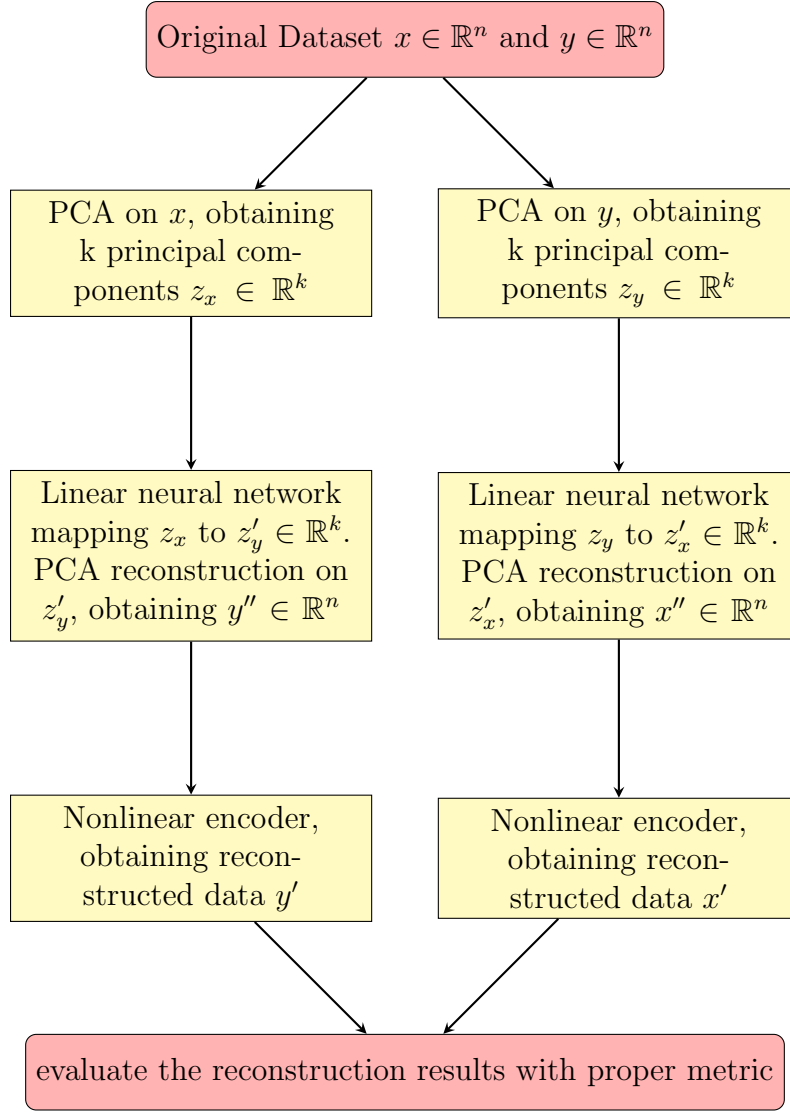
Figure 3.3: Flow chart of comparison model2: Encoder

Chapter 4

Numerical Results

4.1 Data Sets

MNIST data set [39] is a classical data set widely used for machine learning and deep learning study, especially in image processing and pattern recognition problems. With feature data (pixel values of images) and labels provided, it can be used for both supervised and unsupervised learning. The data set contains 55000 images for training and 10000 images for testing. Each image is a 28×28 pixel picture of a human hand-written digit (0-9) one 0-to-1-gray-scale canvas with pure black background (pixel value 0.0) and pure white writing (pixel value 1.0).

Figure 4.1: Part of samples of MNIST hand-written digit image database [47].

To fit our model design, we need a pair of data sets with some correlations in between. Our strategy is to cut each image (784 pixels) to upper and lower parts with equal sizes (28×14), as shown in Figure 4.2. The upper parts of the images are x, and the lower parts of the images are y. In this circumstance, x and y are correlated because their features are potentially predictable from one to another. (A human reader will somehow be able to imagine what the other part

of a hand-writing 7 or 4 or other digits looks like by looking at one part)

Note that given the labels provided by MNIST data indicating which number each image refers to, we group the images by their labels so that each group contains images corresponding to only one digit. We run our model for each group instead of running all images together. In this case, there will be greater correlation between two parts of an image because the patterns for a single digit are close even though they come from different writings.
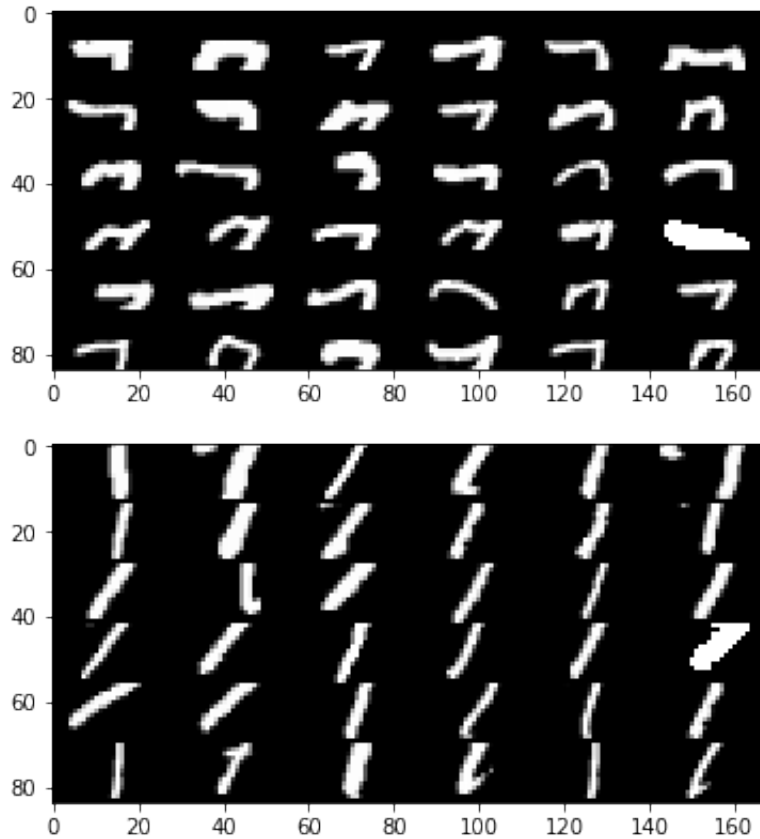

Figure 4.2: 36 examples of upper and lower parts of digit 7 images.

Based on experimental results, we set the dimension $k = 10$ for Principal Component Analysis because the first 10 principal components explain more than 90% variance of the original image data.

4.2 Reconstruction Images

4.2.1 Simple ANNs Reconstruction

Our initial trial for reconstruction is running a pair of ANNs on the MNIST data sets:
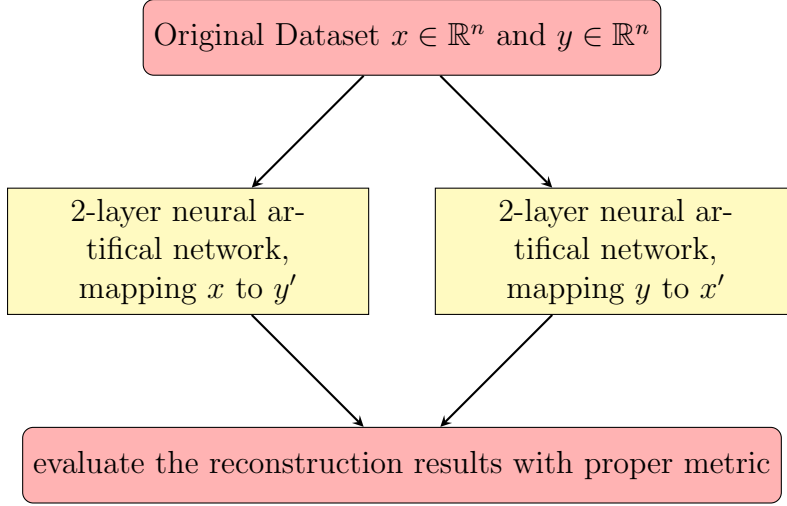
Figure 4.3: Flow chart of simple paired ANNs

The two ANNs above work independently. Each contains a simple two-layer ANN mapping x to y and y to x respectively. All layers follow the classical activation function $h = \sigma(Wx + b)$. Below are some examples of the reconstruction pictures (we randomly select 6 hand-written images for each digit. The left part of an picture is original digit image and the right part is reconstructed image). Although the ANNs reconstruct most of the patterns for these images, it is notable that there are usually some abnormal white pixels in black background. And the positions of these pixels are exactly the same in the samples of each digit. It indicates that the ANNs learn and "memorize" a map between the two datasets x and y containing these flaws. This mapping memory prevents the ANNs from perfect reconstruction. We call this issue "black-to-white" mapping errors. It is an interesting and meaningful theoretical question to explore [40], but here we are not going to explore it.
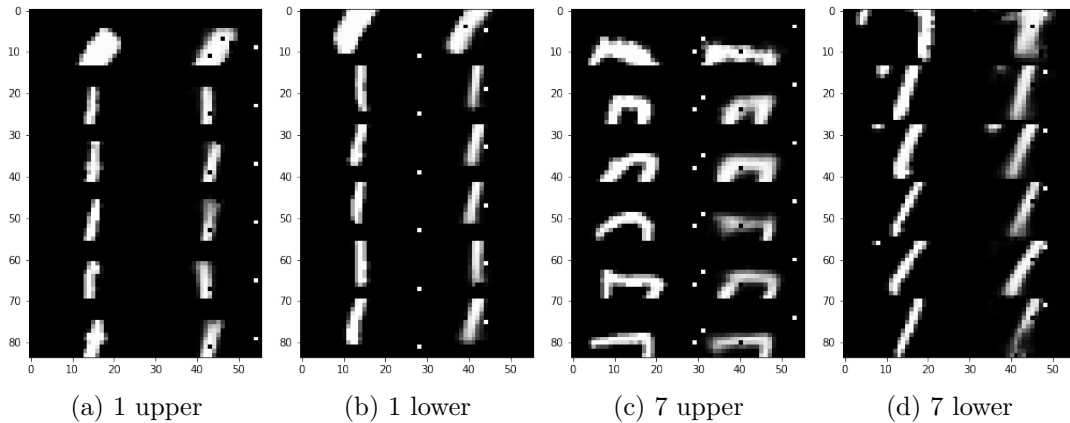

(a) 1 upper   (b) 1 lower   (c) 7 upper   (d) 7 lower

Figure 4.4: ANNs Reconstruction of digit 1 and 7

36

4.2.2 KCE, CE and Encoder Reconstruction

Digit	KCE Upper	KCE Lower	CE Upper	CE Lower	Encoder Upper	Encoder Lower
0						
1						
2						
3						
4						
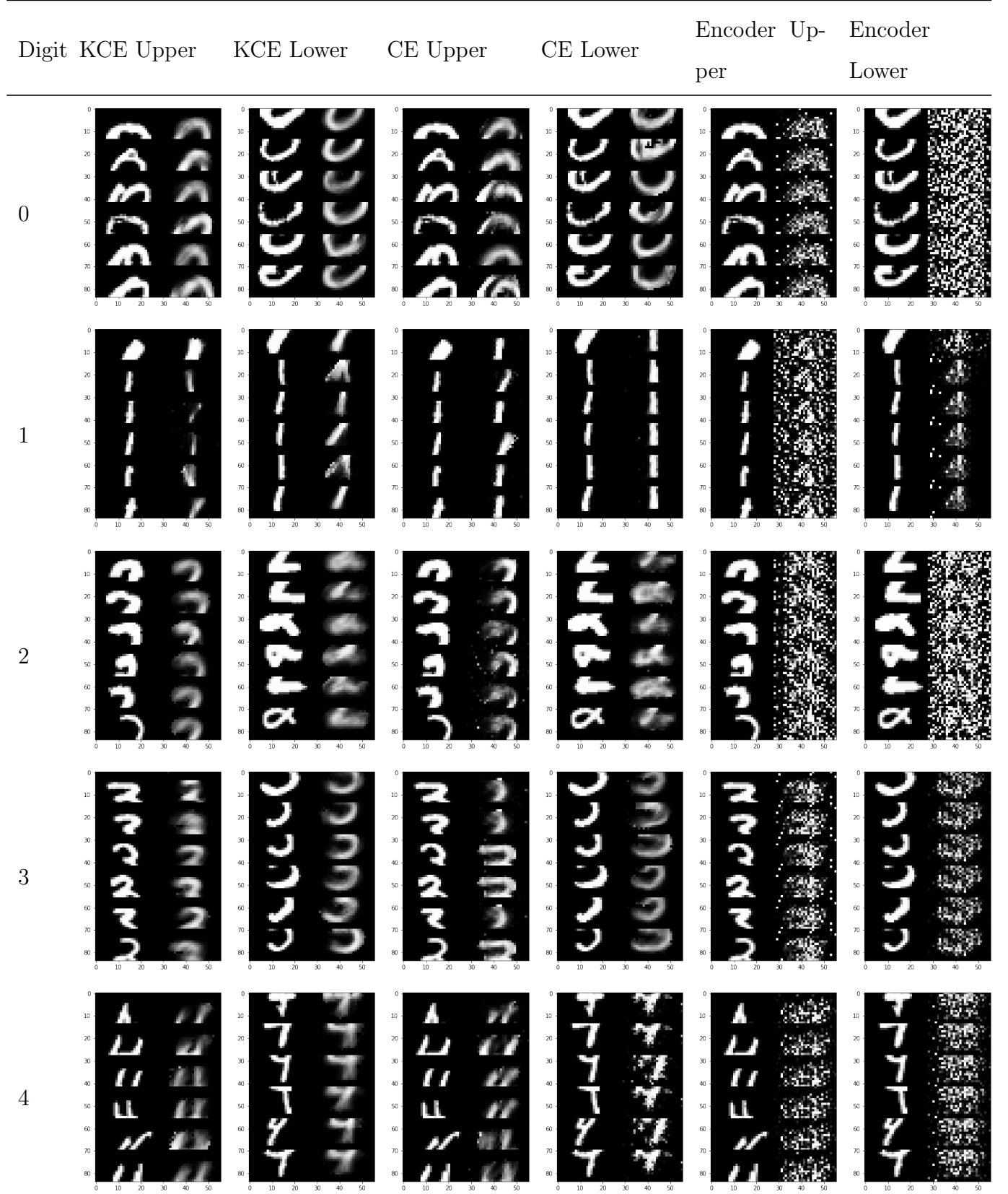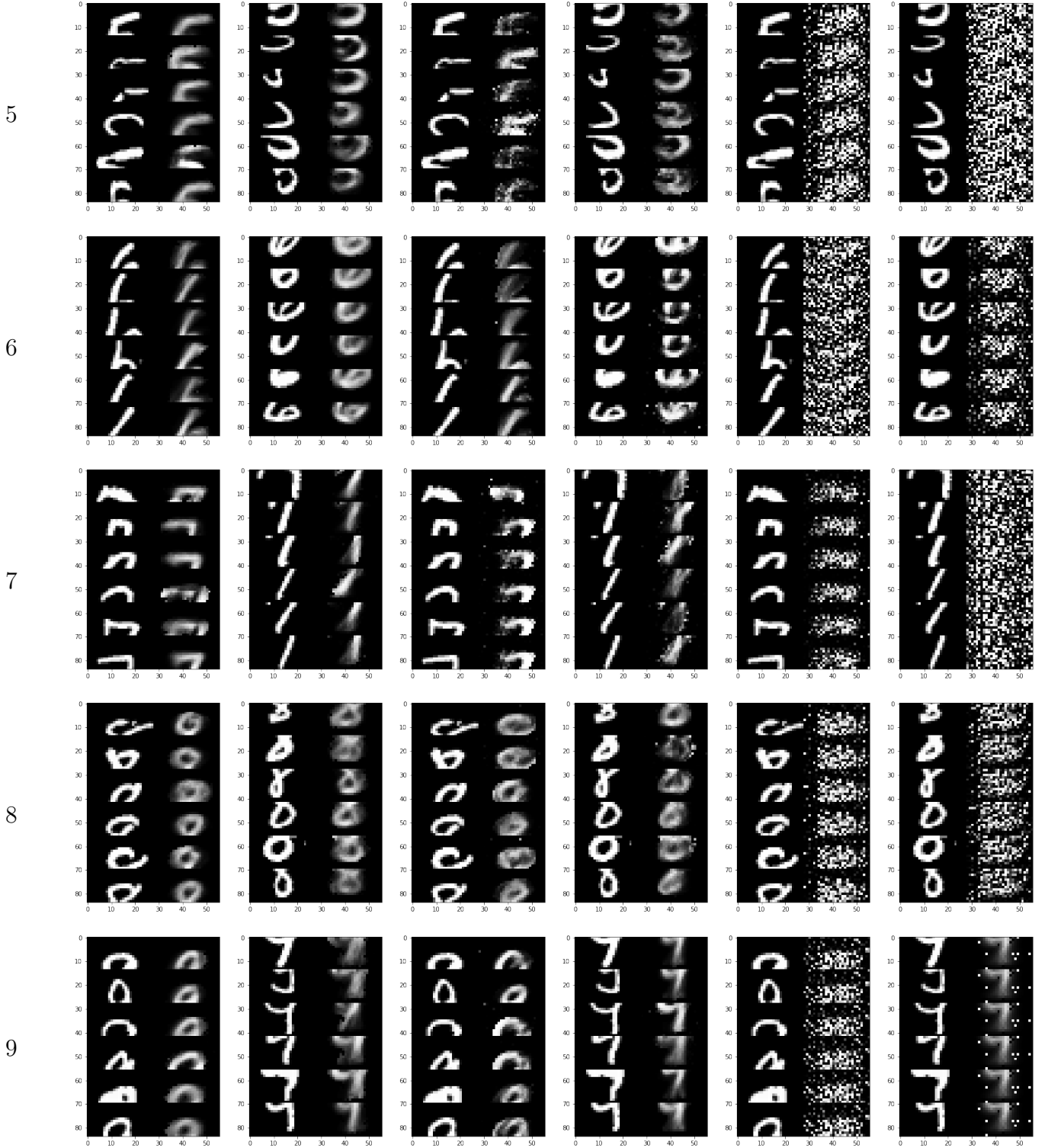
In the above table, we post all the reconstruction results of testing. In the testing part, we randomly select 6 hand-written images for each digit from test data set and run all three models on these samples. Thus in each picture, we have 6 original half-images on the left and 6 reconstructed half-images on the right. The table contains reconstruction results from KCE, comparison model 1 CE and comparison model 2 Encoder.

From the results shown in the table, we can visually evaluate the performances of the three models. First, contrasting the results of KCEA with the results of Encoder, it is obvious that KCE generates better reconstruction. This is a convincing clue to claim the effectiveness of KCCA step. With the help of KCCA, the reconstruction images become identifiable and clean. Then we compare KCE with CE, the differences are not as easily observable. The patterns of hand-written digits are generally reconstructed in both models. However, the patterns of some reconstructed images from CE, such as upper part of 5 and lower part of 4 are smashed to pieces while this never occur in KCE reconstruction. To authenticate the difference, we need to take a further step to quantitative metrics. Notably, our KCE solves the "black-to-white" mapping problems,

4.3 Quantitative Comparison

In section 3.3, we introduced 5 metrics for Image Similarity Analysis. Here we use them to quantitatively evaluate the performances of the three models. For each model, we have 6 pairs of reconstructed image data sets for each digit 0-9, thus 120 arrays of data with size 392 (28×14), the total pixel number for one half-image. We do the image similarity analysis for KCE, CE and Encoder by comparing their reconstructed data with the original data to indicate the similarities between the original images and reconstructed images. For each one of the metric, we calculate the mean of 120 scores/distances/ranks of the three models separately. Then we do T-tests [41] to compare KCE with CE and Encoder respectively. We extract p-values indicating the confidence to claim that scores/distances/ranks are different (the smaller the p-value, the more confident to claim this). The statistical analysis results are shown in Figure 4.5 and Figure 4.6.

	metric	CE average	KCE average	p-value
0	2-norm	5.416527	4.942153	0.002974
1	Pearson Coefficient	0.550588	0.618759	0.006633
2	cross correlation	0.689174	0.712149	0.113121
3	Bhattacharyya distance	0.539608	0.505529	0.027234
4	fft	0.424210	0.497200	0.003529

Figure 4.5: Statistical results between KCE and CE

	metric	Encoder average	KE average	p-value
0	2-norm	8.318797	4.933067	1.070551e-38
1	Pearson Coefficient	0.295733	0.607328	2.367107e-28
2	cross correlation	0.344092	0.711152	1.044875e-58
3	Bhattacharyya distance	0.693693	0.507269	5.284914e-29
4	fft	0.204558	0.489945	3.993558e-27

Figure 4.6: Statistical results between KCE and Encoder

In general, from the statistics provide in the tables, no matter compared with CE or Encoder, the average values of all 5 metrics show that KCE has greater performance. Meanwhile, most of (9/10) the p-values are less than 0.05, small enough to support this claim. The results demonstrate that in our reconstruction task, KCE has better performance than the two comparison models. This further proves the effectiveness of KCCA in our model.

Chapter 5

Conclusion

5.1 Contribution

In this thesis research, we build a model based on PCA, KCCA and ANN. Given two potentially correlated high-dimensional data sets, our model successfully build a map between the two data sets. Thus we can reconstruct one data set by the other and their mutual information. During the research, we first learn and explore CCA. However, we find its limitation on high dimensional data from several experiments. So we apply kernel methods. With the help of Mercer's theorem and Schoenberg's theorem, we choose the proper kernel function that can map data to RKHS. In the training processes that maps canonical variables to the original feature space of another data set, we try both linear and non-linear ANNs and figure out that a linear ANN before PCA reconstruction and a non-linear ANN after PCA reconstruction best fit our demand. In MNIST experiment, our model solves the "black-to-white" mapping error problem for gray-scale image reconstruction. Meanwhile, we prove the effectiveness of KCCA in the paired reconstruction task.

5.2 Future Work

We propose three directions of the future work:

First, the reconstruction results of our KCE are still far from perfect and need to be improved. We believe that during the PCA and PCA reconstruction, some information of original data get lost. On the one hand, we need to prove this claim. On the other hand, we should rethink about the dimensionality trade-off for better reconstruction results.

Second, we have observed that our model is able to avoid the "black-to-white" mapping error problem. It is worth for us to dig out the reason and continue to produce a mathematical proof for this.

Third, so far we have only done the experiments on MNIST data. We should explore more samples including face images, item images or even other types of data to justify the effectiveness of our model.

Bibliography

[1] Lai, Pei Ling, and Colin Fyfe. "Kernel and nonlinear canonical correlation analysis." *International Journal of Neural Systems10.05 (2000): 365-377.*

[2] Tan, Chun Chet. Autoencoder neural networks: A performance study based on image reconstruction, recognition and compression. *LAP Lambert Academic Publishing, 2009.*

[3] aymericdamien, TensorFlow-Examples, (2017), GitHub repository, *https://github.com/aymericdamien/TensorFlow-Examples*

[4] Baldi, Pierre. "Autoencoders, unsupervised learning, and deep architectures." *Proceedings of ICML Workshop on Unsupervised and Transfer Learning. 2012.*

[5] Pezeshki, A., Scharf, L. L., Azimi-Sadjadi, M. R., & Lundberg, M. (2004, November). Empirical canonical correlation analysis in subspaces. *In Signals, Systems and Computers, 2004. Conference Record of the Thirty-Eighth Asilomar Conference on (Vol. 1, pp. 994-997). IEEE.*

[6] Wold, Svante, Kim Esbensen, and Paul Geladi. "Principal component analysis." *Chemometrics and intelligent laboratory systems 2.1-3 (1987): 37-52.*

[7] Pearson, K. (1901). "On Lines and Planes of Closest Fit to Systems of Points in Space". *Philosophical Magazine. 2 (11): 559572.*

[8] amoeba (https://stats.stackexchange.com/users/28666/amoeba), How to reverse PCA and reconstruct original variables from several principal components?, URL (version: 2016-08-12): *https://stats.stackexchange.com/q/229092*

[9] Autoencoder Using Kernel Methods *http://technodocbox.com/3D_Graphics/66255915-Autoencoder-using-kernel-method.html*

[10] Gretton, Arthur. "Introduction to rkhs, and some simple kernel algorithms." *Adv. Top. Mach. Learn. Lecture Conducted from University College London (2013): 16.*

[11] Gram matrix. Encyclopedia of Mathematics.
URL:*http://www.encyclopediaofmath.org/index.php?title=Gram_matrix&oldid=35177*

[12] Weisstein, Eric W. "Singular Matrix." From MathWorld–A Wolfram Web Resource. *http://mathworld.wolfram.com/SingularMatrix.html*

[13] Hrdle, Wolfgang; Simar, Lopold (2007). "Canonical Correlation Analysis". *Applied Multivariate Statistical Analysis. pp. 321330. doi:10.1007/978-3-540-72244-1_14. ISBN 978-3-540-72243-4.*

[14] Abadi, Martn, et al. "TensorFlow: A System for Large-Scale Machine Learning." *OSDI. Vol. 16. 2016.*

[15] "Credits". *TensorFlow.org.* Retrieved November 10, 2015.

[16] "TensorFlow Release". Retrieved February 28, 2018.

[17] James, Gareth, et al. An introduction to statistical learning. *Vol. 112. New York: springer, 2013.*

[18] Theodoridis, Sergios (2008). Pattern Recognition. *Elsevier B.V. p. 203. ISBN 9780080949123.*

[19] Hein, Matthias, and Olivier Bousquet. "Hilbertian metrics and positive definite kernels on probability measures." *AISTATS. 2005.*

[20] Mercer, J. (1909), "Functions of positive and negative type and their connection with the theory of integral equations", *Philosophical Transactions of the Royal Society A, 209 (441458): 415446*

[21] Miller, Kenneth S., and Stefan G. Samko. "Completely monotonic functions." *Integral Transforms and Special Functions 12.4 (2001): 389-402.*

[22] Schoenberg Ann. of Math. textit39 (1938), 811-841)

[23] Deng, Li, and Dong Yu. "Deep learning: methods and applications." *Foundations and Trends in Signal Processing 7.34 (2014): 197-387.*

[24] Schmidhuber, Jrgen. "Deep learning in neural networks: An overview." *Neural networks 61 (2015): 85-117.*

[25] Wang, Sun-Chong. "Artificial neural network." *Interdisciplinary computing in java programming. Springer, Boston, MA, 2003. 81-100.*

[26] Maxwell, Scott E., Harold D. Delaney, and Ken Kelley. Designing experiments and analyzing data: A model comparison perspective. *Routledge, 2017.*

[27] Schlkopf, Bernhard, Alexander Smola, and Klaus-Robert Mller. "Nonlinear component analysis as a kernel eigenvalue problem." *Neural computation 10.5 (1998): 1299-1319.*

[28] Tensorflow Class GradientDescentOptimizer, URL:*https://www.tensorflow.org/api_docs/python/tf/train/GradientDescentOptimizer*

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

[30] Segaran, Toby. Programming Collective Intelligence: Building Smart Web 2.0 Applications. *Sebastopol, CA: O'Reilly Media, 2007.*

[31] Narayanan, Siddharth, and P. K. Thirivikraman. "IMAGE SIMILARITY USING FOURIER TRANSFORM." *Journal Impact Factor 6.2 (2015): 29-37.*

[32] Gradshteyn, I. S. and Ryzhik, I. M. Tables of Integrals, Series, and Products, 6th ed. *San Diego, CA: Academic Press, pp. 1114-1125, 2000.*

[33] "SPSS Tutorials: Pearson Correlation". Retrieved 2017-05-14.

[34] Avants, Brian B., et al. "Symmetric diffeomorphic image registration with cross-correlation: evaluating automated labeling of elderly and neurodegenerative brain." *Medical image analysis 12.1 (2008): 26-41.*

[35] Bhattacharyya, A. (1943). "On a measure of divergence between two statistical populations defined by their probability distributions". *Bulletin of the Calcutta Mathematical Society.*

[36] Guorong, Xuan, Chai Peiqi, and Wu Minhui. "Bhattacharyya distance feature selection." *Pattern Recognition, 1996., Proceedings of the 13th International Conference on. Vol. 2. IEEE, 1996.*

[37] Bracewell, Ronald Newbold, and Ronald N. Bracewell. The Fourier transform and its applications. *Vol. 31999. New York: McGraw-Hill, 1986.*

[38] Bergland, G. D. "A Guided Tour of the Fast Fourier Transform." *IEEE Spectrum 6, 41-52, July 1969.*

[39] Tensorflow Digit Dataset *https://www.tensorflow.org/get_started/mnist/beginners*

[40] Zhang, Chiyuan, et al. "Understanding deep learning requires rethinking generalization." *arXiv preprint arXiv:1611.03530(2016).*

[41] Devore, Jay, Nicholas Farnum, and Jimmy Doi. Applied statistics for engineers and scientists. *Nelson Education, 2013.*

[42] Picture resource by Bioinformatics Lab, ICGEB, New Delhi, *http://bioinfo.icgeb.res.in/lipocalinpred/algorithm.html*

[43] Picture resource by Glosser.ca - Own work, Derivative of File:Artificial neural network.svg, CC BY-SA 3.0, *https://commons.wikimedia.org/w/index.php?curid=24913461*

[44] Picture resource by Qef (talk) - Created from scratch with gnuplot, Public Domain, *https://commons.wikimedia.org/w/index.php?curid=4310325*

[45] Picture resource by Chervinskii - Own work, CC BY-SA 4.0, *://commons.wikimedia.org/w/index.php?curid=45555552*

[46] Picture resource by Abadi, Martn, et al. "TensorFlow: A System for Large-Scale Machine Learning." *OSDI. Vol. 16. 2016.*

[47] Picture resource by Cheng, Chunling, et al. "A multilayer improved RBM network based image compression method in wireless sensor networks." *International Journal of Distributed Sensor Networks 12.3 (2016): 1851829.*