Project Report On

# An approach towards Font-Recognition using Prototypical Networks

"A dissertation submitted in fulfilment of the requirements of 8th semester 2024 project - III (PROJ-CS881) examination in Computer Science and Engineering of the Maulana Abul Kalam Azad University of Technology"


Submitted By

Mangalam Sharma (10200120018)

Akash Paramanik (10200120027)

Swagata Shyamal (10200120040)

Sk Babusona (10200120052)

Under the guidance of

Dr. Kousik Dasgupta

Assistant Professor

Dept. of Computer Science And Engineering

Kalyani Government Engineering College

**Department of Computer Science and Engineering**

**Kalyani Government Engineering College**

(Affiliated to Maulana Abul Kalam Azad University of Technology, West Bengal)

Kalyani - 741235, Nadia, West Bengal

सत्यमेव जयते
কল্যাণী গভঃ ইঞ্জিনিয়ারিং কলেজ
KALYANI GOVERNMENT ENGINEERING COLLEGE
(GOVERNMENT OF WEST BENGAL)
KALYANI, NADIA, PIN - 741 235, WEST BENGAL

Memo No. :

Date :

# *Certificate of Approval*

This is to certify that this report of the B.tech final year (8th semester) 2024 project, entitled **"An approach towards Font-Recognition using Prototypical Networks"** is a record of the bonafide work carried out by Mangalam Sharma(10200120018), Akash Paramanik (10200120027), Swagata Shyamal (10200120040), Sk Babusona (10200120040) under my Supervision and guidance.

In my opinion, the report in its present form is in partial fulfilment of all the requirements, as specified by the **Kalyani Government Engineering College** and as per regulations of the **Maulana Abul Kalam Azad University of Technology**. In fact, it has attained the standard necessary for submission. To the best of my knowledge, the results embodied in this report are original in nature and worthy of incorporation in the present version of the report for the project- III (CS-881) 8th semester B.Tech program in Computer Science and Engineering in the year 2024.

**Guide / Supervisor**

**Dr. Kousik Dasgupta**
Assistant Professor
Computer Science and Engineering
Kalyani Government Engineering College

**Examiner(s)**

**Head of the Department**
Computer Science and Engineering
Kalyani Government Engineering College

# ACKNOWLEDGEMENT

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organisations. We would like to extend our sincere thanks to all of them.

We are highly indebted to our project guide Dr. Kousik Dasgupta for his guidance and constant supervision as well as for providing necessary information regarding the project and also for his support in completing the project.

We would like to express our gratitude towards our parents for their kind cooperation and encouragement which helped us in completion of the project.

We would like to express our special gratitude and thanks to all the respected faculty and staff members of the Computer Science and Engineering department for providing technical and hardware assistance and for their commitment to help all students as much as possible. We would like to extend our gratitude to our peers who kept us motivated in this pursuit.

Finally, we express our gratitude to Dr, Sourabh Kumar Das, Principal, Kalyani Government Engineering College.

Mangalam Sharma Roll No - 10200120018

Akash Paramanik Roll No - 10200120027

Swagata Shyamal Roll No - 10200120040

Sk Babusona Roll No- 10200120052

# Abstract

We present a Few-Shot Learning (FSL) algorithm that uses a Prototypical Network for Font Recognition applied on a synthetic dataset in this project. Prototypical Network is an architecture designed for metric based meta-learning, which aims at learning via plotting prototypes of different classes on the representation space and using distance metric to classify, thus this can be easily computed and can be used to support efficient few-shot classification. In addition, it has convolutional layers first which are followed by fully connected layers helping in the extraction of discriminative features from input images.

The training process consists in optimising the model with Adam optimizer by minimising Cross-Entropy loss function during each epoch, and then training it on batches of data, updating the training loss periodically to check whether or not the model has converged. Furthermore, dropout layers are introduced to improve model generalisation whereas weight decay is employed to regularise the training procedure.

Regarding this project, it will contribute towards enhancing font recognition technology through use of FSL approaches that facilitate effective learning from a small number of annotated examples thereby lessening reliance on large labelled datasets for training purposes.

**Chapter 1.**

# Introduction

Various industries like optical character recognition (OCR), document analysis, and graphic design have a huge dependence on font recognition. Correctly identifying fonts from text images is highly critical in tasks such as digitising historical documents, to producing aesthetic designs. Many traditional methods of identifying fonts often involve the use of deep learning models that have been trained using vast datasets with many samples from each typeface. Nonetheless, getting enough labelled data can be difficult or expensive.

Few-shot learning (FSL) techniques have been developed as a new way forward for font recognition problems such as these. FSL is designed to develop models that can learn from only a few labeled examples per typeface which imitates how humans acquire new knowledge with limited help. One well-known method for FSL is Prototypical Network, which learns feature space where class prototypes can be quickly calculated making it easier to adapt to unseen classes.

This project aims at using the Prototypical Network framework for few-shot learning based font recognition tasks specifically handwritten digit recognition by employing Synthetic dataset as proxy of font styles. The network architecture of the Prototypical Network extracts the most meaningful features
Instead, we intend to illustrate this by training our model on examples from a minimum number of each font style and demonstrate that it can generalizee to an unseen font while still achieving performance comparable to traditional supervised learning approaches.

This project is not only important in terms of the development of font recognition technologies but also demonstrates the potential for FSL techniques to address common data annotation bottlenecks in machine learning. We will rely on empirical evaluation and analysis to provide information on how effective and practical FSL methods are for font recognition in real life contexts.

## 1.1 Motivation

The reason for this project is the problems with normal ways of recognizing fonts. Those ways mostly need a lot of labeled data before they work well. Getting all that data is hard work and takes a bunch of time and money, specially when working with uncommon fonts or different languages. On top of that, those classic approaches do not adapt well to new fonts unless you retrain them with even more data.

Few-shot learning techniques offer a nice alternative here. Instead of huge datasets, these models can learn from a small number of examples for each font style we'd like to recognize and this is really useful for real-life settings where getting tons of labeled data is not practical like with historical or unusual fonts.

The Prototypical Neural Network is one famous few-shot learning technique that aims to extract very distinctive patterns and calculate prototypes for each class, which helps the model adapt rapidly to new fonts with very little data. By expanding on few-shot learning processes, this project aims to improve on existing shortcomings in font recognition, working towards more efficient and versatile systems.

More importantly, the use of FSL to character recognition can have implications beyond the objective of improving recognition accuracy. For example, it could help in the improvement of the automation of OCR systems, in the enhancement of the usefulness of digital typography tools, and, in a sense, it can empower the designer with potentially more creative ways to recognise fonts. The author hopes that some future work can explore the application of FSL to font recognition tasks, whatever forms they take, contribute to the application of ML methods to more real-world learning problems, and promote the science and technology development in the related human-centric fields such as document analysis and graphic design domain.

## 1.2 Background

Few-shot learning (FSL), in recent years, has come as a promising alternative to traditional supervised learning methods. FSL aims at training models that can generalize from a small number

of labeled examples per class, similar to the ability of humans to learn. FLS through use of meta-learning algorithms and metric learning techniques allows for fast adaptation of models into new classes or tasks with limited supervision.

One popular framework for FSL is the Prototypical Network introduced by Snell et al. (2017). Prototypical Networks learn a metric space where class prototypes can be efficiently computed for few-shot classification tasks. This approach has found success in different domains including image recognition, natural language processing, and computer vision.

The motivation behind applying FSL techniques for font recognition tasks stems from the challenges inherent in conventional supervised learning approaches. In particular, when dealing with a wide variety of font styles collecting labeled data for font recognition can be laborious and expensive. Additionally, new fonts may not be well generalized by old or experienced handwriting models.

By exploring FSL for font recognition, we aim to address these challenges and develop more efficient and adaptable font recognition systems. The Prototypical Network serves as a foundational framework for our project, enabling us to learn discriminative representations of font characters and compute class prototypes for efficient classification. Through empirical evaluation and experimentation, we seek to demonstrate the effectiveness of FSL techniques in font recognition tasks and contribute to advancing the state-of-the-art in this domain.

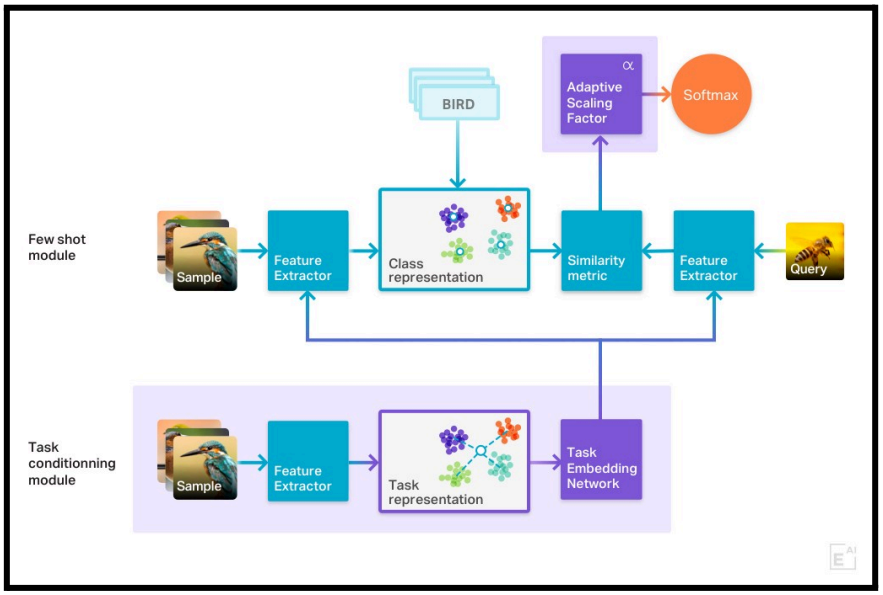


Figure 1.2 Few-shot learning at a glance

**Chapter 2.**

# Related Works

Few-shot classification is the task in which a classifier must be adjusted to board in new classes not present in training, given only a few samples of each of these classes. A primitive approach, such as re-training the model on the new data, would critically overfit. While the problem is quite difficult, it has been showcased that humans have the capability to execute even one-shot classification, where only one example of each new class is given, with great accuracy and precision .

Few shot Learning can be categorised into 2 sections namely Meta Learning Methods and Transfer Learning Methods.

Meta learning methods: It involves using episodic training strategy so that it can generalize each of the new tasks from the base-class data using only a few examples. There are various types of Meta-Learning Methods:-

Metric based Meta Learning: Some methods use the distance metric from the few-shot examples of the base class. Eg Prototypical network that has a prototype of each class and measures the distance from the queries.

Optimization based Meta Learning: The looks to find good optimization direction that converges faster to the optimal solution with fewer gradient steps

Transfer Learning Methods: It involves training the base model using traditional methods using large datasets and preparing a good base model and then adapting the model to new classes with only a few examples. Cosine-similarity is usually used to build a classifier with a frozen feature extractor. Chen et al in his paper systematically analyse the performance of preparing a cosine classifier with freezing feature extractor compared it to the popular meta learning methods.

In recent times there has been a lot of development in this field of few Shot learning, what once started as an exploratory study has advanced a lot since the blooming of the Large Language Models and generative AI. In a study Few-Shot Learning of Video Object Detection in a Transfer Learning Scheme which used two types of datasets to one weak and one strong was used to identify objects using Joint freeze where 500 iterations of 1-shot and 2000 iterations of 2-shot and 3-shot to find out the accuracy.

## Artificial Intelligence

This range of time was when the interest in AI really came to a head. Alan Turing published his work "Computer Machinery and Intelligence" which eventually became the Turing Test, which experts used to measure computer intelligence. The term "Artificial Intelligence" was coined and came into popular use.Artificial intelligence refers to the development of computer systems in a way such that they simulate human intelligence. We can achieve this through AI concepts such as Machine Learning, Expert Systems, Neural Networks, and Reinforcement Learning. AI can also be categorized into different types depending on its level of autonomy, such as Assisted Intelligence, Augmented Intelligence, and Autonomous Intelligence. AI systems act as agents that employ reasoning and problem-solving techniques, like Backward Chaining, to achieve intelligent behavior. Thus, AI is a strong tool with applicability to a number of applications.The human brain is a prediction machine. It sees patterns, then makes predictions from previous experiences. This aspect of human intelligence has been critical to our survival. For example, many years ago, a forager might have eaten a particular berry, gotten sick, and thus learned the clues that indicate that a berry is poisonous. This would happen automatically—we'd get nauseous on seeing the berry again, which would make us clear it. In other words, our brain and nervous system make a prediction that dictates our outcomes.Artificial intelligence follows the same approach. It is trained, learns through trial and error (though it can employ other techniques, like deep learning, too), and then predicts outcomes. Just as we learn not to eat poisonous berries, or to wait for our coffee to cool down so as not to destroy our taste buds for half a day, AI model training teaches an AI system how to respond to a set of conditions. That way, an organization can offload a lot of the 'grunt' work to an AI. And combined with other automation tools, AI can enhance a wide range of business processes—from advanced customer service to rapid time to market for products.Training the artificial intelligence models accordingly can make AI very transformative for your organization. This post will go over the main steps in training AI models and different ways of using enterprise artificial intelligence.Artificial intelligence can be defined as a program that can learn and think for itself. In this view, it's possible to consider anything that constitutes an artificial intelligence if it's a program that's performing a task that we will normally presume a human would be performing.The merits of artificial intelligence are many, but there are also demerits. The merits are efficiency in regard to task automation, analysis of big data for informed decisions, medical diagnosis, and creation of autonomous vehicles. The demerits are job displacement, human issues in regard to bias and privacy, security risks due to hacking, human-like imagination and empathy.Let's start with the merits of artificial intelligence.

## Generative AI

Generative AI is a type of artificial intelligence technology that has the capability to create different types of content, such as text, imagery, audio, and synthetic data. The recent fad around generative AI has been the simplicity of new user interfaces that create high-quality text, graphics, and videos in a matter of seconds.The technology, to be sure, is not new. Generative AI was introduced in the 1960s, in chatbots. But not until 2014, when generative adversarial networks—or GANs, a type of machine-learning algorithm—hit the market, could generative AI produce convincingly authentic images and videos and audio of real people.Like any other project, a GenAI project begins with Problem Definition, where the challenge/opportunity is defined that the GenAI Application will address. The second step is Data Investigation, in which the team selects the data to augment and/or train the GenAI Large Language Model, the latter being an advanced AI predictive model that processes and produces human-like text. Third, in Data Preparation, this data is organized for the best AI use. Fourth is Development, whereby the GenAI application is built by the team. Fifth comes Evaluation, where the reliability and usability of the AI-based application are tested. Successful testing leads to Deployment, the placing of the AI application within its operational environment. Finally, the cycle ends with Monitoring and Improvement, where feedback continuously tunes the AI application, ensuring its relevance and performance in the real world.Enterprise use of generative AI enables the large-scale creation of creative and engaging content. Take the advertising industry as an example: AI-powered systems can generate compelling ad copy, visuals, and even video content automatically—valuable for ideas that replace extensive creative manual work.\n\n"When we think about the future of the internet, I would guess that 90% of content will no longer be generated by humans. It will be generated by bots," says Latanya Sweeney, Professor of the Practice of Government and Technology at the Harvard Kennedy School and in the Harvard Faculty of Arts and Sciences. That means that generative AI saves valuable time and reduces operational costs by automating tasks that have otherwise needed human intervention. Generative AI, in architecture and design, for instance, can come up with building designs based on given specifications—that's considerably faster than the process of designing buildings used to be. More than that, it can generate new product concepts and designs with inputs from stakeholder feedback and market trends. The speed at which it is able to sift through mounds of data and make design recommendations is an immense source of its power.

## Machine-Learning

Machine learning is a subset of AI that uses algorithms trained on datasets to create self-learning models that predict outcomes and classify information without human intervention. Machine learning is applied today for an unbelievable number of commercial purposes—from suggesting products to consumers based on their past purchases to the prediction of stock market fluctuation to translation of text from one language to another. In ordinary usage, the two terms "machine learning" and "artificial intelligence" are often used as each other's synonyms, due to the omnipresence of machine learning for AI purposes in the world today. But the two terms are meaningfully distinct. While AI refers to the general attempt to create machines capable of human-like cognitive abilities, machine learning specifically refers to the use of algorithms and datasets to do so.

**Types of Machine Learning:**

There are several different types of machine learning that power the many different digital goods and services that we use every day. While each of these different types of machine learning attempts to accomplish similar goals — creating machines and applications that can act without human oversight — the precise methods they use differ somewhat. To help you get a better feel for how the types differ from one another, here's an overview of the four different types of machine learning primarily in use :

1. Supervised Learning: Supervised learning is a paradigm of machine learning in which the algorithm learns from labeled data. This approach includes training data with an input-output pair such that each input is associated with an output label. Under the paradigm of supervised learning, the inference of a mapping function from the input to the output is conducted for making predictions or decisions on unseen data. In this process, the algorithm adapts internal parameters to best fit the data; that is, the algorithm learns to minimize the discrepancy between its predictions and true labels in the training set. Once it has been trained, it generalizes its learned patterns over new, unseen data for accurately predicting the outputs of the inputs that it might not have seen. Supervised learning applies in many domains: image and speech recognition, natural language processing, regression analysis, and classification tasks. Being so

versatile and effective, it is considered one of the most popular approaches in machine learning.

2. Unsupervised Learning: Unsupervised learning represents an area of machine learning in which an algorithm learns patterns and structure from data without labels. Unlike in supervised learning, no predefined output labels are associated with the input data. The algorithm, in essence, is usually expected to find the hidden patterns or relationships in the data by itself. This is often a task of clustering, where data points that are similar are grouped, or the task of dimensionality reduction, where the algorithm aims to learn the most essential features of the data that describe it while discarding the rest. The methods of unsupervised learning algorithms are statistical and optimization techniques, which can discover meaningful representations from data without explicit guidance. Such techniques uncover insights, identify anomalies, or organize data in a meaningful manner; hence, unsupervised learning is invaluable in exploratory data analysis, data preprocessing, and feature engineering. Applications include customer segmentation, anomaly detection, recommendation systems, and data visualization. While unsupervised learning is much tougher than the other two, it still gives very valuable tools for understanding and drawing insights from voluminous and complex datasets.
3. Semi-supervised Learning: Semi-supervised learning is at the interface of supervised and unsupervised learning and aims to leverage both labeled and unlabeled data for better performance. It often makes sense when getting labeled data is either expensive or time-consuming, so the idea is to take advantage of the plethora of available unlabeled data, in addition to a small number of labeled examples. The algorithm learns on labeled data in a supervised way but also uses the inherent structure or patterns in the unlabeled data. It can be used to build a more robust and generalized model, as the unlabeled data provides information regarding the overall distribution. Some common semi-supervised learning techniques include self-training—iteratively labeling the unlabeled data points with confident predictions made by the model—and co-training, in which a diverse ensemble of multiple learners is trained on different views of the data and shares information during the training. Semi-supervised learning is used in such fields as natural language processing, computer vision, and bioinformatics, where labeled data is scarce but unlabeled data is abundant. This makes it a really strong framework, leveraging both labeled and unlabeled data, to deal with a lot of real-world machine learning challenges.

4. Reinforcement Learning: Reinforcement Learning is a paradigm of machine learning in which sequential decisions are made to maximize the expected reward from an environment. Contrary to supervised learning, where algorithms are trained on labeled data, and unsupervised learning, which aims to discover patterns in unlabeled data, RL learns from the feedback of rewards and penalties. It takes sequential actions in the environment to maximize cumulative rewards over time, based on the consequences of those actions. Reinforcement learning seeks to balance exploration—the process of trying new actions to discover their effects—with exploitation, or leveraging known actions that yield high rewards. Other common RL techniques include Q-learning, in which the value of taking a particular action given a certain state is learned, and policy gradients, in which the agent directly learns a policy that maps states to particular actions. Reinforcement learning has been found to be useful in an extremely large set of applications, including game playing—for example, AlphaGo—robotics, autonomous vehicles, and even recommendation systems. The power of learning through trial and error makes it very suitable for problems in which the environment is dynamic and the optimal decision strategy is not well specified in advance.

Low requirement of computational resources: Traditional CNN requires a large amount of training data to achieve a good result. The training of such large datasets requires not only a powerful computational resource but also takes a long time, for example, hours, days, or even weeks. Transfer learning techniques can be performed based on the existing resources (datasets or models) and a small amount of local training data. It saves storage space and runtime.

## Meta-Learning

In computer science, meta-learning studies and approaches started in the 1980s and became popular after the works of Jürgen Schmidhuber and Yoshua Bengio on the subject. Marcial Losada and other researchers tried to develop a meta-learning model to analyse teams and relationships. A 2013 paper offered a scathing critique of this attempt as a misapplication of complex mathematical modelling, leading at least one former proponent to abandon it.

Meta-learning, also known as "learning to learn", is a subset of machine learning in the domain of computer science. It is used in enhancing the results and performance of any learning algorithm by changing some aspects of it due to experiment results. Meta-learning helps researchers to understand which algorithm(s) generate the best/better predictions from datasets.

To understand how meta-learning works, in simple terms, let's say we have a robot that can pick up items and put them where they belong. However, the robot has not been trained to pick up an object that it's never seen. In a traditional paradigm of supervised learning, we need to first collect a huge dataset of images of the new object with its location coordinates and train the robot for each new object.

Finally, with meta-learning, we can train the robot on how to quickly adapt to new objects by leveraging its previous experience with other objects. It could learn the common features of the objects—like their shape or texture—so that it can adapt quickly to new objects. In the setup of the training process for meta-learning, we are using few-shot learning.

A key advantage of meta-learned systems lies in their flexibility and potential for fast adaptation from little data. As such, the drawbacks of traditional machine learning algorithms can be overcome—no need for large datasets, no high costs of training, no substantial efforts resulting from many training trials, and no need for extensive hyperparameter tuning. No long training times. On the other hand, meta-learning reduces the need for large amounts of task-specific training data, and this is especially important in applications where obtaining high-quality labelled data is rare, costly, or time-consuming. This reduced demand for data also means that there is a reduction in training time and cost during the task adaptation. In all, a well-trained meta-learned system is a generalised model, and this model could be efficiently used to solve many related tasks instead of one. Furthermore, individual tasks can achieve higher prediction accuracy by exploiting insights from the other tasks.

## Transfer-Learning

In 1976, Bozinovski and Fulgosi addressed the problem of transfer learning in neural network training. The paper gives a mathematical and geometrical model of the topic. In 1981, a report considered the application of transfer learning to a dataset of images representing letters of computer terminals experimentally, demonstrating positive and negative transfer learning.

Transfer Learning is one of the most powerful techniques in deep learning. By harnessing the ability to use existing models and their knowledge of new problems, transfer learning has opened doors to training deep neural networks even with limited data. This is a very important breakthrough in data science, where practical scenarios usually have less than optimal labeled data. This article peels the layers of transfer learning and shows how it powers data scientists to

accomplish their complex challenges with newfound efficiency and effectiveness. Usually, in computer vision, neural networks try to detect edges in the earlier layers, shapes in the middle layer, and some task-specific features in the latter layers. In transfer learning, we will use the early and middle layers, and we just retrain the latter layers. It helps leverage the labeled data of the task it was initially trained on.

Let's come back to the example of the model that was trained for the recognition of a backpack on an image that is going to be used for the recognition of sunglasses. In the early layers, the model has learned to identify objects; because of that, we will retrain only the latter layers so that it can learn what makes sunglasses different from other objects.

Transfer learning requires a small amount of local training data. Transfer learning consists of two phases: a pre-trained model and a fine-tuned model. The pre-trained model is trained with an open dataset or some other available resources. A fine-tuned model is fine-tuned from a pre-trained model with training local datasets. However, acquiring local training data is labor-intensive and costly. It has been one of the most important factors that hinder the advancement of deep learning technology. Take building detection as an example, a model can be pre-trained by using training datasets from the US, UK, or other countries. In order to make a good prediction from Finnish datasets, we need to train the model with a small amount of Finnish local datasets. Emilia compared the prediction by making use of transfer learning and without transfer learning in her Master thesis. If one does not make use of transfer learning, a large amount of local training data is needed in order to get a similar result when making use of transfer learning.

Datasets for the fine-tuned model can be different from the ones for the pre-trained model. Under the conventional deep learning method, such as CNN, it is exclusively used for the same type of datasets. For instance, we trained a CNN model with satellite images; without the use of the transfer learning technique, the model can predict satellite images only. However, under the transfer learning technique, the data used for the initial model can be different from the fine-tuned model. For example, the initial model was trained with satellite images but the fine-tuned model can be aerial images or satellite images with different spatial resolutions and so on. However, in case of a massive difference of resolutions or geometries between the pre-trained model and the fine-tuned model, fine-tuning efforts are needed. In addition, the classifier from the fine-tuned model can be different from the pre-trained model. For example, the pre-trained model classifies cars from the scene data, and the fine-tuned model can be trucks or other vehicles.

## Few-shot Learning

Few-shot learning is a special branch of machine learning that seeks to solve problems where the model is to be trained on a limited number of labeled data. This comes in contrast to the traditional machine learning paradigm, which needs big labeled datasets to learn. It seeks to generalize with merely a few examples per class or task, a domain that is especially relevant in situations where collection of abundant labeled data is infeasible or too expensive, for example, in medical diagnosis or personalized recommender systems. Few-shot learning techniques typically involve methods such as meta-learning, where the model learns to adapt to new tasks quickly based on just a few examples, or transfer learning, where the knowledge learned from training on one task is transferred to a new, related task with little data. Few-shot learning makes models generalizable from small datasets, a promise for applications with real-world data scarcity, a potential way to extend machine learning reach to diversified domains where labeled data is hardly available.

Few-shot Learning (FSL) is divided into few parts. The types of few shot learning are -

1. N-shot Learning (NSL) : N-shot learning, a generalization of few-shot learning, implies scenarios where a machine learning model is exposed to a limited number of examples, N, per class or task. This is to deal with challenges in those areas where it is very hard or expensive to build large amounts of labeled data. These small numbers of labeled examples for classes or tasks enable N-shot learning algorithms to generalize well to new unseen instances. The flexibility of N-shot makes it applicable in a wide domain of applications, such as image recognition, natural language processing, and medical diagnosis. Some commonly used techniques in N-shot learning are meta-learning, where models are trained to learn from a variety of N-shot tasks, and episodic training, where episodes consisting of N examples per class are used for training. By providing the ability to models to learn from limited data, N-shot learning provides opportunity in applications where data availability is constrained and hence assists in the development of more robust and adaptive machine learning systems.
2. Zero-shot learning (ZSL): Zero-shot learning is an innovative method in machine learning that seeks to train models for object recognition and classification without

requiring labeled examples from all classes during model training. As opposed to learning merely from the labeled data, zero-shot learning enables models to generalize to unseen classes by exploiting semantic descriptions or attributes of those classes. Zero-shot learning will allow models to make predictions for novel classes at inference time by learning relationships between seen and unseen classes based on shared attributes. This paradigm is particularly valuable for such domains where labeled data for all possible classes is not practically or cost-effectively achievable, or in cases of fine-grained categorization or cross-domain recognition. Usually, the semantic representations of classes are embedded into a common semantic space, enabling models to learn relationships between the seen and unseen classes. Through this innovative way of generalization, zero-shot learning opens up whole new lines of machine learning applications in various and changing domains with its adaptability and scalability.

3. One-shot learning (OSL) : One-shot learning is a sub-area of machine learning dealing with training models with just one example per class or task. In contrast to mainstream paradigms of machine learning, which require a big volume of labeled data for training, one-shot learning intends to generalize from minimal data, hence becoming extraordinarily vital in conditions of scarce and costly data acquisition. Learning to differentiate between classes with one example is evidence of the adaptability and efficiency of one-shot learning algorithms. These models usually adopt siamese networks or metric learning, which compare and classify instances with regard to a reference example. The applications of one-shot learning are found across several applications, including image recognition, handwriting recognition, and facial recognition, where scarcity of data is a significant challenge. One-shot learning opens up possibilities for developing robust and scalable machine learning systems with applications in the real world, given that it offers a practical solution for learning from limited data.

Several existing algorithms and techniques have been developed for few-shot learning tasks, including font recognition. Here are some of the prominent ones:

## Matching Networks

The Matching Networks of Vinyals et al. proposed a meta-learner that is able to learn a function of similarity between a query sample and examples in a support set. They make use of attention mechanisms that dynamically scale the contribution of different examples in the support set for the classification of the query sample.

Figure 2.1 shows that Matching networks consist of an embedding network that maps input samples to a representation space and an attention mechanism that calculates a weighted sum of the support set embeddings to a context vector for classification.

Matching Networks [Figure 2.1], during training, optimize a loss function that pushes the model to predict the right label for query samples based on information brought in by the support set. This is made possible by the attention mechanism, which allows the model to focus on relevant support set examples for each query.

Matching Networks allow flexibility and adaptability in few-shot learning scenarios, where models are to use the information from support set examples effectively by the attention mechanism.

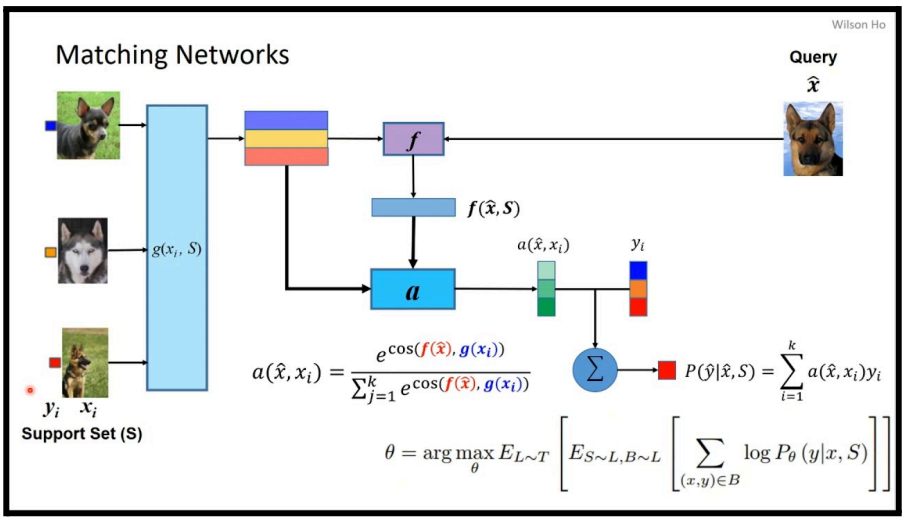


Figure 2.1 Matching Networks

## Relation Networks

The Relation Networks, proposed by Santoro et al. in 2018, learn a relation function that measures the relation of a query sample with every class prototype. They model the relation function as a neural network that takes the concatenation of a query sample and a class prototype as input.

The Relation Networks consist of an Embedding Network for feature extraction followed by a relation module, which computes the similarity score between a query sample and each of the class prototypes.

During training, we can obtain from figure 2.2 that Relation Networks optimize a loss function that guides the model to predict the correct class label for query samples, based on the computed relation scores. The relation module learns to capture the similarity between query samples and class prototypes.

Relational networks can enable flexibility and scalability in few-shot learning tasks by capturing complex relationships between query samples and class prototypes with learnable relation functions.

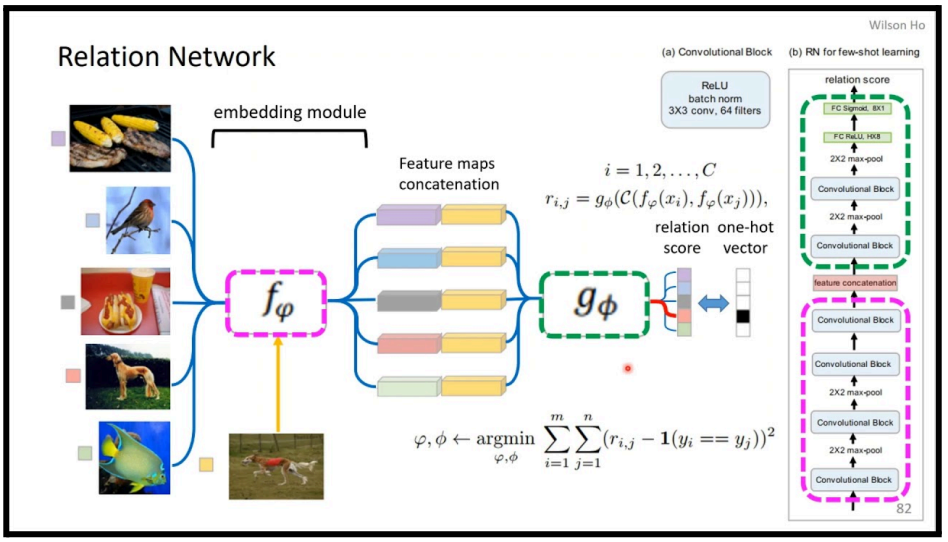


Figure 2.2 Relation Network

These algorithms, along with their variations and extensions, have been applied to various few-shot learning tasks, including font recognition. By leveraging these techniques, researchers aim to develop more efficient and adaptable models that can generalize effectively from limited labelled

examples, thereby addressing the challenges posed by data scarcity and domain adaptation in real-world applications.

## Siamese Network

The Siamese neural networks were introduced by Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah in 1994. However, Gregory Koch introduced Siamese networks in 2015.

The Siamese Neural Network is an artificial neural network that utilizes the same weights while working in tandem on two different input vectors to compute comparable output vectors. It is observed from the figure 2.3 Often one of the output vectors is precomputed, thus forming a baseline against which the other output vector is compared. This is similar to comparing fingerprints, or more technically, a distance function for Locality-sensitive hashing.

Siamese network is an artificial neural network that contains two or more identical sub-networks. In other words, they have the same configuration with the same parameters and weights.

Mostly, we only train one of N (the number of subnetworks chosen for solving the problem) the subnetworks and use the same configuration (parameters and weights) for other sub-networks.

The aim of Siamese network networks is to find the similarity of the inputs by comparing their feature vectors.

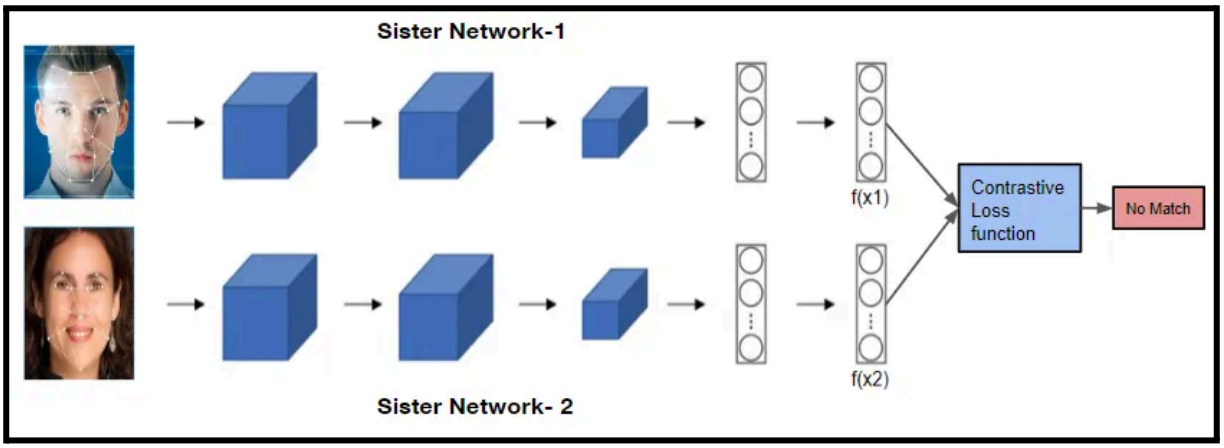


Figure 2.3 Siamese Network

The advantage of the Siamese network is that unlike other traditional neural networks in deep learning, we need not have too many instances of a class. A few are enough to build a good model.The biggest advantage of the Siamese network is that in the case of face detection

applications like attendance, when we have a new employee or class for our model. For the network to detect his/her face, the model only requires a single image of his/her face. Using the single image as the reference image, the network will calculate the similarity score for any new instances presented to it. That's the reason we say that the network predicts the score in one shot and we say it's a one-shot learning model.

**Chapter 3.**

# Proposed Methodology: Font-Recognition

This project proposes a method for identifying fonts from document images in cases where there is limited annotated data. The low availability of labelled data poses the problem of not having good generalisation of neural networks across novel font classes, as these networks tend to overfit when trained on very few data samples. Moreover, training a deep neural network repeatedly for all the classes whenever a new font class is to be identified is a very time-consuming process. Therefore, font identification networks should also have the capability to quickly adapt to novel classes with much less annotated data. This motivates us to explore state-of-the-art few-shot learning-based approaches like prototypical networks for font identification from text images. Therefore, the problem of font identification is formulated as a few-shot classification problem, whereby a prototypical network is trained on text images of most commonly occurring fonts to learn font embeddings in low data scenarios. After this, the trained network is adapted to new font classes quickly whenever a new font image comes by computing the embeddings based on the support set and then assigning labels to the query image using the Euclidean distance measure.

The proposed Font-Recognition architecture consists of two stages: the Meta-training stage and the Meta-testing stage. In the meta-training stage, the prototypical networks are trained to learn representations of known/ most common fonts for which annotated data is available. This stage acquires the cluster representations using the embeddings of the support set examples. The network receives input images sized $64 \times 64$, having text images only of a particular font. At the meta-test stage, the support set images of the given font class pass through the trained network to acquire the embeddings, followed by the calculation of cluster representation. The test image is, then mapped to the label of the font class having the smallest Euclidean distance. The details of the prototypical networks and the network architecture of the convolutional network used are given in the following subsections.

## 3.1 Prototypical Networks

Prototypical Networks in few-shot learning form a complex approach for handling cases where only a small amount of labelled data is available for training.

These architectures operate through learning a representation space that clusters examples together for each class and separates examples from different classes.

Prototypes form a core part of prototypical networks. In essence, each class in the dataset is represented by a prototype vector, which is the central representation of that class in the learned feature space.
During the training phase, it computes prototypes based on a support set, which usually consists of a few examples for every class in the dataset. These prototypes are computed by taking an average of the feature embeddings from examples belonging to the same class.

At inference time, upon input of a query example, the network computes its feature embedding and measures its similarity or distance to all prototypes in the feature space. Then, it classifies the query example as belonging to the class of the nearest prototype.

Prototypical Networks have shown impressive performance on a variety of few-shot learning tasks due to their ability to use the available labelled data effectively and generalise to new classes with only a few examples.

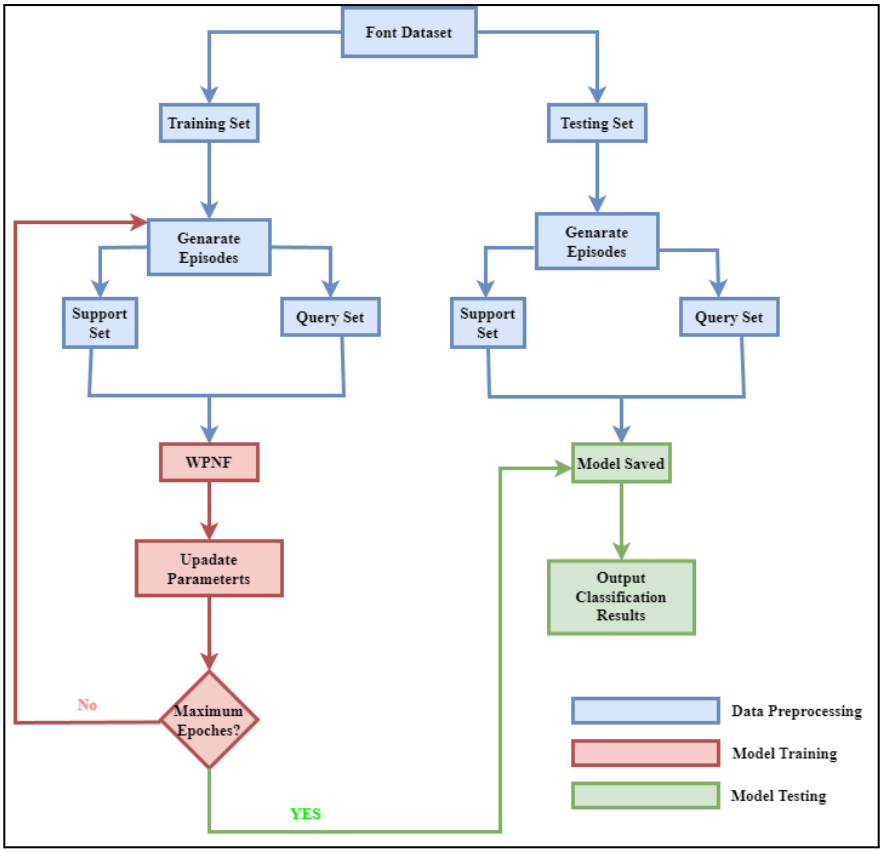


Figure 3.1 Representing Prototypical network flowchart

## 3.2 Prerequisite conditions

Prerequisites of the Prototypical Networks algorithm for this project include:

Few-shot Learning Setup: Prototypical Networks are designed for few-shot learning tasks—tasks designed with only a few labelled examples per class for training. In this project, the model is trained on a set of support examples composed of a limited number of examples per class.

Neural Network Architecture: Prototypical Networks usually utilise neural network architectures, such as convolutional neural networks, to learn meaningful feature representations from the input data. Used in this project is an architecture composed of a series of convolutional layers followed by one fully connected layer.

Metric Learning Objective: The algorithm is based on the principles of metric learning, where it aims to learn a feature space in which samples from the same class are closer together compared to samples from different classes. The model is trained with the aim of minimising a suitable loss function which encourages this behaviour—such as cross-entropy loss.

Distance Metric: Prototypical Networks rely on a distance metric, such as Euclidean distance, to measure the similarity between query samples and class prototypes in the learned feature space. This metric is applied during both training and testing phases for classification.
GPU Availability: While not a strict requirement, the algorithm can benefit from GPU acceleration for training to be faster, particularly when dealing with large datasets or complex architectures of neural networks. The provided code in this project checks if a GPU is available and uses it if present.

In general, the prerequisite conditions for the Prototypical Networks algorithm in this project are to have access to labelled data, set up a few-shot learning scenario, define a suitable neural network architecture, formulate a metric learning objective, specify a distance metric, and optionally use GPU resources for accelerated training.

## 3.2 Specialties

There are several unique features and advantages of using Prototypical Networks in the realm of few-shot learning and classification tasks:

Few-shot Learning: Prototypical Networks have proven to be excellent in scenarios where only a few labelled examples per class are available for training. This makes them particularly useful in tasks where collecting large amounts of labeled data is either impractical or too costly.
Metric Learning: Prototypical Networks are grounded on the principles of metric learning. By learning a proper distance metric in a low-dimensional embedding space, the network can capture the similarity between samples belonging to the same class while enforcing the dissimilarity between samples belonging to different classes.
Prototypes for Class Representation: One of the key concepts in Prototypical Networks is the use of prototypes for representing classes. Each class prototype is computed by taking the mean embeddings of support samples belonging to the same class. This approach provides a compact and informative representation for each class.
Flexibility and Adaptability: Prototypical Networks are flexible and adaptable to various tasks and datasets. They can be applied to different domains, such as image classification and natural language processing. They can easily deal with different few-shot learning scenarios, including one-shot, five-shot, or even zero-shot learning.

Effective Generalisation: Despite being trained on a limited number of examples per class, Prototypical Networks show strong generalisation capabilities. They can accurately classify new samples, even those from unseen classes, based on the learned metric space and class prototypes.
Robustness to Data Imbalance: Prototypical Networks are robust to data imbalance issues commonly encountered in real-world datasets. Since they focus on learning discriminative embeddings and computing class prototypes, they are less affected by class imbalance compared to traditional classifiers.
Interpretability: Using prototypes in Prototypical Networks provides some degree of interpretability with the model's decisions. It is, for example, possible to look at the distances from query samples to class prototypes and understand why certain classifications are made, making the model interpretable and trustworthy.

Overall, prototypical networks are a powerful and flexible framework for few-shot learning tasks and provide an effective solution to challenges associated with limited labelled data and data imbalance.

## 3.4 Network Architecture

The ProtoNet architecture [Figure 3.2] is a sophisticated framework designed for effective few-shot learning tasks, where representations can be learned from a limited amount of labeled data. The architecture consists of an Encoder and a Fully Connected Layer .

The Encoder extracts features from input images through two convolutional blocks. Each block consists of a convolutional layer followed by a ReLU activation function and a max-pooling operation. The Figure 3.2 represents the first convolutional layer processes grayscale images into 32 feature maps, while the second convolutional layer further refines these maps to 64 channels. Dropout layers with a dropout rate of 0.5 are inserted after each max-pooling operation to enhance regularization.

After the convolutional layers, the feature maps are flattened and fed into a fully connected layer. This layer's size is determined by the number of output channels from the convolutional layers and the spatial dimensions of the feature maps. In this architecture, the flattened feature maps yield

1600 input features for the fully connected layer, which uses a linear activation function (no activation).

Improvements Made in Training: The architecture employs learning rate scheduling techniques, which are essential in tuning the learning rate depending on the validation performance, through techniques such as ReduceLROnPlateau. Early stopping is done based on validation loss to prevent overfitting. Thirdly, L2 regularization or weight decay with a value of 1e-5 is added to the optimizer to further prevent overfitting.

In summary, the ProtoNet architecture aims to provide discriminative features from input images while being able to generalize effectively to unseen classes in few-shot learning.

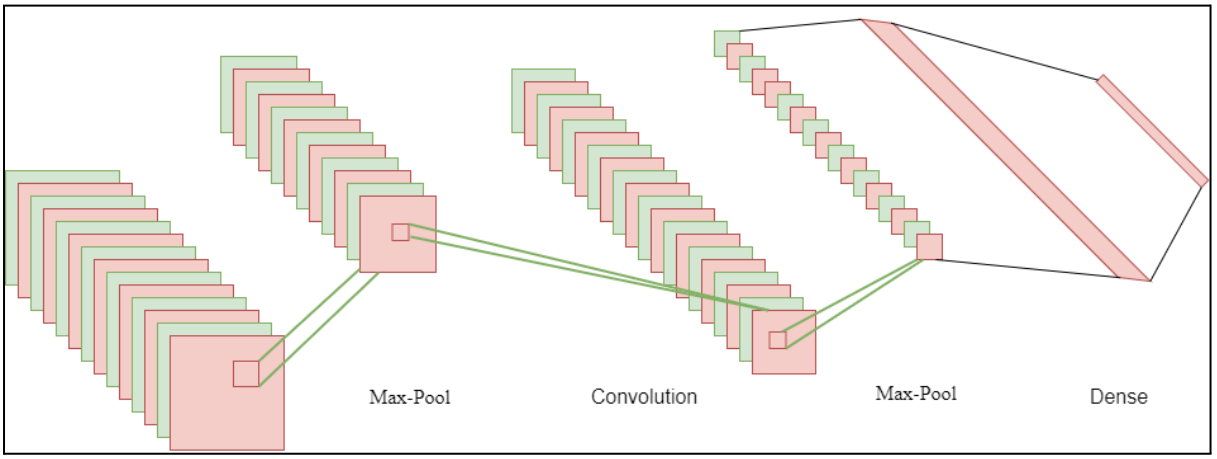


Figure 3.2 Protonet architecture describing max-pooling and convolutional layer

**Chapter 4**

# Experimental Result and Discussion

## 4.1 Datasets

Font recognition is a classic pattern recognition problem for which researchers have worked since the early days of computer vision. Font Recognition is a fundamental issue in document analysis and recognition. Presence of numerous possible fonts, having very subtle and character dependent differences makes it often a difficult and time-consuming task to accurately determine. Font recognition plays an important role in the OCR system also. Font recognition is very important for designers. With today's omnipresence of cameras, the applications of automatic character recognition are broader than ever. This is largely considered a solved problem in constrained situations, such as images of scanned documents containing common character fonts and uniform background. However, images obtained with popular cameras and hand held devices still pose a formidable challenge for character recognition. The challenging aspects of this problem are evident in this dataset.

**Font Recognition (Synthetic) Data**:- We are using a Synthetic Dataset [Figure 4.1] for our given project. This Dataset has 48 different fonts representing 48 different classes. It consists of different words of English language These classes are used to find episodic training mechanisms where at a time 10 classes are used to train the model and then 10 novel classes each with 10 samples are used to test the model and validate it making it a 10 way 10 shot model.


Figure 4.1 Font-Recognition Dataset

**Adobe VFR Datasets**: AdobeVFR dataset [Figure 4.2] contains real-world images of words having different fonts along with annotations. In total, it contains 612 types of fonts available, and each font class has one or more word images. We discarded font classes for which only one image was present because, to evaluate the performance of the proposed Font-Recognition using a prototypical network, we require at least one word image of a given font in the train set at the meta-test stage to perform 1-shot and 5-shot text image classification for the rest of the images of the same font in the test set for the meta-test stage. For the meta-train stage, we used a synthetic dataset of 10 fonts available in the Font-recognition dataset to train the prototypical network.


Figure 4.2 AdobeVFR Dataset

## 4.2 Training Details

**Prototypical Network:** We used 10 and 100 font classes from Font-Recognition dataset for generating training episodes for n-way classification while we performed evaluation experiments for Font-Recognition and AdobeVFR test sets, respectively. The prototypical networks were trained using train shots as 1, 5, 10 for n-way classification tasks, i.e., we sampled 1, 5, 10 images per class per episode from the n classes for training Font-Recognition on these images. We have performed 5-way and 10-way classification and trained the network using the Adam optimizer with a learning rate of 0.001 through stochastic gradient descent. The input to the network is the text image of size 64 × 64. During the meta-test stage, we generated the episodes similar to the training, such as 1-shot, 5-shot, and 10-shot learning for 5-way and 10-way classification. In the test set, we have a meta-test set having 100 fonts for both synthetic font — recognition and AdobeVFR datasets. In the case of the Font recognition dataset, during the testing stage, we sampled n classes randomly from the meta-test set to form an n-way classification task and 1, 5, and 10 samples are selected from each font class for creating a support set corresponding to 1-shot, 5-shot, and 10-shot learning, and the remaining samples in the meta-test set are used as a query set. For the test-set of Font-recognition, we also created a set of images having single words belonging to the 100 font classes of the meta-test set. For evaluation on the AdobeVFR dataset, we chose only those font

classes where we have more than one word-image. This was done so as to take the first word for creating a support-set of meta-test stage for a given font class and the remaining word images of that particular font class were chosen as a query-set. We report the average accuracy on font identification both at the character level and at the word level.

| Dataset | n-way | K-shot | Font accuracy |
|---|---|---|---|
| **Synthetic Font-Recognition Dataset** | 5-way | 1-Shot | 87.1 |
| | | 5-Shot | 95.27 |
| | | 10-Shot | **97.1** |
| | 10-way | 1-Shot | 84.7 |
| | | 5-Shot | 95.41 |
| | | 10-Shot | **97** |
| **AdobeVFR Dataset** | 5-way | 1- Shot | 61.3 |
| | | 5- Shot | **71.7** |

Table 1. Performance comparison (average accuracy in %) of Font-Recognition for Font Identification on the Synthetic Font-Recognition and AdobeVFR datasets.

## 4.3 Results

We compute text font identification accuracy. Text font accuracy is computed by finding font predictions of a given text image using the Font-Recognition for the text style present in the image and then taking the majority vote of the font style predictions to assign the font class to the text. We observe an average accuracy in 5-way font recognition for 1-shot, 5-shot and 10-shot are 87.1, 95.27, and 97.1 respectively. From Table 1 We can also obtain the average accuracy of 10-way font recognition for 1-shot, 5-shot and 10-shot are 84.7%, 95.41% and 97% respectively. Prototypical network performs remarkably well on novel / unseen fonts with very few labeled samples. Next, we present results of 5-way 1-shot and 5-shot font identification on AdobeVFR test-set and obtain an average accuracy of 61.3% and 71.7% at font for 1-shot and 5-shot font identification, respectively on AdobeVFR dataset. We could not perform a 10-shot experiment on the AdobeVFR test-set due to the non-availability of enough examples for creating supportset.

Some resources have been used To perform the experiments and achieve the result. The resources are categorized into two parts such as *Software* and *Hardware*

## Resources (Software / Hardware) used

For conducting the project on font recognition using few-shot learning techniques on Google Colab, the following software and hardware resources were utilized:

**Software**:

Python is a versatile, massively sourced, general-purpose programming language, and it comes with deep learning framework support, hence perfect for machine learning. The model of the Prototypical Network was implemented using PyTorch, which is a great open-source implementation of a deep learning framework, featuring efficient tensor computation with GPU

acceleration and high-level interfaces to the development of neural networks. The Jupyter Notebook was integrated within Google Colab, such that interactive coding and experimentation through combining code, visualizations, and explanatory text all in one document allow for collaboration and documentation of a project. Matplotlib, being a Python plotting library, is used to plot such metrics as the loss curves and accuracy plots during training and evaluation directly in the Jupyter Notebook environment with its versatile plot functions.

**Hardware**:

Google Colab provides access to resources that are GPU-based, like NVIDIA Tesla K80, T4, P4, or P100 GPUs, which would otherwise accelerate deep learning computations and save experimentation time. CPU resources with several cores handle general-purpose computing tasks and project execution in addition to the GPU. RAM, or Random Access Memory, ensures that code runs smoothly and intermediate results are stored without hitting memory-related problems. This, together with Python, PyTorch, Jupyter Notebook, and Matplotlib, supplies an environment in which font recognition models using few-shot learning techniques can be efficiently developed and trained.

**Chapter 5**

# Conclusion and Future Work

We proposed to use few-shot based Prototypical networks for identifying fonts from scanned document images when sufficient amount of annotated data is not available. This was a novel approach as we used Few Shot Learning instead of the traditional Deep Learning methods we used prototypical networks. We named the network Font-ProtoNet and demonstrated that the network is able to learn embeddings which can quickly adapt to newer font classes easily with very few data samples. We also created a synthetic dataset of text-images having varying fonts called FewShot-FontID for font identification purposes and benchmarked the dataset using our proposed network. We would also release this dataset for evaluation by the research community.

In future, we would like to extend the use of prototypical networks to identify different language texts present in a scanned document having multilingual texts which would be very helpful in improving multilingual OCR. We can begin with the basic omniglot dataset to benchmark the model and then go up to different fonts of different languages. We could also use it to make an open source answer script checking model that could identify key

# Reference


[1] Wang, Chi, et al. "Cf-font: Content fusion for few-shot font generation." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.

[2] Wang, C., Zhou, M., Ge, T., Jiang, Y., Bao, H., & Xu, W. (2023). Cf-font: Content fusion for few-shot font generation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (pp. 1858-1867).

[3] Tang, Licheng, Yiyang Cai, Jiaming Liu, Zhibin Hong, Mingming Gong, Minhu Fan, Junyu Han, Jingtuo Liu, Errui Ding, and Jingdong Wang. "Few-shot font generation by learning fine-grained local styles." In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pp. 7895-7904. 2022.

[4] Xi Y, Yan G, Hua J, Zhong Z. Jointfontgan: Joint geometry-content gan for font generation via few-shot learning. InProceedings of the 28th ACM International Conference on Multimedia 2020 Oct 12 (pp. 4309-4317).

[5] Park, Song, Sanghyuk Chun, Junbum Cha, Bado Lee, and Hyunjung Shim. "Few-shot font generation with localized style representations and factorization." In *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 3, pp. 2393-2402. 2021.

[6] Tang, L., Cai, Y., Liu, J., Hong, Z., Gong, M., Fan, M., Han, J., Liu, J., Ding, E. and Wang, J., 2022. Few-shot font generation by learning fine-grained local styles. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 7895-7904).

[7] Xi Y, Yan G, Hua J, Zhong Z. Jointfontgan: Joint geometry-content gan for font generation via few-shot learning. InProceedings of the 28th ACM International Conference on Multimedia 2020 Oct 12 (pp. 4309-4317).

[8] Yaoxiong Huang, Mengchao He, Lianwen Jin, and Yongpan Wang. Rd-gan: few/zero-shot

chinese character style transfer via radical decomposition and rendering. In European Conference on Computer Vision, pages 156–172. Springer, 2020.

[9] Ming-Yu Liu, Xun Huang, Arun Mallya, Tero Karras, Timo Aila, Jaakko Lehtinen, and Jan Kautz. Few-shot unsupervised image-to-image translation. In Proceedings of the IEEE/CVF Interrnational Conference on Computer Vision, pages 10551–10560, 2019.

[10] Pengyuan Lyu, Xiang Bai, Cong Yao, Zhen Zhu, Tengteng Huang, and Wenyu Liu. Auto-encoder guided gan for chinese calligraphy synthesis. In 2017 14th IAPR International Conference on Document Analysis and Recognition (ICDAR), volume 1, pages 1095–1100. IEEE, 2017.

[11] Yunjey Choi, Minje Choi, Munyoung Kim, Jung-Woo Ha, Sunghun Kim, and Jaegul Choo. Stargan: Unified generative adversarial networks for multi-domain image-to-image translation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8789–8797, 2018.

[12] Samaneh Azadi, Matthew Fisher, Vladimir G Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. Multi-content gan for few-shot font style transfer. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 7564–7573, 2018.

[13] Chuan Wen, Yujie Pan, Jie Chang, Ya Zhang, Siheng Chen, Yanfeng Wang, Mei Han, and Qi Tian. Handwritten chinese font generation with collaborative stroke refinement. In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision, pages 3882–3891, 2021.

[14] Yexun Zhang, Ya Zhang, and Wenbin Cai. Separating style and content for generalized style transfer. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 8447–8455, 2018.

[15] Jun-Yan Zhu, Taesung Park, Phillip Isola, and Alexei A Efros. Unpaired image-to-image translation using cycleconsistent adversarial networks. In Proceedings of the IEEE international conference on computer vision, pages 2223– 2232, 2017.

[16] Pengyuan Lyu, Xiang Bai, Cong Yao, Zhen Zhu, Tengteng Huang, and Wenyu Liu. 2017. Auto-encoder guided gan for Chinese calligraphy synthesis. In Proceedings of the International Conference on Document Analysis and Recognition, Vol. 1. 1095–1100.

[17] Sandun LK. 2018. SandunLK 10K fonts pack. https://sandunlk.home.blog/.

[18]Samaneh Azadi, Matthew Fisher, Vladimir Kim, Zhaowen Wang, Eli Shechtman, and Trevor Darrell. 2018. Multi-content GAN for few-shot font style transfer. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 7564–7573.

[19]Justin Johnson, Alexandre Alahi, and Li Fei-Fei. 2016. Perceptual losses for realtime style transfer and super-resolution. In Proceedings of the European Conference on Computer Vision. 694–711.

[20]Alfred Zong and Yuke Zhu. 2014. StrokeBank: Automating personalized Chinese handwriting generation. In Proceedings of the Innovative Applications of Artificial Intelligence Conference.