POLITECNICO DI TORINO

Master degree course in Computer Engineering, Data Analytics

Master's Degree Thesis

Integrating Deep Metric Learning into Predictive Frameworks

Supervisors

Prof. Mahdi KHODAYAR

Prof. Andrea BOTTIONO

Candidate

Mahtab NIKNAHAD

October 2022

Summary

One the most important topics in technology is machine learning and its applications. machine learning methods are used to understand different concepts and make decisions to improve the performance. One of the famous subsets of machine learning is Deep learning. Deep learning is the way of creating an abstract hierarchical representation of the raw input data to extract useful features for the traditional machine learning algorithms. Arising deep learning approaches have improved many real world applications and problems in different areas such as computer vision, forecasting future events, text recognition, and etc. In this study, a deep generative model is proposed to predict the future probability of the Solar Power in the United State sites. we propose a scalable algorithm/framework which is able to capture the distribution of each power number in the historical data. the probabilistic data generation model is devised based on the feature extraction function in which the Kernel metric learning is used, deep learning, and the conditional variational autoencoder. We apply the model to the problem, the probabilistic solar power prediction. We use the Solar Power datasets obtained from Oklahoma and eastern states located in the United States. Using our proposed model, the distribution of future Solar power given historical power observations is estimated for every locations. Numerical results on the National Solar Power Database show state-of-the-art performance for probabilistic power prediction on geographically distributed power data in terms of Mean Absolute Error and RMSE has a great improvement in comparison to other solutions.

Acknowledgements

I would like to express my sincere gratitude to my supervisors Professor Mahdi Khodayar and Professor Andrea Bottino for giving me the opportunity to further my research under their supervisions.

My gratitude and love belongs to my parents, Mojtaba and Mahnaz, and my sister, Bahar for their support and love during the whole years of my life.

I would like to thank my dear roommate and friend, Fereshteh, for all of her kindness and helps during the first year of my master's degree.

I would like to thank dear Jackie Evans, my kind and nice landlord in Tulsa and her nice and talented daughters, Kayla and Meriden, for all of her efforts to boost my mood during my stay in Tulsa.

I would like to thank my dear friend Elisa, for her kindness and beautiful heart.

I would like to thank my close friends, Farzaneh, Maryam and Sara for listening to me in my darkest time and being there for me even from miles away.

I would like to thank my office mates at the university of Tulsa, Mohsen Saffari, Mukesh Yadav and Jacob Regan for helping me during the procedure of completing my thesis.

Table of Contents

List of Tables VII

List of Figures VIII

1 Introduction 1
- 1.1 Overview . . . 1
- 1.2 Generative models . . . 2
- 1.3 Deep Metric Learning . . . 2
- 1.4 Thesis Objectives . . . 3
- 1.5 Thesis Structure . . . 3

2 Literature Review 5
- 2.1 Metric Learning . . . 5
- 2.2 Deep Metric Learning . . . 7
- 2.3 Loss function for Deep metric learning . . . 11
  - 2.3.1 Siamese Network . . . 11
  - 2.3.2 Triplet Network . . . 12
- 2.4 Autoencoders . . . 13
- 2.5 Variational Autoencoders . . . 14
- 2.6 Condtional Variational Autoencoders . . . 15
- 2.7 Long Short-term memory Architecture . . . 16

3 Data and Model 19
- 3.1 What is Solar Energy . . . 19
- 3.2 Data . . . 20
- 3.3 Data Pre-processing . . . 20
- 3.4 Proposed Model . . . 21
- 3.5 Kernel Regression Problem . . . 21
- 3.6 Time Series Approximation by PDF learning . . . 23
- 3.7 Error Function . . . 25
- 3.8 Learning Algorithm . . . 25

4 Implementation and Results 27
4.1 Pytorch Framework . . . 27
4.2 Implementation . . . 28
4.3 Hyperparameters . . . 30
4.3.1 Learning Rate . . . 30
4.3.2 Window Length . . . 30
4.4 Evaluation Metric . . . 30
4.4.1 Root Mean Square Error . . . 30
4.4.2 Mean Absolute Error . . . 31
4.5 Training the model . . . 31
4.5.1 Model 1 . . . 31
4.5.2 Model 2 . . . 31
4.5.3 Model 3 . . . 31
4.6 Comparison with other methods . . . 31
4.6.1 Convolutional Graph Autoencoder . . . 31
4.6.2 Deep Convolutional Graph Rough Auto-encoder . . . 33

5 Future Work 35

6 Conclusion 36

A Appendix A 37
A.1 Implemented Code . . . 37

Bibliography 41

List of Tables

3.1 Input data from one site in Oklahoma State 21

4.1 Evaluation of the best trained model 32

List of Figures

1.1 Generative Models 2
1.2 deep metric model 3

2.1 Deep Metric Learning 6
2.2 Distance Relationship for a Siamese Network (a) Desired handwritten data discrimination for three and eight digits (b) after Siamese network applied to MNIST data for three and eight dig 10
2.3 The Siamese and Triplet Network 12
2.4 Metric Loss functions 13
2.5 Auto encoder 14
2.6 Variational Auto encoder 15
2.7 Conditional Variational Auto encoder 16
2.8 LSTM architecture 18

3.1 Photo of sample Solar site 20
3.2 An illustration of kernel regression under varying distance metrics. The color of the points represents the function value. The circle shows the radius that encapsulates 95the weights. The function value is estimated at a test point at the center. Left: With the use of the Euclidean Distance metric, the kernel is spherical and ignores the present structure that points along the diagonal share similar function values. Right: After training, under the Mahalanobis metric, the kernel function follows the direction of the target function. The radius has also shrunk and has therefore adapted to the relatively densely sampled and noiseless training samples. identity matrix. If local minima are a concern, one can use several runs with different random initializations and choose the outcome with minimum training error L. 24

4.1 Pytorch cycle 28
4.2 Network graph in Pytorch 29

4.3 Implemented Architecture 29
4.4 Model 1 learning curve 32
4.5 Model 2 learning curve 33
4.6 Model 3 learning curve 34

Chapter 1

Introduction

1.1 Overview

One of the most important topics in technology is machine learning and its applications. machine learning methods are used to understand different concepts and make decisions to improve the performance. One of the famous subsets of machine learning is Deep learning. Deep learning is the way of creating an abstract hierarchical representation of the raw input data to extract useful features for the traditional machine learning algorithms. Arising deep learning approaches have improved many real world applications such as computer vision, speech recognition, and Text recognition.
On the other hand, with the significant increase of using solar power energy all around the world, predicting the amount of this energy for next time period is a critical and hard task, this task is categorized as a time series problem. Recently, different deep neural networks have been developed and used successfully to produce the most accurate outcome for the time series prediction. These networks are often trained using RNNs and LSTMs. Moreover, combining generative models with the mentioned architectures have obtained better improvements for the time series problem and its density estimation.
In recent publications, detecting the similarity between extracted features in deep learning has increased the final accuracy in classification tasks in Face recognition and face verification applications. Applying similarity metrics for extracting useful features in time series problems is a new challenge and the result should be analyzed and studied.

1.2 Generative models

Deep generative models (DGM) are neural networks with many hidden layers trained to approximate complicated, high-dimensional probability distributions using a large number of samples. When trained successfully, we can use the DGMs to estimate the likelihood of each observation and to create new samples from the underlying distribution. Developing DGMs has become one of the most hotly researched fields in artificial intelligence in recent years. The literature on DGMs has become vast and is growing rapidly. Some advances have even reached the public sphere, for example, the recent successes in generating realistic-looking images, voices, or movies; so-called deep fakes [18].
Normalizing flow, Variational Autoencoders and the Generative adversarial Networks are the most popular generative models.

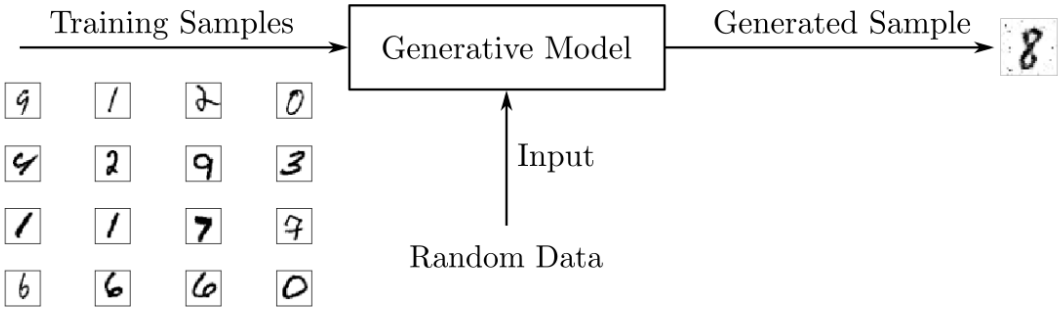

Figure 1.1: Generative Models

1.3 Deep Metric Learning

Metric learning is an approach based directly on a distance metric that aims to establish similarity or dissimilarity between objects. While metric learning aims to reduce the distance between similar objects, it also aims to increase the distance between dissimilar objects. For this reason, there are approaches, such as k-nearest neighbors, which calculate distance information, and approaches where the data is transformed into a new representation. While the metric learning approaches are moved to the transformation space with distance information, the method is basically based on a W projection matrix. Current studies are directly related to Mahalanobis distance in general [4–6]. When Mahalanobis distance is transformed into the Euclidean distance, the metric learning approach is presented based on the decomposition of the covariance matrix and the use of symmetric positive definite

matrices while performing these operations.

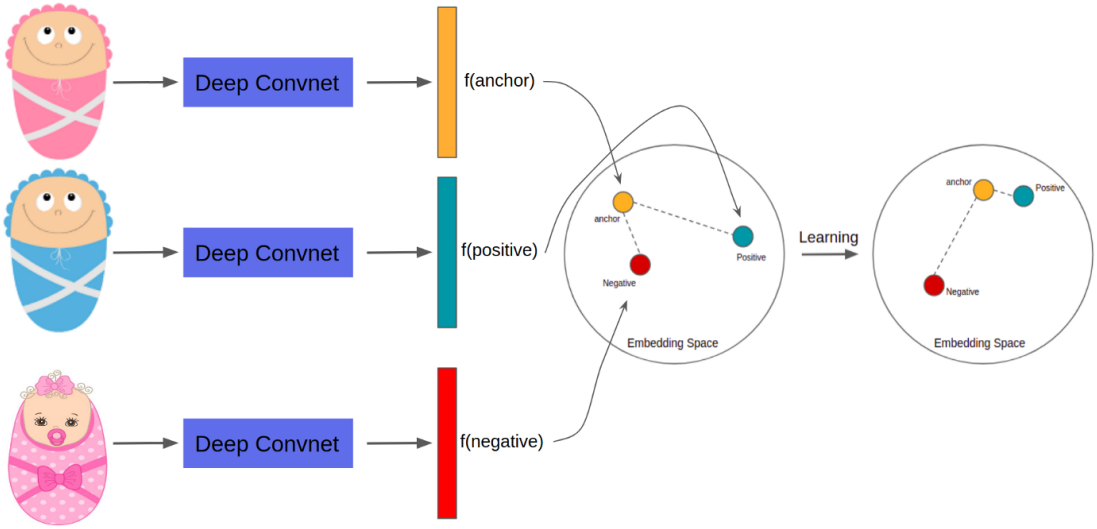

Figure 1.2: deep metric model

1.4 Thesis Objectives

Current research took place in the Software Engineering Lab at the University of Tulsa under the supervision of Prof. Mahdi Khodayar. The goal of this study is to integrate deep distance metric learning into predictive models and propose a new algorithm to improve the prediction result of the Solar Power dataset for the different future time periods in terms of Mean Absolute Error (MAE) and Mean Squared Error (MSE). the application is applied on Oklahoma's solar power sites. Moreover, we will use this frameworks as a probability density function and compare its result based on the Sharpness and Reliability metrics.

1.5 Thesis Structure

The rest of the thesis is organized as follows. Chapter 2 provides literature review about recent researches on Solar forcasting and related deep learning architectures. Deep metric learning method, Variational Autoencoders and LSTMs architectures are studied. Later, other existing frameworks as a probability density function and solar forcasting frameworks are analyzed to be compared with the final result of the proposed algorithm. Description of data and tools used during this research are presented in chapter 3.
Chapter 4 discusses data exploration and the proposed algorithm. Performance

metrics used during this research are introduced. Hyperparameter tuning is used to find the best set of hyperparameters. Results obtained from different hyperparameters are compared and all the results and its interpretation are presented, After finding the best hyperparameters for this faramework, the result of comparing this method with other existing frameworks with respect to the specific metrics are shown.
Finally, chapter 5 concludes the thesis research by pointing out the findings and outcome of this work. Moreover, possibile improvements are proposed for further research.

Chapter 2

Literature Review

This chapter is specified to the study of deep metric learning approaches, generative models and deep learning architectures which are used for forcasting future events. These architectures include Long-Short term memory (LSTM) and Recurrent Neural Network (RNN).

2.1 Metric Learning

Metric learning aims to measure the similarity among samples while using an optimal distance metric for learning tasks. Metric learning methods, which generally use a linear projection, are limited in solving real-world problems demonstrating non-linear characteristics. Kernel approaches are utilized in metric learning to address this problem.

Each dataset has specific problems in terms of classification and clustering. Distance metrics that do not have a good learning ability independent of the problem can be claimed not to yield successful results in the classification of data. Therefore, a good distance metric is required to achieve successful results on the input data [23][27].

To address this problem, several works have been conducted while using metric learning approaches [23][27]. Metric learning provides a new distance metric by analyzing data.

A metric learning approach that performs the learning process on the data will have a higher ability to distinguish the sample data. The main purpose of metric learning is to aim to learn a new metric to reduce the distances between samples of the same class and increase the distances between the samples of different class [6]. As can be seen in Figure 1c, while metric learning aims to bring similar objects closer, it increases the distance between dissimilar objects.

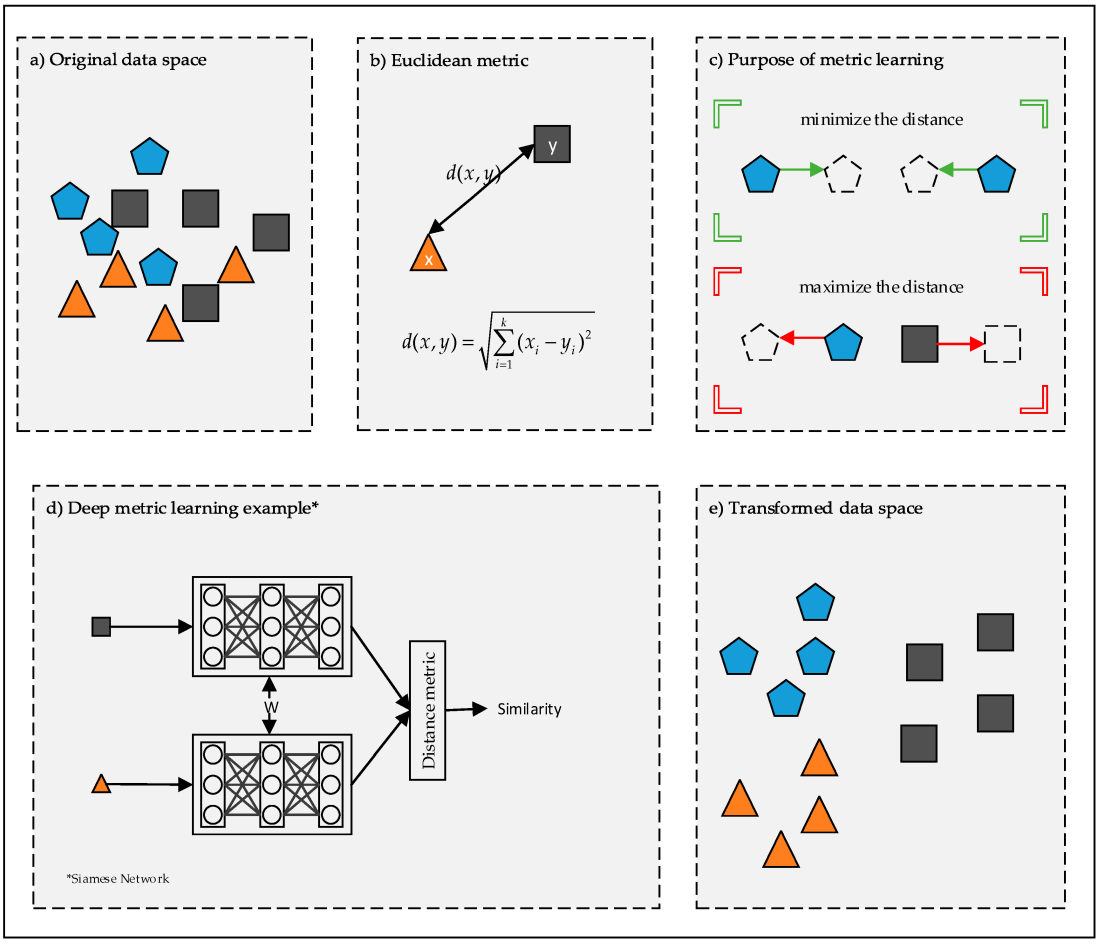

Figure 2.1: Deep Metric Learning

it can be seen that the studies for metric learning are directly related to Mahalanobis distance metric. Let $X = [x_1, x_2, ..., x_N] \in R^{d \times N}$ be the training samples, where $x_i \in R^d$ is i_{th} training example and N is the total number of training samples. The distance between x_i and x_j is calculated as:

$$d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T M (x_i - x_j)} \tag{2.1}$$

$d_M(x_i, x_j)$ is a distance metric, it must have the properties of nonnegativity, the identity of indiscernibles, symmetry, and the triangle inequality. M needs to be symmetric and positive semidefinite. All of the eigenvalues or determinants of M must be positive or zero to be positive semidefinite. When we decompose M, as follows:

$$M = W^T W \tag{2.2}$$

$$d_M(x_i, x_j) = \sqrt{(x_i - x_j)^T M (x_i - x_j)} \tag{2.3}$$

$$= \sqrt{(x_i - x_j)^T W^T W (x_i - x_j)} \tag{2.4}$$

$$= ||Wx_i - Wx_j||_2 \tag{2.5}$$

As can be seen from Equation (2.5), W has a linear transformation property. Thanks to this property, Euclidean distance in the transformed space is equal to Mahalanobis distance in original space for two samples.

This linear transformation shows us the reality in the infrastructure of metric learning. Obtaining a better representation capability for data will certainly enable us to make more accurate predictions possible in classification or clustering problems [27].

Metric learning aims to learn a good distance metric from data. The distance metric provides a new data representation that has more meaningful and powerful discrimination using similarity relationship between samples. When we discuss metric learning, it is useful to mention a linear transformation at first. Linear metric learning approaches provide more flexible constraints in the transformed data space and improves learning performance. These approaches have some advantages, such as convex formulations and robustness to overfitting [2].

Although linear approaches help us to learn a good metric, it is possible to gain better representation capabilities over the data. To better interpret the data, it is necessary to act in accordance with its nature.
The linear transformation has a limited ability to achieve optimum performance over the new representation of data, because they have poor performance to capture nonlinear feature structure. Higher performance is aimed to be achieved by carrying the problem to a non-linear space through kernel methods in metric learning in order to overcome this problem [23].
Although these nonlinear approaches are practical to solve non-linear problems, they may have a negative effect against overfitting. In recent years, with the interest in deep metric learning, it is possible to propose a more compact solution to overcome such problems that exist in both approaches.

2.2 Deep Metric Learning

Traditional machine learning techniques are limited by their ability to process data on raw data.
Therefore, they need feature engineering, such as preprocessing and feature extraction steps before classification or clustering tasks. All of these steps require

expertise and they are not directly within the classification structure.
However, deep learning learns the higher level of data directly in the classification structure. This perspective shows the fundamental difference between traditional machine learning methods and deep learning.
Unlike traditional machine learning methods, deep learning needs a high size of data to achieve successful results, since it is not successful enough in low data size. Besides, deep learning algorithms require a lot of time to train data because of the high data size and a large number of parameters of the algorithm. Therefore, NVIDIA introduced a cuDNN GPU-accelerated library for deep neural networks to perform these high-performance calculations.
Thanks to this library, a lot of deep learning frameworks, such as Caffe, Caffe2, Chainer, Microsoft CNTK, Matlab, Mxnet, PaddlePaddle, PyTorch, TensorFlow, have been developed while using the power of GPU. Basic similarity metrics that are used for data classification are the distances of Euclidean, Mahalanobis, Matusita [16], Bhattacharyya [1], and Kullback-Leibler [7]. However, these pre-defined metrics have limited capabilities in data classification.
Hence, an approach based on the Mahalanobis metric was proposed to classify the data into traditional metric learning to address this problem.
In this approach, the data is transformed into a new feature space with higher discrimination power.

Usually, metric learning approaches are related to the linear transformation of the data without any kernel function. However, these approaches are not successful enough to reveal nonlinear knowledge of the data [29]. For this reason, the expected outcomes could not be obtained while using metric learning.

Although a solution with a kernel-based approach was provided to overcome this problem, there is no obvious success due to some issues such as scaling[12]. Unlike traditional metric learning methods, deep learning solves this problem using activation functions that have nonlinear structure.
Most of the existing deep learning approaches are based on the deep architectural background rather than the distance metric in a new representation space of the data. However, distance-based approaches have recently become one of the most interesting topics in deep learning [6].
While decreasing the distance between dissimilar samples [15], deep metric learning, which aims to increase the distance between similar samples, is directly related to the distance between samples. To execute this process, the metric loss function has been benefited in deep learning. While aiming to bring the samples from the same classes closer to each other, we pushed the samples from different classes apart from each other (Figure 2a).

To illustrate this process with a figure, some experiments on the MNIST image dataset were conducted while using contrastive loss [10].
Distance values represent the mean of distances among similar or dissimilar images

in Figure 2b. As can be seen in the Figure, the distance value for similar images decreased step by step after each epoch. On the other hand, the distance value for dissimilar images also increased at the same. The distance relationship for Siamese network has been successfully applied in each epoch for similar or dissimilar images (Figure 2b).

This experiment proves to us that the purpose of the approach can be successfully implemented. Deep metric learning consists of three main parts, which are informative input samples, the structure of the network model, and a metric loss function. Although deep metric learning especially deals with metric loss function, informative sample selection also plays a very important role in classification or clustering.

Informative samples are one of the most substantial elements that increase the success of deep metric learning. The sampling strategy is capable of increasing both the success of the network and the training speed of the network. The easiest way to determine train samples in contrastive loss is by means of randomly chosen positive or negative pairs of objects. In the beginning, some papers tend to use easy sample pairs for the Siamese network in embedding learning [10].
However, the authors in [21] emphasized that the learning process could be slowed down and negatively affected after the network reached an acceptable performance level. To address this problem, more discriminative models were obtained while using hard negative mining [21]. Triplet network uses an anchor, a positive, and a negative sample to train a network for classification.

In [26], it was observed that some easy triplets had no effect in updating a model due to their poor discriminative power. These triplets cause a waste of time and resources. For this reason, to overcome these problems, it is very convenient to use informative sample triplets, and more feasible train models with a better sample strategy could be provided instead of selecting random samples [26].

Hard negative samples correspond to false-positive samples that are determined by training data. Semi-hard negative mining used for the first time in [20] aims to find negative samples within the margin.
Negative samples are farther from the anchor sample when compared with hard negative mining. There is also a softer transition between positive and negative samples in this approach. The negative mining relationship according to the distance among anchor, positive, and negative samples was illustrated for triplet mining [11], as seen in Figure 4. If the negative samples are too close to the anchor, we can see that the gradient has a high variance and a low signal to the noise ratio, according to [16].
Hence, distance weighted sampling was suggested to avoid noisy samples in [16]. Thanks to this method, a wider range of examples as compared with semi-hard negative mining was also offered. Negative class mining can also be found in the literature instead of negative sample mining [22].

This approach uses one of each class samples for a negative sample of the triple network. To achieve this, the authors chose multiple negative samples with a greedy search strategy To summarize, even if we create good mathematical models and architectures, the learning ability of the network will be limited, depending on the discriminating power of the samples that are presented to the network.
Distinguishing training examples should be presented to the network so that the network can learn better and gain better representation. For this reason, the effect of the relationship between the samples for deep metric learning should be carefully examined.
Thus, sample selection will be very useful as a preprocessing step to increase the success of network model before applying the deep metric learning model. The results in the literature point out that studies for negative mining in deep metric learning have a high impact value.
When considering the benefit of choosing informative samples; on the other hand, the main one could be to avoid overfitting, because similar patterns have similar interaction when the network is being trained, because overfitting may cause slower learning or local optimal learning. Moreover, the number of all possible triplets corresponds O (n3) time complexity when we consider a problem with two classes, which results in a waste of time and unnecessary use of resources. To overcome this problem, it is enough to only deal with valuable triple samples. Thus, significant improvements in performance can be achieved after selecting informative samples.

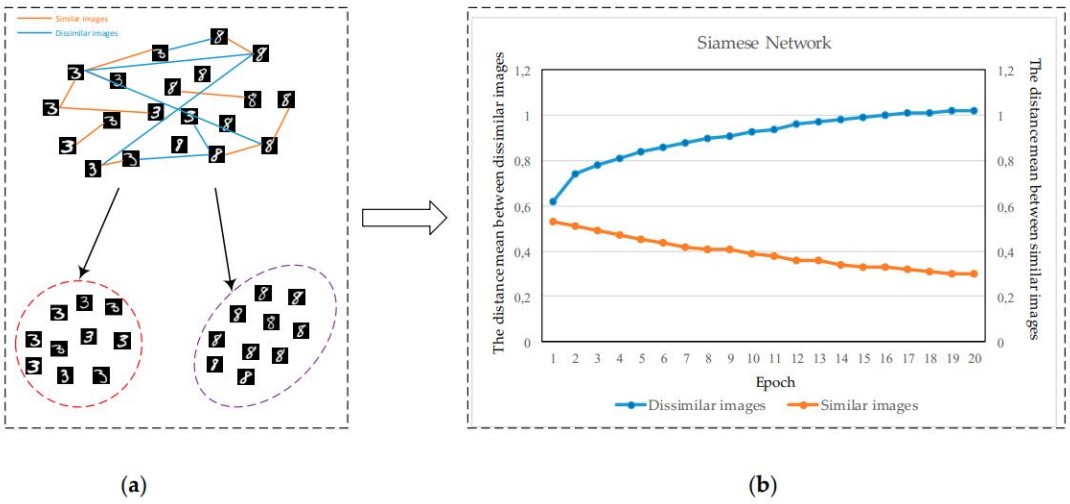

Figure 2.2: Distance Relationship for a Siamese Network (a) Desired handwritten data discrimination for three and eight digits (b) after Siamese network applied to MNIST data for three and eight dig

2.3 Loss function for Deep metric learning

In this section, some loss functions that have been used to apply deep metric learning in the literature will be highlighted. The way that these functions are used will also be introduced and some details about their differences will be given. These functions provide us to increase or decrease the distance between the objects by looking at their similarity. The goal is to achieve the highest feature representation between different objects.

2.3.1 Siamese Network

Initially, the Siamese network was used with neural networks for signature verification [3].
Different from [3], the Siamese network is based on learning from a discriminative learning framework for energy-based models [5].
In this approach, two identical images are taken into the Siamese network and a binary value is obtained as a result of learning from these images. The images are considered as the same class when they are "0"; if they are "1", they are considered as a different class. The Siamese network, as a metric learning approach, receives pair images, including positive and negative samples to train a network model (Figure 5) [5].
The distance between these pair images is calculated via a loss function.
Contrastive Loss has been benefited for the Siamese network in the literature [10]. This approach is a study that provides inspiration for researchers working in the field of deep metric learning. As can be seen in Figure 6a, the siamese network is a very successful model to maximize or minimize the distance between objects to improve classification performance. Shared weights that positively affect the performance of a neural network are used to obtain a meaningful pattern among images in deep metric learning, as shown in Figure 5.
These weights also have important advantages in terms of time and memory. It is also possible to combine the Siamese network and Convolutional neural network, which has important advantages [28], which include similarity learning from direct image pixels, color and texture information at the same time, and its flexible structure. In the deep metric learning model [98], two Siamese Convolutional neural network and Mahalanobis metric were combined for person re-identification, where the Mahalanobis metric was used to classify the data.
The deep metric learning model was created by limiting the weights to a C constant while calculating the L2 norm. The authors in [25] combined softmax loss and center loss for face recognition. While the center loss aims to find a center for deep features of each class to minimize the distances between deep features and their class center, like the contrastive loss, the softmax loss aims to find deep features

that maximize the distances between different classes

Siamese networks [9] are general models for comparing entities. Their applications include signature [9] and face verification, tracking , one-shot learning, and others. In conventional use cases, the inputs to Siamese networks are from different images, and the comparability is determined by supervision

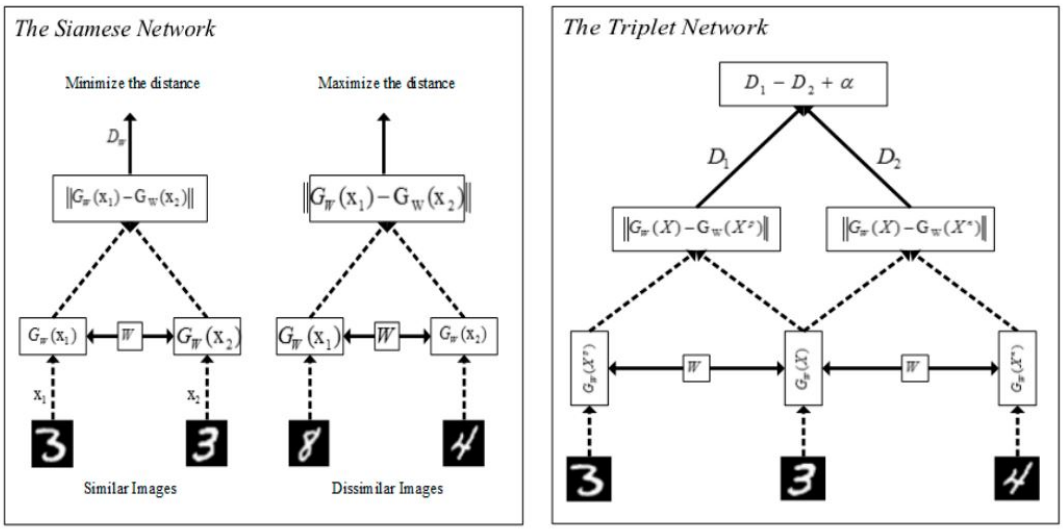

Figure 2.3: The Siamese and Triplet Network

2.3.2 Triplet Network

Triplet networks utilize Euclidean space to compare the objects in the pattern recognition process, and this approach is directly related to metric learning. As can be seen in Equation 6, triplet loss first focuses on the similarity between the pair samples of the same and different classes using shared weights. The classification is carried out comparing the similarity of pair samples (Figure 6b). Triplet networks provide a higher discrimination power while using both in-class and inter-class relations. The authors in [8] proposed a new metric loss that is based on Triplet network inspired by Siamese network contains three objects, which are formed positive, negative, and anchor samples [11]. Triplet networks utilize Euclidean space to compare the objects in the pattern recognition process, and this approach is directly related to metric learning. As can be seen in Equation 6, triplet loss first focuses on the similarity between the pair samples of the same and different classes using shared weights. The classification is carried out comparing the similarity of pair samples . Triplet networks provide a higher discrimination power while using both in-class and inter-class relations. The authors in [8] proposed a new metric

loss that is based on Figure 2.5. Metric loss functions.

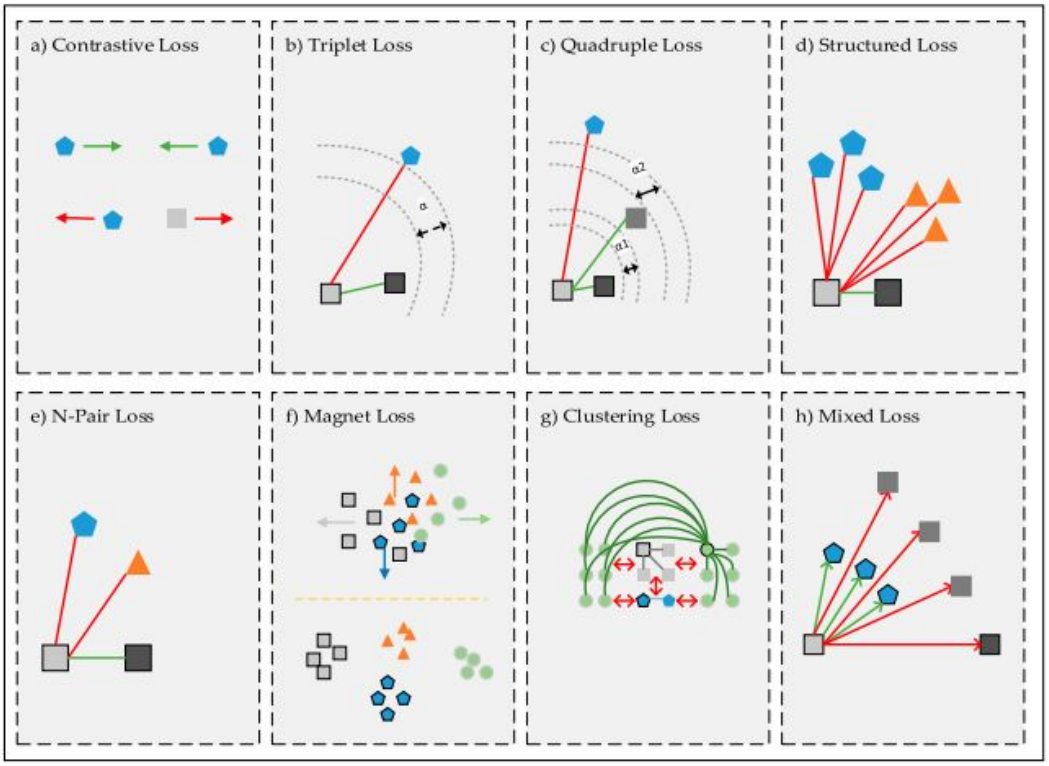

Figure 2.4: Metric Loss functions

2.4 Autoencoders

An autoencoder is a specific type of a neural network, which is mainly designed to encode the input into a compressed and meaningful representation, and then decode it back such that the reconstructed input is similar as possible to the original one. This chapter surveys the different types of autoencoders that are mainly used today. It also describes various applications and use-cases of autoencoders. An autoencoder is a neural network that learns to reconstruct its original input with the aim of learning useful representations. The input vector x is first mapped to a hidden representation $z = f(x)$. The function f is parametrized by the encoder network using one or more layers of non-linearity. The hidden representation z is then mapped to the output $\hat{x} = g(z)$ using the decoder network which parametrizes the decoder function g. Similar to the encoder network, the decoder network can consist of multiple layers of non-linearity. The parameters are optimized to

minimize the following cost function: $L(x, g(f(x))) = ||k\hat{x} - xk||_2^2$ The cost function of mentioned equation is usually optimized with an additional constraint to limit the representational capacity of the autoencoder in order to prevent the autoencoder from learning the useless identity function. The constraint could be imposed on the architecture of the autoencoder (e.g., limiting the latent code dimensionality) or could be in the form of a regularization term in the final objective (e.g., sparsity constraint). [14]

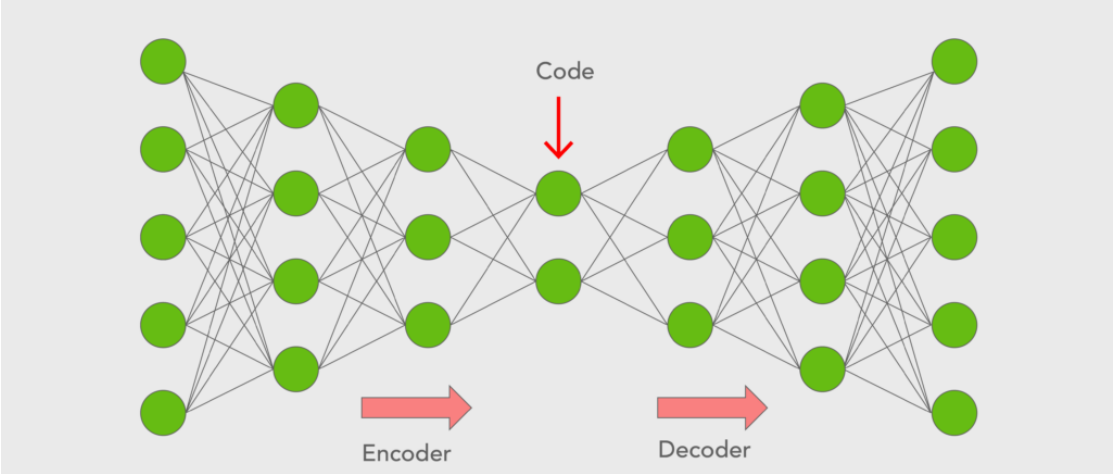

Figure 2.5: Auto encoder

2.5 Variational Autoencoders

The VAE is inspired by the Helmholtz Machine (Dayan et al., 1995) which was perhaps the first model that employed a recognition model. However, its wake-sleep algorithm was inefficient and didn't optimize a single objective. The VAE learning rules instead follow from a single approximation to the maximum likelihood objective. VAEs marry graphical models and deep learning.
The generative model is a Bayesian network of the form $p(x|z)p(z)$, or, if there are multiple stochastic latent layers, a hierarchy such as $p(x|zL)p(zL|zL1)...p(z1|z0)$. Similarly, the recognition model is also a conditional Bayesian network of the form $q(z|x)$ or as a hierarchy, such as $q(z0|z1)...q(zL|X)$.
But inside each conditional may hide a complex (deep) neural network, e.g.$z|xf(x, \epsilon)$, with f a neural network mapping and ϵ a noise random variable. Its learning algorithm is a mix of classical (amortized, variational) expectation maximization but through the reparameterization trick ends up backpropagating through the many layers of the deep neural networks embedded inside of it[14].

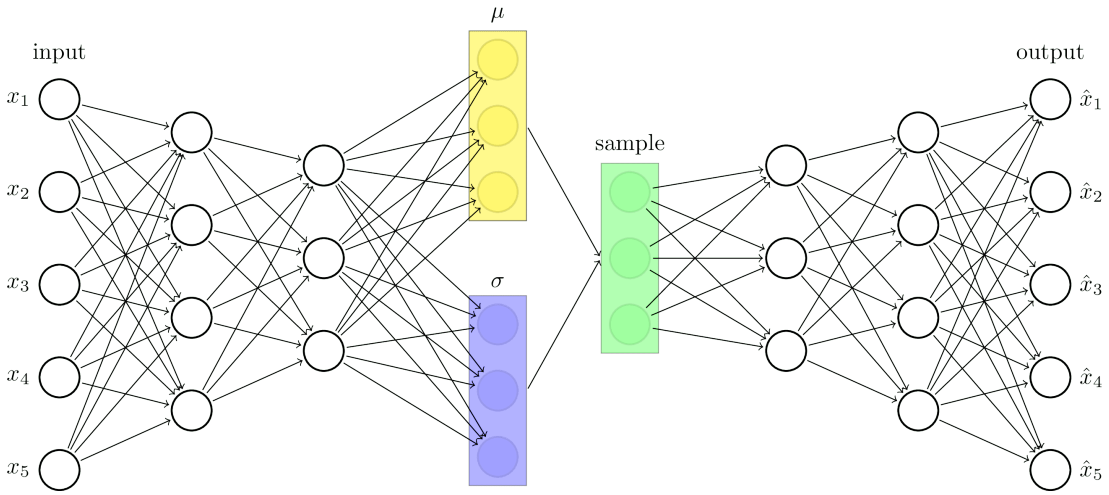

Figure 2.6: Variational Auto encoder

2.6 Condtional Variational Autoencoders

Conditional variational autoencoders (CVAEs) are versatile deep generative models that extend the standard VAE framework by conditioning the generative model with auxiliary covariates.
The original CVAE model assumes that the data samples are independent, whereas more recent conditional VAE models, such as the Gaussian process (GP) prior VAEs, can account for complex correlation structures across all data samples. While several methods have been proposed to learn standard VAEs from partially observed datasets, these methods fall short for conditional VAEs.
Conditional VAEs (CVAEs) [Sohn et al., 2015] were proposed as an extension to the VAE that models the distribution of the high-dimensional output as a generative model conditioned on auxiliary covariates (control variables). The CVAE model offers an approach to control the data generating process of a VAE and thereby perform structured output predictions.
However, though the standard VAEs and CVAEs are powerful generative models for complex datasets, they ignore the possible correlations between the data samples. Recent work has extended the VAE modelling framework to incorporate arbitrary correlations across all data samples by replacing the i.i.d. standard Gaussian prior with a Gaussian process (GP) prior [Casale et al., 2018, Fortuin et al., 2020, Ramchandran et al., 2021]. These GP prior VAEs are also conditional generative models as the generative process depends on auxiliary covariates (such as time, image rotation, etc.). GP prior VAEs have been demonstrated to be especially well suited for temporal and longitudinal datasets.
Many real-world datasets, such as clinical studies, usually contain a significant

number of missing values. Moreover, real-world datasets contain variables that can be regarded as either covariates or dependent variables, both with varying amounts of missing information. For example, in a clinical study, dependent variables can comprise of lab measurements of a participant's blood sample, whereas covariates may contain information about the participant, such as age, sex, height, etc. Recent papers have demonstrated the ability of VAEs to handle missing values and have shown good imputation performance for the dependent variables.
However, these models either do not incorporate the auxiliary variables or do not account for the missing values in the auxiliary variables. In other words, no methods have been developed to learn CVAEs with missing auxiliary variables[14].

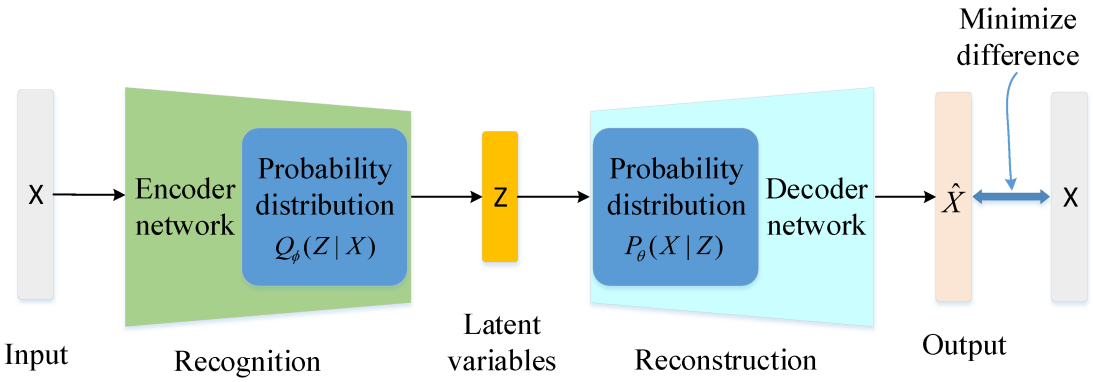

Figure 2.7: Conditional Variational Auto encoder

2.7 Long Short-term memory Architecture

The LSTM contains special units called memory blocks in the recurrent hidden layer. The memory blocks contain memory cells with self-connections storing the temporal state of the network in addition to special multiplicative units called gates to control the flow of information. Each memory block in the original architecture contained an input gate and an output gate.

The input gate controls the flow of input activations into the memory cell. The output gate controls the output flow of cell activations into the rest of the network. Later, the forget gate was added to the memory block [4]. This addressed a weakness of LSTM models preventing them from processing continuous input streams that are not segmented into subsequences.

The forget gate scales the internal state of the cell before adding it as input to the cell through the self-recurrent connection of the cell, therefore adaptively forgetting or resetting the cell's memory. In addition, the modern LSTM architecture contains

peephole connections from its internal cells to the gates in the same cell to learn precise timing of the outputs [19]. An LSTM network computes a mapping from an input sequence x = (x1, ..., xT) to an output sequence y = (y1, ..., yT) by calculating the network unit activations using the following equations iteratively from $t = 1$ to T:

$$\begin{aligned}
&\text{LSTM} : h_t^{l-1}, h_{t-1}^l, c_{t-1}^l \rightarrow h_t^l, c_t^l \\
&\begin{pmatrix} i \\ f \\ o \\ g \end{pmatrix} = \begin{pmatrix} \text{sigm} \\ \text{sigm} \\ \text{sigm} \\ \tanh \end{pmatrix} T_{2n,4n} \begin{pmatrix} h_t^{l-1} \\ h_{t-1}^l \end{pmatrix} \\
&c_t^l = f \odot c_{t-1}^l + i \odot g \\
&h_t^l = o \odot \tanh(c_t^l)
\end{aligned}$$

In these equations, sigm and tanh are applied element-wise. the logistic sigmoid function, and i, f, o and c are respectively the input gate, forget gate, output gate and cell activation vectors, all of which are the same size as the cell output activation vector m,

is the element-wise product of the vectors, g and h are the cell input and cell output activation functions, generally and in this paper tanh, and ϕ is the network output activation function, softmax in this paper [17].

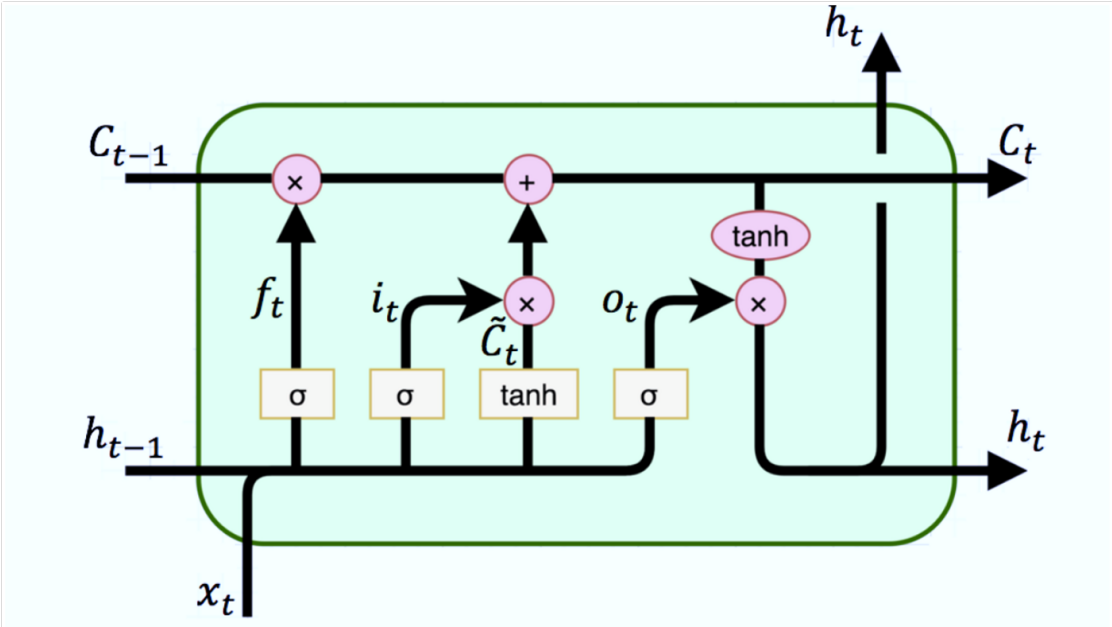

Figure 2.8: LSTM architecture

Chapter 3

Data and Model

In this chapter, the Train, validation and Test datasets plus the pre-processing steps will be explained. Moreover, we will describe the proposed model.

3.1 What is Solar Energy

Solar energy is sunlight that has been transformed into thermal or electrical energy. The United States contains among of the world's most prolific and cleanest solar resource bases. Solar energy is the most abundant and clean renewable energy source currently available. Solar technology can capture this energy for a range of purposes, such as electricity generation, interior lighting, and water heating for household, commercial, and industrial use.
Photovoltaics, solar heating and cooling, and concentrating solar power are the three basic methods for utilizing solar energy. To power anything from small gadgets like calculators and traffic signs up to homes and huge commercial enterprises, photovoltaics directly convert sunshine into electricity via an electrical process. Both solar heating and cooling (SHC) and concentrating solar power (CSP) applications employ the heat produced by the sun to run conventional electricity-generating turbines in the case of CSP power plants or to offer space or water heating in the case of SHC systems.
Solar energy is a very adaptable energy source that may be used to create utility-scale solar power plants or distributed production systems that are placed at or close to the point of use (similar to traditional power plants). Using cutting-edge solar + storage technologies, both of these approaches can also store the energy they generate for distribution when the sun goes down. In order to move the U.S. toward a clean energy economy, solar power coexists with other technologies like wind power in a complicated and interconnected electricity infrastructure in the country.

For consumers and companies to have equal access to sustainable energy technologies like solar, all of these applications rely on supportive regulatory frameworks at the local, state, and federal levels.

Figure 3.1: Photo of sample Solar site

3.2 Data

NREL's Solar Power Data for Integration Studies are synthetic solar photovoltaic (PV) power plant data points for the United States representing the year 2006. This dataset consists of 34 sites of solar photovoltaic (PV) power plants located in the Oklahoma State. The data has been recorded for every 5 minutes. We picked 1/10 portion of the data as the test data samples, 1/10 as the validation data points and the rest is dedicated to the training data samples.

3.3 Data Pre-processing

Based on the extracted data from the solar photovoltaic (PV) power plants located in the Oklahoma State, we decided to omit zero powers and the related date from

Local Time	Power
1/1/2006 0:00	0
1/1/2006 0:05	0
...	...
12/31/2006 8:45	11.5

Table 3.1: Input data from one site in Oklahoma State

the train, validation and test datasets. this operation leads to a better forecasting result in the final model.

Listing 3.1: Python code for pre-processing

```
import pandas as pd
import numpy as np
import math
df = pd.read_csv('/content/drive/MyDrive/Dataset/Actual_34.05_-97.15
    _2006_UPV_50MW_5_Min.csv')
print(df.head())
df= df[df['Power(MW)'] != 0.0]
time = pd.to_datetime(df.pop('LocalTime'))
series = df['Power(MW)']
series.index = time
```

3.4 Proposed Model

In this study, we aim to learn the probability distribution function (PDF) of the PV in Solar Power time series to predict the future values of the time series. The proposed methodology will be explained in the following sections.

3.5 Kernel Regression Problem

The standard regression task is to estimate an unknown function $f : R^D \longrightarrow R$ based solely on a training set of (possibly noisy) evaluations $(x_1, y_1), ..., (x_n, y_n)$. Here $y_i = f(x_i) + \epsilon$, where ϵ represents some small noise.

Specifically, for some distribution of test points x_t, our task is to find an estimator $\hat{f} : R^D \longrightarrow R$ of f that minimizes some loss function (e.g., squared error) over the distribution of test points.

In kernel regression, the value of $y_t = f(x_t)$ is approximated by a weighted average of the training where $k_{ij} = k(x_i, x_t) >= 0$ is referred to as the kernel

function.
A wide variety of kernel functions have been studied in prior literature, e.g., Gaussian, triangular, spherical, wave, etc.. To make the estimate $\hat{y}_t$ truly local, the kernel function $k(xi, xt)$ should decay rapidly with increasing (squared) distance $d(x_i,\ x_t)$. The optimal rate of decay depends on the noisiness and smoothness of the function f, the density of the training samples, and the scale of the input features.
Hence the choice of kernel function may require great care, especially in high dimensional spaces. Several publications elaborate on how to chose the right kernel function for a given data set [24]. Additionally, kernel functions may have several tunable parameters pertaining to the rate of decay; these also must be set properly to obtain good performance on the regression task.

In [24] the author presents the algorithm Metric Learning for Kernel Regression (MLKR) for learning an appropriate distance function for kernel regression.
The approach applies to any distanced-based kernel function

$$k(x_i, x_j) = k_D(D_\theta(x_i, x_j)) \tag{3.1}$$

with differentiable dependence on parameters θ specifying the distance function D_θ. Specifically, MLKR consists of setting initial values of θ, and then adjusting the values using a gradient descent procedure:

$$\Delta\theta = -\epsilon\frac{\partial L}{\partial\theta} \tag{3.2}$$

where ϵ is an adaptive step-size, and the loss function L is the cumulative leave-one-out quadratic regression error of the training samples:

$$L = \Sigma(y_i - \hat{y}_i)^2 \tag{3.3}$$

with $\hat{y}_i$ defined as in 3.1. Of course, other methods for minimizing 3.3 such as conjugate gradient, stochastic gradient or BFGS may also be used and might lead to faster convergence results. While MLKR may apply to many types of kernel functions and distance metrics, we hereafter focus the exposition on a particular instance of the Gaussian kernel and Mahalanobis metric, as these are used in our empirical work. The Gaussian kernel is generally defined as follows:

$$k_{ij} = \frac{1}{\sigma\sqrt{2\pi}}e\frac{-d(x_i, x_j)}{\sigma^2} \tag{3.4}$$

where $d(\ x_i,\ x_j)$ is the squared distance between the vectors $\ x_i$ and $\ x_j$. As the constant factor before the exponent in 3.4 cancels out in 3.1, we will drop it for simplification. Also, we will absorb σ^2 in $d()$ and can fix $\sigma = 1$.

A Mahalanobis metric is a generalization of the Euclidean metric, in which the squared distance between two vectors x_i and x_j is defined as

$$d(\ x_i,\ x_j) = (\ x_i\ x_j)^T M(\ x_i\ x_j) \tag{3.5}$$

where M can be any symmetric positive semidefinite1 real matrix. (Setting M to the identity matrix recovers the standard Euclidean metric.) Figure 1 illustrates the difference between a Mahalanobis metric and the Euclidean metric on a synthetic regression example.
Unfortunately, learning the matrix M directly requires enforcing a positive semi-definite constraint during optimization. This is non-linear and expensive to satisfy. One way to eliminate this expensive constraint is to decompose M as

$$M = A^T A \tag{3.6}$$

where the $i_t h$ row of A is the corresponding eigenvalue of M. Substituting 3.6 into 3.5 enables us to express the Mahalanobis distance as the Euclidean distance after the mapping x → A x: d(xi , xj) = kA(xi xj)k 2 . (7) Equation (6) allows us to rewrite (3) in terms of the unconstrained matrix A, instead of M. Let y^i be the target estimate of vector f(xi). The gradient of (3) with respect to A can be stated as

$$\frac{\partial L}{\partial A} = 4A\Sigma(y_i - \hat{y}_i)^2\Sigma(\hat{y}_j - y_j)^2 k_{ij} x_i x_j^T \tag{3.7}$$

where xij = (xi xj). We note that minimizing 3.3 gives us a kernel function in a entirely data-driven, non-parametric way. There are no parameters to tune, apart from the initialization of A and the gradient stepsize. Also, the algorithm learns its own scaling (in the form of A) and is therefore invariant to the scaling of the input vectors. In our experiments, we usually initialized with the

3.6 Time Series Approximation by PDF learning

Here, our problem is to capture a probability distribution $P(x)$ over n-dimensional data points X in a potentially high dimensional vector space.

In fact, we want to be able to generate many samples X^* as close as possible to X . As the complexity of the dependencies between variables of grows, the difficulty of learning the true $P(X)$ increases. Hence, we define a "latent variable"-based model in which the hidden random vector Z embody the major characteristics of $P(X)$ (e.g. the PDF of the future PV, or any desired nodal PDF in the whole dataset).

More specifically, is sampled following some unknown distribution $P(Z)$ over the high dimensional space Z . To justify that our approach is generative (i.e.

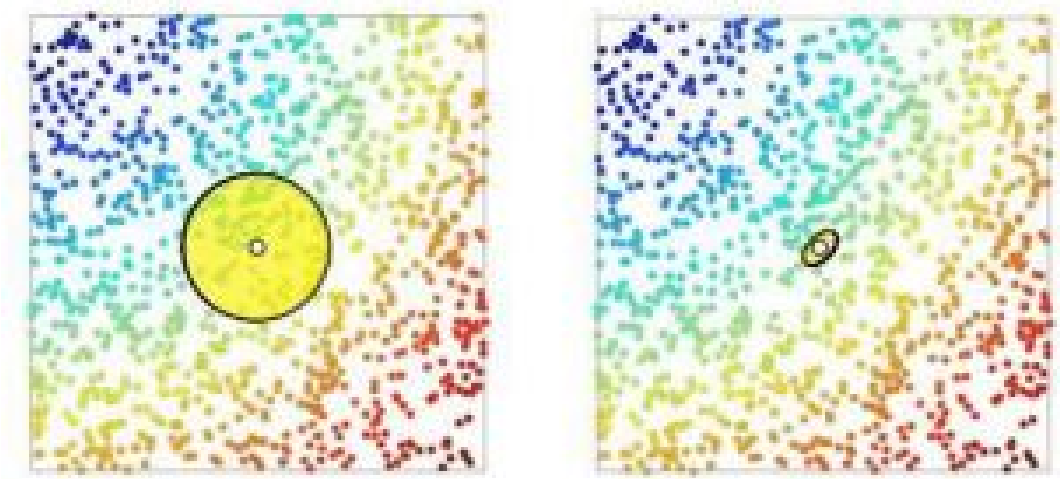

Figure 3.2: An illustration of kernel regression under varying distance metrics. The color of the points represents the function value. The circle shows the radius that encapsulates 95the weights. The function value is estimated at a test point at the center. Left: With the use of the Euclidean Distance metric, the kernel is spherical and ignores the present structure that points along the diagonal share similar function values. Right: After training, under the Mahalanobis metric, the kernel function follows the direction of the target function. The radius has also shrunk and has therefore adapted to the relatively densely sampled and noiseless training samples. identity matrix. If local minima are a concern, one can use several runs with different random initializations and choose the outcome with minimum training error L.

the model can generate samples X^*), we ensure that there exists at least one configuration $\hat{z}$ in Z that causes the model to generate some sample $\hat{X}$ in X . Assuming a family of deterministic functions $f(z, \theta)$ with parameters θ , each "latent variable-parameter" pair is mapped to a sample in using $f : Z * \theta \longrightarrow X$. We find an optimal θ , In this case, the probability of f creating an output X^* similar to the observed data X is maximized; hence, our optimization is written as:

$$\hat{\theta} = argmax_{\theta}[P(X) = \int f(z, \theta)P(z)dz] \tag{3.8}$$

Theorem (1): In any space A, any complicated probability density function over samples can be modeled using a set of $dimA$ random variables with normal distribution, mapped through a high capacity function[13]. We consider the z belongs to the distribution $N(0, I)$

$$\hat{\theta} = argmax_{\theta} \int N(X|f(z,\theta), \sigma^2 * I) N(z|0, I) dz \tag{3.9}$$

To solve 3.9 we have $Q(Z|X)$ and we compute it using the Kullback–Leibler (KL) divergence:

$$KL[Q(z)||p(z|X)] = E_{z\ Q}[logQ(z) - log(p|X)] \tag{3.10}$$

In order to generate X (that is, create samples X^*), our objective is to maximize $logP(x)$ while minimizing the KL divergence; hence, we minimize

$$E_{z\ Q}[P(X|z) - KL[Q(z|X)||p(z)]] \tag{3.11}$$

using SGD.

3.7 Error Function

Our aim is to integrate the Mlkr approach to the generative framework which consists of the encoder and decoder architectures. The error function for the new network can be written as:

$$L = ||y - y_t||^2 + KL[Q(z|X,y)||p(z)] + ||y - y_{t^*}||^2 \tag{3.12}$$

3.8 Learning Algorithm

This model is very effective to learn better features and distance measurement at the same time, since the metric learning can guide the network to learn better parameters.

Algorithm 1 Learning Algorithm for Deep metric learning for pdf learning. Good values are $\alpha = 0.001$ or 0.0001

procedure ALGORITHM($\alpha, X_{train}, Y_{train}$)
    ▷ α is the learning rate
    ▷ for each training epoch
    for $i = 1,2,3,..,n$ **do**
        for $[X_{batch}, Y_{batch}]$ in $[X_{train}, Y_{train}]$ **do**
            ▷ Calculate forward propagation
            Calculate $f(X_{batch})$
            ▷ Calculate Error function based on 3.12
            Calculate L according to 3.6
            Calculate Pdf according to 3.10
            Calculate gradients w.r.t the L , A , W

The output is the deep model parameterized by $W^{(i)}$, $b^{(i)}$ and the learned linear transformation A.

Chapter 4

Implementation and Results

4.1 Pytorch Framework

The open-source deep learning framework PyTorch is renowned for its adaptability and simplicity. This is made possible, in part, by its interoperability with Python, a well-liked high-level programming language used by data scientists and machine learning engineers.
Deep learning models, a type of machine learning frequently used in applications like image recognition and language processing, may be built using PyTorch, a fully featured framework. Most machine learning developers find it reasonably simple to understand and use because it is written in Python. PyTorch stands out for its superior GPU support and usage of reverse-mode auto-differentiation, which permits dynamic modification of computation graphs. It is therefore a well-liked option for quick testing and prototyping.
Because it is built in Python and leverages that language's imperative, define-by-run eager execution mode, which executes actions as they are called from Python, PyTorch is particularly well-liked among Python developers. According to a poll, AI and machine learning activities are becoming increasingly important as the use of the Python programming language grows, and PyTorch is being more often used to support these tasks. PyTorch is a solid option for Python developers who are new to deep learning because of this, and an expanding collection of deep learning courses are built using PyTorch. Since the API hasn't changed much over the years, seasoned Python developers should have little trouble understanding the code.
Smaller projects and rapid prototyping are two areas where PyTorch excels. It is favored by the academic and scientific communities due to its versatility and ease of usage.
A group of nested functions are applied to input parameters by neural networks to change the input data. By computing their partial derivatives (gradients) with

respect to a loss metric, deep learning aims to optimize these parameters, which consist of weights and biases and are stored as tensors in PyTorch. Forward propagation involves feeding input parameters into the neural network, which then outputs a confidence score to nodes in the following layer up until the output layer, which is where the error of the score is determined. The errors are transmitted back through the network again through backpropagation as part of a technique called gradient descent, and the weights are changed to improve the model. Facebook

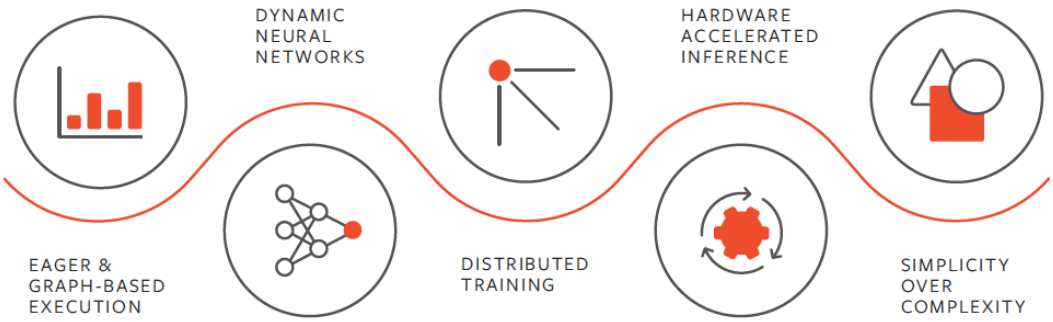

Figure 4.1: Pytorch cycle

programmers have been putting a lot of effort into enhancing PyTorch's useful applications. Just-in-time compilation and support for Google's TensorBoard visualization tool are two recent improvements. Additionally, ONNX (Open Neural Network Exchange) support has been increased, allowing developers to choose the deep learning frameworks or runtimes that are most effective for their applications.

4.2 Implementation

Here, we describe that the proposed model consists of three main parts, the first part which is called the feature extraction function will use the LSTM architecture plus the Kernel Metric Learning approach.
the Kernel metric approach is used to improve the result of the function with minimizing the Mean Squared Loss.
The extracted features of the input X and the actual label will be forwarded to the encoder architecture. The encoder architecture consists of two to four Linear Layers and in the final layers the parameters for mapping the input to a new space will be calculated.
Finally, the Z parameter and the extacted features of the input X will be forwarded to the decoder architecture in the Conditional Variational Auto-encoder, the decoder

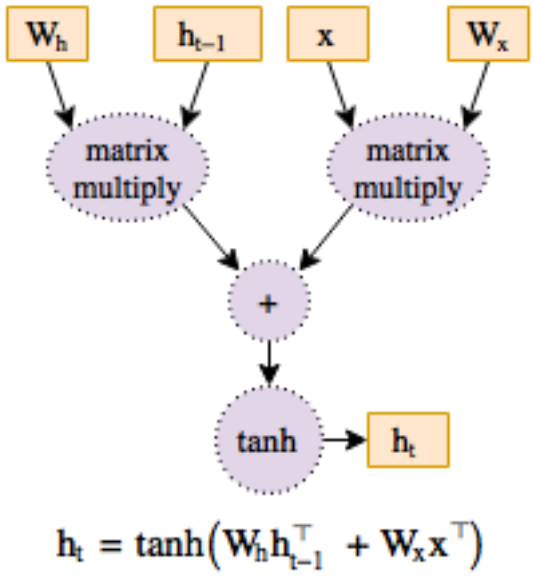

Figure 4.2: Network graph in Pytorch

contains two to four Linear layer and in the final layer the generated label and its probability can be calculated. in this version the decoder's output dimension is equal to 1 and the result of training and evaluation of different metrics are shown in the next section. For the Testing Architecture, the Encoder architecture in the Conditional Variational Autoencoder will be removed and the generated results will be obtained by passing a random number from the $N(0,1)$.

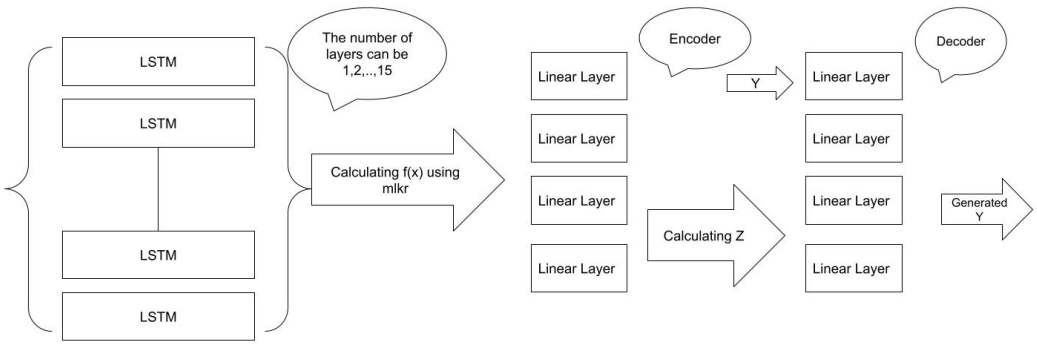

Figure 4.3: Implemented Architecture

4.3 Hyperparameters

4.3.1 Learning Rate

The learning rate is a hyperparameter that determines how much to alter the model each time the model weights are updated in response to the predicted error. It can be difficult to choose the learning rate since a number that is too little could lead to a lengthy training process that could become stuck, but a value that is too large could lead to learning a suboptimal set of weights too quickly or to an unstable training process.

When constructing your neural network, the learning rate can be the most crucial hyperparameter. Therefore, understanding how to conduct research into the impact of learning rate on model performance and to develop an intuitive understanding of the dynamics of learning rate on model behavior is crucial.

4.3.2 Window Length

The number of samples in a time series that are used as input for prediction is represented by window length. In this study, sliding windows are utilized. The time series window moved over the time window. The model receives input from window elements at each step to create predictions, after which the window is slid forward and the process is repeated. More information is taken into account when making predictions with larger windows, however very big windows might reduce the sensitivity of the forecasts and produce highly smooth predictions. Here we considered 288 which is the data in 48 hours as the sliding time window.

4.4 Evaluation Metric

Since we have 34 sites in our datasets, first, each metric is applied to a specific dataset and the final result will be the average of all the applied metrics.

4.4.1 Root Mean Square Error

The reason of choosing this metric is having more readable evaluation, regardless of data scale [13]. The lower the value of RSME the better is the result.

$$RMSE = \sqrt{\frac{1}{n}\Sigma_{i=1}^{n}(y_i - f_i)} \tag{4.1}$$

4.4.2 Mean Absolute Error

A metric that tells us the mean absolute difference between the predicted values and the actual values in a dataset. The lower the MAE, the better a model fits a dataset.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|f_i - y_i| \tag{4.2}$$

4.5 Training the model

We trained the mentioned model for 50 and 100 epochs. our hyper-parameters are learning rate, number of neurons and layers in encoder/decoder and the number of LSTMs.

4.5.1 Model 1

In this configuration, we chose the learning rate equal to 0.001 and we trained the model for 50 epochs with 32 neurons as input for the 4 blocks of the encoder and decoder. the number of hidden layers in LSTM is equal to 1.As we can see in the shape, the learning curve is not enough smooth , so we need to normalize the labels and then change the hyper-parameters.

4.5.2 Model 2

In this configuration, we chose the learning rate equal to 0.0001 and we trained the model for 100 epochs with 32 neurons as input for the 4 blocks of the encoder and decoder.the number of hidden layers in LSTM is equal to 4. As we can see in the shape, decreases rapidly and it shows that the learning rate is low for this dataset.

4.5.3 Model 3

In this configuration, we chose the learning rate equal to 0.001 and we trained the model for 100 epochs with 32 neurons as input for the 4 blocks of the encoder and decoder.the number of hidden layers in LSTM is equal to 2.

4.6 Comparison with other methods

4.6.1 Convolutional Graph Autoencoder

In [19] A novel deep generative model, Convolutional Graph Autoencoder, is presented for a new problem, nodal distribution learning in graphs. The model

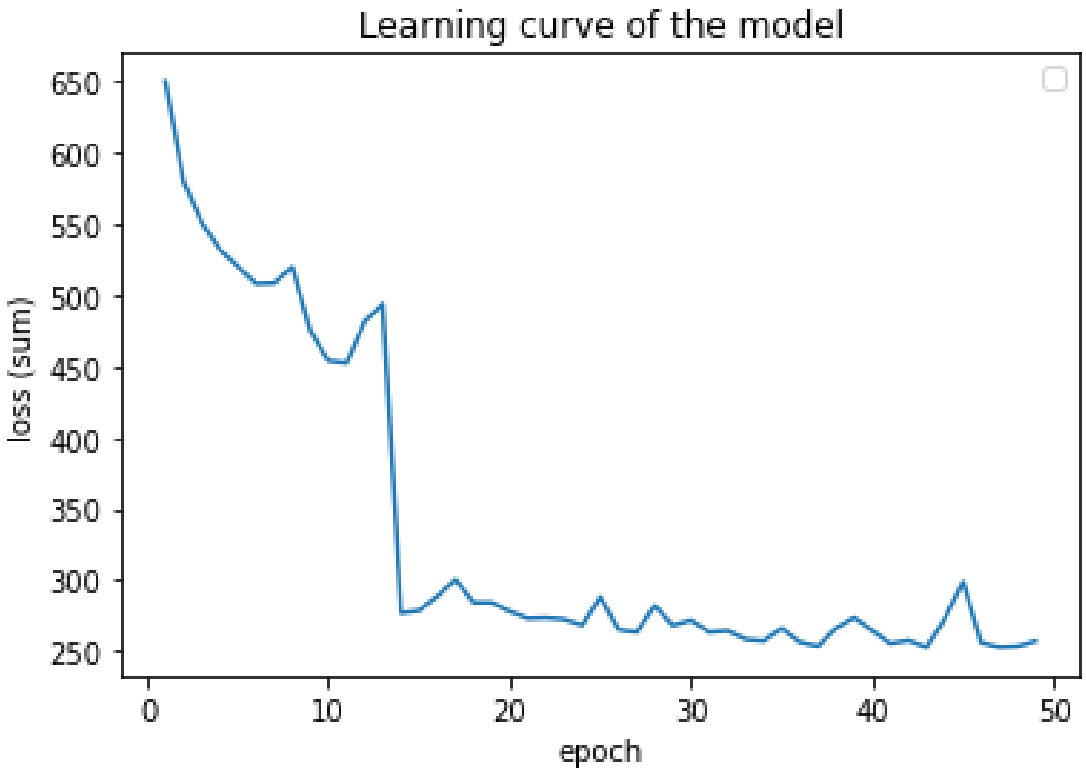

Figure 4.4: Model 1 learning curve

model	MAE	RSME	time
Model 3	0.108	0.358	10-min
Model 3	0.578	0.450	30-min
Model 3	0.690	0.712	1-hour
Model 3	0.866	0.957	3-hour
Model 3	0.963	1.098	6-hour

Table 4.1: Evaluation of the best trained model

captures deep convolutional features from an arbitrary graph-structured data, to learn the corresponding probability densities of nodes. Here, the problem of spatio-temporal solar irradiance forecasting is presented as a graph distribution learning problem where each node of the graph represents a solar irradiance measurement site, while each edge represents the distance between the sites. Using graph spectral convolutions, the spatial features of the solar data are extracted, that are further

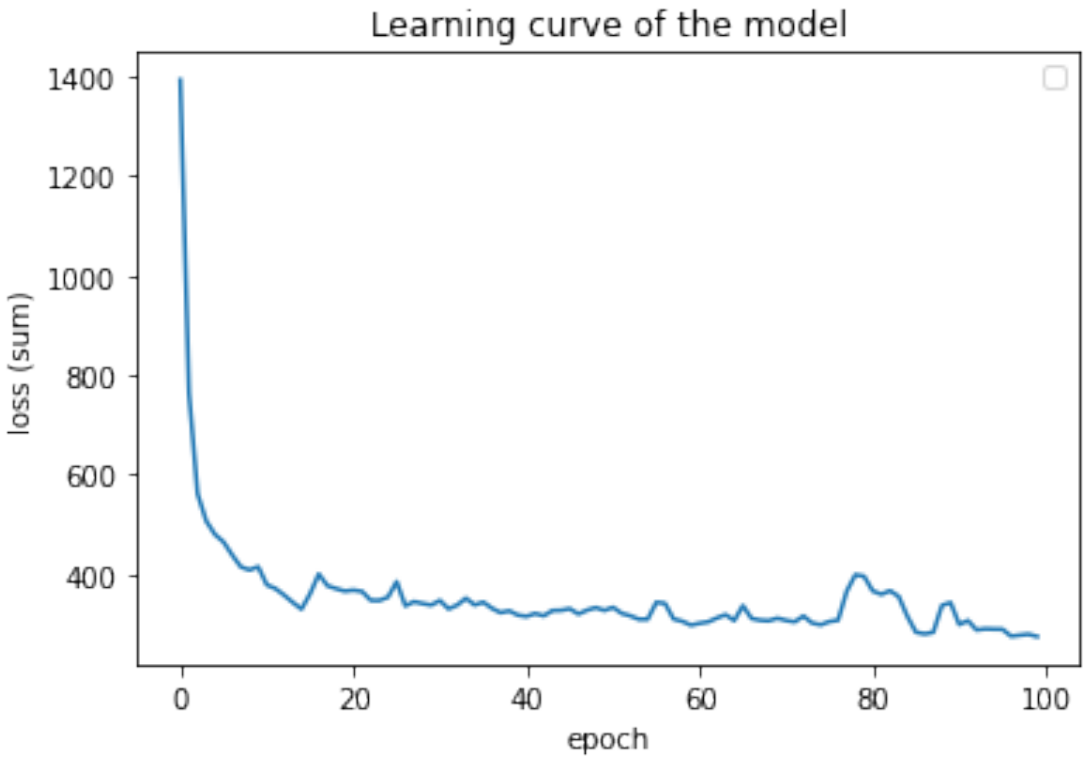

Figure 4.5: Model 2 learning curve

used by an encoding and decoding ANN to capture the distribution of future solar irradiance. Our deep learning model is used to provide probabilistic forecasts for the National Solar Radiation Database. Simulation results show better reliability, sharpness and Continuous Ranked Probability Score compared to recent baselines in the literature.

4.6.2 Deep Convolutional Graph Rough Auto-encoder

In [13] presents a novel deep generative model, convolutional graph rough variational autoencoder, to learn the probability densities of a graph of photovoltaic sites. Here, we formulate the PV power forecasting problem as a graph distribution learning where each PV site is modeled as a node in the graph, and the distance between sites is modeled by edges in the graph. The model applies a deep convolutional graph to extract spatial features between the nodes in the graph. Further, the extracted features are fed to a rough variational auto-encoder to estimate the probability density of future PV power outputs.

Based on the comparison result, the proposed model in this study achieved 0.2

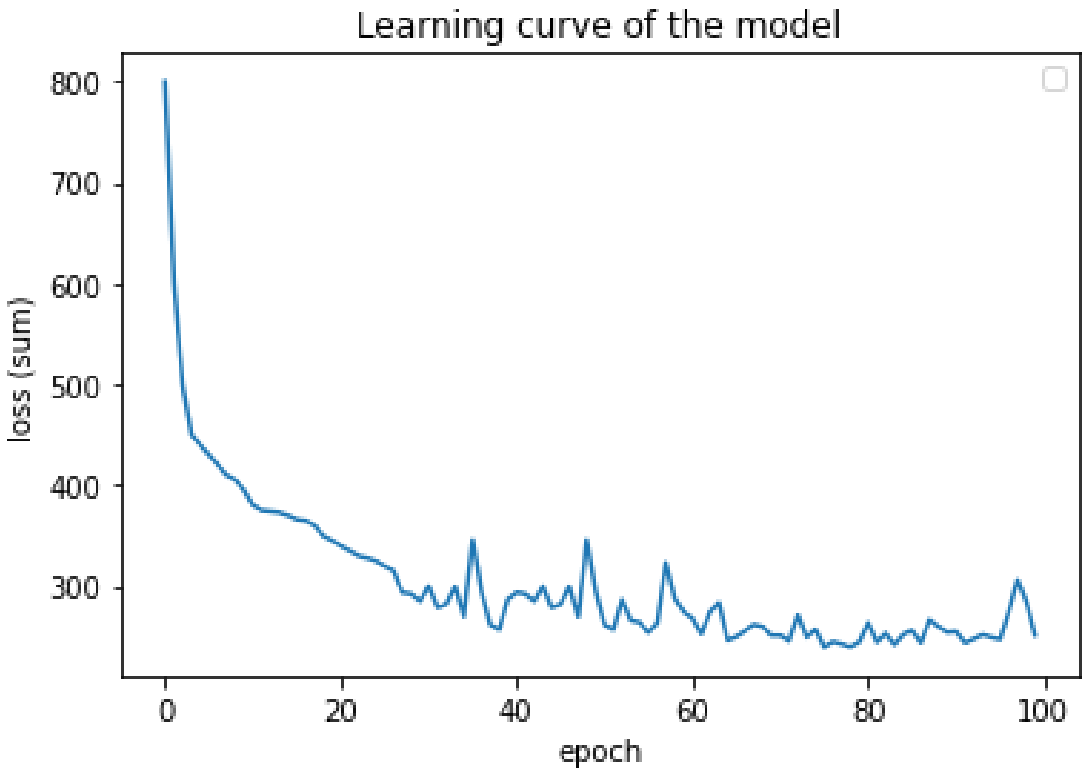

Figure 4.6: Model 3 learning curve

more performance in case of MAE and RSME.

Chapter 5

Future Work

The proposed solution can be modified in different sections, such as changing the generative back-bone architecture and applying the final models on different datasets. The use of deep metric learning in novel algorithms in Image processing is increasing and one of the algorithms that we want to propose will be the use of Convolutional LSTM on the Image datasets and analyzing the final results in the Domain Adaptation field. Deep metric learning is an efficient approach to extract useful features from different types on Input datasets.

Chapter 6

Conclusion

A novel deep generative model, Metric LSTM Conditional Variational Autoencoder, is presented for power probability distribution in Solar Power Dataset. The model captures features using LSTM and optimizes them with using metric learning extracting the predicted label from each node's neighbors.
to learn the corresponding probability densities of the output power, the problem of Solar Power forecasting is presented as a feature extraction function problem where the output of the function for each data will be calculated by minimizing it with respect to the the output of Kernel metric learning for each data point's neighbors.
Applying the Kernel Metric method on whole data points was computationally expensive so this method was applied only on the nearest neighbors of the data point, with selecting the proper number of neighbors which is equal to the square root of data samples the performance of the model was close to the ideal case.
the outputs of the feature extraction function are further used by an encoding and decoding ANN to capture the distribution of future solar power. This deep learning model is used to provide probabilistic forecasts for the Solar Power Data which are synthetic solar photovoltaic (PV) power plant data points for the United States representing the year 2006. Simulation results shows the great performance in terms of Mean Absolute Error and RMSE.

Appendix A

Appendix A

A.1 Implemented Code

```
    class DenseLSTM(nn.Module):
    def __init__(self, input_dim, hidden_dim, lstm_layers,encoder_dim
    ,max_number, bidirectional=False):
        super(DenseLSTM, self).__init__()
        self.input_dim = input_dim
        self.hidden_dim = hidden_dim
        self.layers = lstm_layers
        self.bidirectional = bidirectional
        self.encoder_dim=encoder_dim
        self.max_number=max_number
        self.lstm = nn.LSTM(input_size=self.input_dim,
                            hidden_size=self.hidden_dim,
                            num_layers=self.layers,
                            bidirectional=self.bidirectional)
        self.act1 = nn.ReLU()

        self.encoder1= nn.Linear(self.hidden_dim+1, self.encoder_dim)
        self.relu1=nn.ReLU()
        self.encoder2= nn.Linear(self.encoder_dim , self.encoder_dim)
        self.relu2=nn.ReLU()
        self.encoder3= nn.Linear( self.encoder_dim , self.encoder_dim
    )
        self.relu3=nn.ReLU()
        self.encoder4= nn.Linear(self.encoder_dim , 2)
        self.relu4=nn.ReLU()
        self.encoder5= nn.Linear(self.encoder_dim , 2)
        self.relu5=nn.ReLU()

        # decoder structure
```

```
28          self.decoder1= nn.Linear(2+self.hidden_dim , self.encoder_dim
    )
29          self.relud1=nn.ReLU()
30          self.decoder2= nn.Linear(self.encoder_dim , 2*self.
    encoder_dim)
31          self.relud2=nn.ReLU()
32          self.decoder3= nn.Linear(2*self.encoder_dim , 4*self.
    encoder_dim)
33          self.relud3=nn.ReLU()
34          self.decoder4= nn.Linear(4*self.encoder_dim , 1)
35          self.relud4=nn.ReLU()
36
37
38          self.A= np.identity(self.hidden_dim , dtype = float)
39          self.A= torch.tensor(self.A, dtype=torch.float ,
    requires_grad = True)
40
41          self.linear_yhat= nn.Linear(self.hidden_dim , 1)
42
43
44      def encoder(self , features):
45
46          X= self.encoder1(features)
47          X= self.relu1(X)
48          X= self.encoder2(X)
49          X= self.relu2(X)
50          X= self.encoder3(X)
51          X= self.relu3(X)
52          miu= self.encoder4(X)
53          miu= self.relu4(miu)
54          sigma= self.encoder5(X)
55          sigma=self.relu5(sigma)
56
57          return miu,sigma
58
59
60      def decoder(self , features):
61
62        X= self.decoder1(features)
63        X= self.relud1(X)
64        X= self.decoder2(X)
65        X= self.relud2(X)
66        X= self.decoder3(X)
67        X= self.relud3(X)
68        X= self.decoder4(X)
69        X= self.relud4(X)
70
71        return X
72
```

```
73      def forward_features(self, inputs , labels):
74        out = inputs.unsqueeze(1)
75        out, (h_n, c_n) = self.lstm(out)
76        out = self.act1(out)
77        return out,(h_n , c_n)
78
79
80
81      def reparameterize(self, mu, logvar):
82        std = torch.exp(0.5*logvar)
83        eps = torch.randn_like(std)
84        return mu + eps*std
85
86      def nearest_neighbors(self, X_train, y_train , features_i ,
        labels):
87        X_list= features_i
88        X_list=X_list.detach().cpu()
89        labels=labels.detach().cpu()
90        X_train = X_train.detach().cpu().numpy()
91        y_train = y_train.detach().cpu().numpy()
92        features_i= features_i.detach().cpu().numpy()
93
94        nsamples, nx, ny = X_train.shape
95        d2_train_dataset = X_train.reshape((nsamples,nx*ny))
96
97        nsamples, nx, ny = features_i.shape
98        d2_f_x = features_i.reshape((nsamples,nx*ny))
99        n=int(math.sqrt(nsamples))
100       if n>=1000:
101         n=1000
102       knn_model = KNeighborsRegressor(n_neighbors=n)
103       knn_model.fit(d2_train_dataset, y_train)
104       distances, indicies= knn_model.kneighbors(d2_f_x)
105
106       for x in X_list:
107         klatest=0
108         ylatest=0
109         for j in indicies:
110
111           f_j = X_train[j]
112           labels_j = y_train[j]
113           f_j=torch.tensor(f_j)
114           for k in f_j:
115             equal=torch.allclose(x, k)
116             if equal is False:
117              x=self.A*x
118              k=self.A*k
119              dij= torch.dist(x , k , p=2)
120              kij=1/math.sqrt(2*math.pi) * torch.exp(-1*dij)
```

```
121                 klatest+=kij
122                 ylatest+=labels* kij
123
124         yhat= ylatest/klatest
125         return yhat
126
127
128     def forward(self, inputs, labels ,xlist , ylist):
129         out, (fxi_h , fxi_c) = self.forward_features(inputs, labels)
130         xlist = torch.cat((xlist , out), 0)
131         ylist = torch.cat((ylist , labels), 0)
132         y_hat = self.nearest_neighbors(xlist , ylist ,out, labels )
133         y_hat=y_hat.to(device)
134         labels= labels.reshape(len(labels),1)
135         out= out.reshape(len(labels) , self.hidden_dim)
136         features = torch.cat([out, labels], 1)
137         miu,sigma= self.encoder(features)
138         z= self.reparameterize(miu,sigma)
139         features = torch.cat([z, out], 1)
140         pdf= self.decoder(features)
141         return pdf,y_hat,miu,sigma
```

Bibliography

[1] Frank J Aherne, Neil A Thacker, and Peter I Rockett. «The Bhattacharyya metric as an absolute similarity measure for frequency coded data». In: *Kybernetika* 34.4 (1998), pp. 363–368.

[2] Aurélien Bellet, Amaury Habrard, and Marc Sebban. «A survey on metric learning for feature vectors and structured data». In: *arXiv preprint arXiv:1306.6709* (2013).

[3] Jane Bromley et al. «Signature verification using a" siamese" time delay neural network». In: *Advances in neural information processing systems* 6 (1993).

[4] Kyunghyun Cho et al. «Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation». In: *arXiv preprint arXiv:1406.1078* (2014).

[5] Sumit Chopra, Raia Hadsell, and Yann LeCun. «Learning a similarity metric discriminatively, with application to face verification». In: *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. Vol. 1. IEEE. 2005, pp. 539–546.

[6] Yueqi Duan et al. «Deep localized metric learning». In: *IEEE Transactions on Circuits and Systems for Video Technology* 28.10 (2017), pp. 2644–2656.

[7] Ahmed Elgammal, Ramani Duraiswami, and Larry S Davis. «Probabilistic tracking in joint feature-spatial spaces». In: *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings.* Vol. 1. IEEE. 2003, pp. I–I.

[8] Weifeng Ge. «Deep metric learning with hierarchical triplet loss». In: *Proceedings of the European Conference on Computer Vision (ECCV)*. 2018, pp. 269–285.

[9] Amir Globerson and Sam Roweis. «Metric learning by collapsing classes». In: *Advances in neural information processing systems* 18 (2005).

[10] Raia Hadsell, Sumit Chopra, and Yann LeCun. «Dimensionality reduction by learning an invariant mapping». In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. IEEE. 2006, pp. 1735–1742.

[11] Elad Hoffer and Nir Ailon. «Deep metric learning using triplet network». In: *International workshop on similarity-based pattern recognition*. Springer. 2015, pp. 84–92.

[12] Thien Khai Tran and Tuoi Thi Phan. «Deep learning application to ensemble learning—the simple, but effective, approach to sentiment classifying». In: *Applied Sciences* 9.13 (2019), p. 2760.

[13] Mahdi Khodayar et al. «Convolutional graph auto-encoder: A deep generative neural architecture for probabilistic spatio-temporal solar irradiance forecasting». In: *arXiv preprint arXiv:1809.03538* (2018).

[14] Diederik P. Kingma and Max Welling. «An Introduction to Variational Autoencoders». In: *Foundations and Trends® in Machine Learning* 12.4 (2019), pp. 307–392. ISSN: 1935-8237. DOI: 10.1561/2200000056. URL: http://dx.doi.org/10.1561/2200000056.

[15] Jingtuo Liu et al. «Targeting ultimate accuracy: Face recognition via deep embedding». In: *arXiv preprint arXiv:1506.07310* (2015).

[16] Kameo Matusita. «Decision rules, based on the distance, for problems of fit, two samples, and estimation». In: *The Annals of Mathematical Statistics* (1955), pp. 631–640.

[17] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. «Exploiting similarities among languages for machine translation». In: *arXiv preprint arXiv:1309.4168* (2013).

[18] Lars Ruthotto and Eldad Haber. *An Introduction to Deep Generative Modeling*. 2021. DOI: 10.48550/ARXIV.2103.05180. URL: https://arxiv.org/abs/2103.05180.

[19] Mohsen Saffari et al. «Deep convolutional graph rough variational autoencoder for short-term photovoltaic power forecasting». In: *2021 International Conference on Smart Energy Systems and Technologies (SEST)*. IEEE. 2021, pp. 1–6.

[20] Jürgen Schmidhuber. «Deep learning in neural networks: An overview». In: *Neural networks* 61 (2015), pp. 85–117.

[21] Edgar Simo-Serra et al. «Discriminative learning of deep convolutional feature point descriptors». In: *Proceedings of the IEEE international conference on computer vision*. 2015, pp. 118–126.

[22] Kihyuk Sohn. «Improved deep metric learning with multi-class n-pair loss objective». In: *Advances in neural information processing systems* 29 (2016).

[23] Kilian Q Weinberger and Lawrence K Saul. «Distance metric learning for large margin nearest neighbor classification.» In: *Journal of machine learning research* 10.2 (2009).

[24] Kilian Q Weinberger and Gerald Tesauro. «Metric learning for kernel regression». In: *Artificial intelligence and statistics*. PMLR. 2007, pp. 612–619.

[25] Yandong Wen et al. «A discriminative feature learning approach for deep face recognition». In: *European conference on computer vision*. Springer. 2016, pp. 499–515.

[26] Chao-Yuan Wu et al. «Sampling matters in deep embedding learning». In: *Proceedings of the IEEE international conference on computer vision*. 2017, pp. 2840–2848.

[27] Eric Xing et al. «Distance metric learning with application to clustering with side-information». In: *Advances in neural information processing systems* 15 (2002).

[28] Dong Yi et al. «Deep metric learning for person re-identification». In: *2014 22nd international conference on pattern recognition*. IEEE. 2014, pp. 34–39.

[29] Jun Yu et al. «Deep multimodal distance metric learning using click constraints for image ranking». In: *IEEE transactions on cybernetics* 47.12 (2016), pp. 4014–4024.