"Offline Handwritten Kannada Character Recognition Using Manifold Smoothing and Label Propagation"

A Dissertation Report

Submitted in the partial fulfilment for the award of degree of
Bachelor of Engineering in Computer Science and Engineering

Submitted by

A NAVEEN
16GAEC9001

ANURAG PANDEY
16GAEC9009

RUTURAJ RAMCHANDRA SHITOLE

16GAEM9068

VIKAS SAINI

16GAEC9066

Under the guidance of

Dr. CHAMPA H N

Professor,

Department of CSE, UVCE,

Bangalore

Department of Computer Science and Engineering

University Visvesvaraya College of Engineering

BANGALORE UNIVERSITY

2019-2020

“Offline Handwritten Kannada Character Recognition Using Manifold Smoothing and Label Propagation”

A

Dissertation Report

Submitted in the partial fulfilment for the award of degree of
Bachelor of Engineering in Computer Science and Engineering

Submitted by

A NAVEEN	**16GAEC9001**
ANURAG PANDEY	**16GAEC9009**
RUTURAJ RAMCHANDRA SHITOLE	**16GAEM9068**
VIKAS SAINI	**16GAEC9066**

Under the guidance of

Dr. CHAMPA H N

Professor,

Department of CSE, UVCE,

Bangalore

Department of Computer Science and Engineering

University Visvesvaraya College of Engineering

BANGALORE UNIVERSITY

2019-2020

Bangalore University

University Visvesvaraya College of Engineering

Department of Computer Science and Engineering

K.R. Circle, Bangalore-560001

CERTIFICATE

This is to certify that **A NAVEEN, ANURAG PANDEY, RUTURAJ RAMCHANDRA SHITOLE, VIKAS SAINI** bearing USN **16GAEC9001, 16GAEC9009, 16GAEM9068, 16GAEC9066** have satisfactorily completed the project entitled **"Offline Handwritten Kannada Character Recognition Using Manifold Smoothing and Label Propagation"**, in partial fulfilment for the requirement of completion of Eight Semester of Bachelor of Engineering in Computer Science and Engineering, University Visvesvaraya College of Engineering affiliated to Bangalore University, during the academic year 2019-2020.

Dr. CHAMPA H N

Guide and Professor,
Department of CSE,
UVCE

Dr. S M DILIP KUMAR

Chairperson,
Department of CSE,
UVCE

Examiner 1:

Date:

Examiner 2:

Date:

ACKNOWLEDGEMENT

We take this opportunity to thank our institution **UNIVERSITY VISVESVARAYA COLLEGE OF ENGINEERING** for having given us an opportunity to carry out this project.

We would like to thank **Professor H N Ramesh, Principal, UVCE,** for providing us all the facilities to work on this project. We are indebted to him for being our pillar of strength and inspiration.

We wish to place our grateful thanks to **Dr. S M Dilip Kumar, Chairperson, Department of Computer Science and Engineering, UVCE,** who helped us to make our project a great success.

We are grateful to acknowledge **Dr. Champa H N, Professor, UVCE,** for her valuable suggestions and relentless support, which has sustained us throughout the course of the project.

We extend our gratitude to **Mr. Ramesh G, Research Scholar, UVCE,** for his constant support and feedback, which has enabled us to complete the project.

We express our sincere thanks to all the teaching staff of Computer Science and Engineering Department, for their encouragement and inspiring guidance and for all the facilities that they provided us to successfully complete this project.

We also thank our parents for their help, encouragements and financial support.

A Naveen
(16GAEC9001)

Anurag Pandey
(16GAEC9009)

Ruturaj Ramchandra Shitole
(16GAEM9068)

Vikas Saini
(16GAEC9066)

ABSTRACT

Supervised learning techniques using deep learning models are highly effective in their application to handwritten character recognition. However, they require a large dataset of labelled samples to achieve good accuracies. For applications where classes of characters are difficult to train for, due to their rarity in occurrence, techniques from semi supervised learning and self-supervised learning can be used to improve the accuracy in classifying these novel classes. In this paper, we propose a framework that incorporates techniques from semi-supervised learning and analyze the effectiveness of using a combination of regularization to features extracted from a convolutional neural network backbone, augmentation to improve the trained features and label propagation to classify previously unseen characters. We use the dataset used in Karthik et al., 2018 and the episodic learning framework introduced by Vinyals et al., 2016 to validate our approach. We show through experimentation that the accuracy obtained make this framework competitive with the currently available supervised learning counterparts, despite the large reduction in the number of labelled samples available for the novel classes.

CONTENTS

1. **Introduction** **1-8**

1.1 Computer Vision 2

1.1.1 Gathering of input images 3

1.1.2 Input preprocessing 3

1.1.3 Feature Extraction 4

1.2 Handwritten character recognition 5

1.3 Semi-Supervised learning 6

1.4 Kannada Language 7

2. **Literature Survey** **9-12**

3. **Motivation** **13-13**

4. **Problem Definition** **14-14**

4.1 Problem statement 14

4.2 Objective 14

5. **System Architecture** **15-16**

5.1 Experimental Steps 15

5.2 Episodic Framework 16

6. **Hardware and Software requirements** **17-18**

6.1 Hardware requirements 17

6.2 Software requirements and libraries 17

7. **Proposed method** **19-26**

7.1 Manifold smoothing with metric learning 19

7.2 Label Propagation 20

7.3 Feature extraction using Convolutional Neural Networks 21

7.4 Pretraining process 23

7.5 Finetuning process 25

8. **Implementation** **27-37**

8.1 Simulation dataset 28

8.2 Conv4 Backbone 29

8.3 ResNet-12 Backbone 30

8.4 Episode Generator: 32
8.6 Manifold Smoothening and Label Propagation 35
9. **Result** **38-46**
9.1 Performance evaluation 38
9.2 Conv4 network 39
9.3 ResNet-12 network 41
9.4 Comparison between the networks 45
9.5 Comparison with previous works 46
10. **Conclusion** **47-47**
References **48-51**

LIST OF FIGURES

Fig 1.1: Steps in Computer Vision 5
Fig 1.2: Handwritten character recognition 6
Fig 1.3: Kannada Script 8
Fig 5.1: Flow diagram of episodic framework 16
Fig 7.1: Gaussian Distribution about two centers 20
Fig 7.2: Graph Clustering from power iteration 20
Fig 7.3: Diagram of the Conv4 Model 21
Fig 7.4: Diagram of the Resnet-12 Model 22
Fig 7.5: Flow Diagram of the pretraining process 24
Fig 7.6: Flow Diagram of the finetuning process 26
Fig 8.1: Samples from C_{train} 28
Fig 8.2: Samples from C_{test} 28
Fig 8.3: Samples from C_{val} 29
Fig 9.1: Pretraining accuracy vs Number of epochs 39
Fig 9.2: Pretraining loss vs Number of epochs 39
Fig 9.3: 1-Shot finetuning accuracy vs Number of epochs 40

Fig 9.4: 1-Shot finetuning loss vs Number of epochs 40
Fig 9.5: 5-Shot finetuning accuracy vs Number of epochs 41
Fig 9.6: 5-Shot finetuning loss vs Number of epochs 41
Fig 9.7: Pretraining accuracy vs Number of epochs 42
Fig 9.8: Pretraining loss vs Number of epochs 42
Fig 9.9: 1-Shot finetuning accuracy vs Number of epochs 43
Fig 9.10: 1-Shot finetuning loss vs Number of epochs 43
Fig 9.11: 5-Shot finetuning accuracy vs Number of epochs 44
Fig 9.12: 5-Shot finetuning loss vs Number of epochs 44

CHAPTER 1

INTRODUCTION

The challenge of converting manuscripts and printed documents into digital formats has been a cornerstone of Computer Vision research. Recent advances have blurred the interface between physical copies of text and their digital counterparts. Large scale scanning of thousands of historical documents has been performed. Enabling visually impaired individuals to read signboards and paper, faster processing of cheques. Legislative bodies have benefitted from the ease of digitizing legal documents, allowing for seamless transfer, signing, and searching. The field of handwritten character analysis has strived to make effective algorithms to perform various goals, such as the classification of handwritten characters, classification of the writers of different manuscripts, generating text matching the handwriting of a writer and so on. Prior to supervised deep neural networks, handcrafted methods were used for handwritten character recognition which often required several different steps such as binarization of images, rescaling and rotating the images, performing statistical aggregations on different parts of the images etc. This required finetuning of a large number of parameters to obtain accurate results and could not generalize well to variations in the input images.

Supervised learning using deep neural networks has allowed most of the explicit tasks to be replaced by a single neural network model that by virtue of backpropagation is able to learn the weights required for effective extraction of features from the images that are used for classification. By providing a large training set which includes diverse samples of each character, the neural network is rendered more robust in its accuracy in classifying a larger range of handwriting samples. However, the creation of a large labelled training set of images is laborious and certain character classes have few real-world samples. By utilizing already pretrained models to predict the new classes, sample efficiency is improved. The difficulty in obtaining such a dataset for Kannada handwritten characters is compounded by the large number of possible graphemes in the Kannada script, stemming from the use of combination of base characters to form digraphs. Semi-supervised learning techniques which exploit the use of a large unlabeled dataset to improve the robustness and accuracy of a model trained on a small labelled training set, have been successfully used to achieve this goal. We model the scenario of incorporation of novel classes with the episodic

learning approach [1]. Recent works in few shot learning utilize this framework to emulate meta-learning tasks [2]. Improved generalization of the neural network is achieved through the use of data augmentation where sample images are rotated in 4 different orientations, increasing the number of training samples the network sees [3]. The use of label propagation allows the incorporation of new classes into the classification framework with very few extra training samples [4].

This project aims to incorporate these techniques to create an Offline Kannada handwritten character classifier that can be trained to retain high accuracies on classes with as few as 1 or 5 samples. This allows for rapid incorporation of classes with minimal extra samples required.

1.1 Computer vision:

Recent work has seen the resurgence of feature-based methods, used in conjunction with machine learning techniques and complex optimization frameworks. The advancement of Deep Learning techniques has brought further life to the field of computer vision. The accuracy of deep learning algorithms on several benchmark computer vision datasets for tasks ranging from classification, segmentation and optical flow has surpassed prior methods. Computer Vision extracts information from images and recognizes specific concepts. It can therefore perform a variety of tasks such as recognizing faces or characters in an image, detecting the location of an object in an image, or classifying images.

The most common CV tasks are object detection and image classification. Object detection consists in searching for a particular element and locating it within an image, using a “box”. There is also a more elaborate and accurate detection method (to the pixel) called polygon segmentation. As for image classification, it makes it possible to identify to which category an image belongs, based on its composition, i.e. to identify the main subject of the image. However, it is possible to associate more than one category to an image thanks to tagging with an operation similar to classification. Traditional computer vision steps are shown in Fig 1.1.

1.1.1 Gathering of input images:

The configuration of the input acquisition device to robustly capture the required input with a large variety of perturbations is necessary. The format the images are stored in, color channel representation, size of Individual input samples are important factors.

1.1.2 Input preprocessing:

In order to improve the representation of the image for a particular task, the preprocessing step is performed:

- **Point Operators:**

  All the points in an image are used to create a transformed version of the original image (in order to make explicit the content inside an image, without changing its content). Some examples of Point Operators are: Intensity Normalization, Histogram Equalization and Thresholding. Point Operators are commonly used in order to help visualize better an image for human vision but don't necessarily provide any advantage for a Computer Vision system.

- **Group Operators:**

  In this case, a group of points from the original image are taken in order to create a single point into the transformed version of the image. This type of operation is typically done by using Convolution. Different types of kernels can be used to be convolved with the image in order to obtain the transformed result. Some examples are: Direct Averaging, Gaussian Averaging and the Median Filter. Applying a convolution operation to an image can, as a result, decrease the amount of noise in the image and improve smoothing (although this can also end up slightly blurring the image). Since a group of points are used in order to create a single new point in the new image, the dimensions of the new image will necessarily be lower than the original one. One solution to this problem is to apply either zero paddings (setting the pixel values to zero) or by using a smaller template at the border of the image. One of the main limitations of using convolution is its execution speed when working with large template sizes, one possible solution to this problem is to use a Fourier Transform instead.

1.1.3 Feature Extraction

Once pre-processed an image, there are 4 main types of Feature Morphologies which can be extracted from an image by using a Feature Extractor:

- **Global Features:**

  The whole image is analyzed as one and a single feature vector comes out of the feature extractor. A simple example of a global feature can be a histogram of binned pixel values.

- **Grid or Block-Based Features:**

  The image is split into different blocks and features are extracted from each of the different blocks. One of the main techniques used in order to extract features from blocks of an image is Dense SIFT (Scale Invariant Feature Transform). This type of Features is used prevalently to train Machine Learning models.

- **Region-Based Features:**

  The image is segmented into different regions using techniques such as thresholding or K-Means Clustering and then connect them into segments using Connected Components, and a feature is extracted from each of these regions. Features can be extracted by using region and boundary description techniques such as Moments and Chain Codes).

- **Local Features:**

  Multiple single interest points are detected in the image and features are extracted by analyzing the pixels neighboring the interest points. Two of the main types of interest points which can be extracted from an image are corners and blobs, these can be extracted by using methods such as the Harris & Stephens Detector and Laplacian of Gaussians. Features can finally be extracted from the detected interest points by using techniques such as SIFT (Scale Invariant Feature Transform). Local Features are typically used in order to match images to build a panorama/3D reconstruction or to retrieve images from a database.

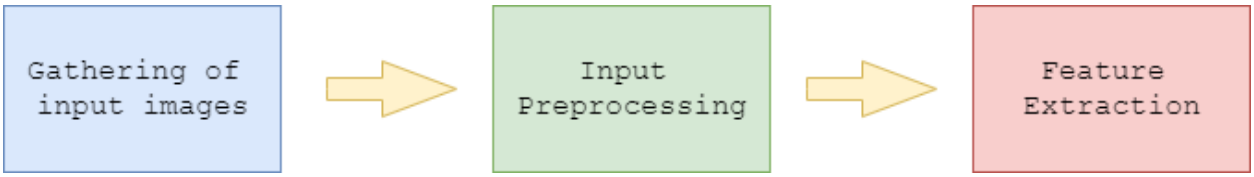

Fig 1.1: Steps in Computer Vision

1.2 Handwritten character recognition:

There has been significant growth in the application of offline handwriting recognition (Fig 1.2) during the past decade. Few of those are mail sorting, bank processing, document reading, postal addresses recognition, handwritten address interpretation and writer identification. Handwritten address interpretation is the task of assigning a mail piece image to a delivery address by determining the country, state, city, post office, street number, the firm or the person's name. Bank processing includes recognition of legal amounts, date and signature. Writer Identification deals with the establishment of authorship of a document for which some prototypes toolsets for document examination. As online recognition refers to methods dealing with the automatic processing of a message as it is written using a digitizer.

Over the years these methods have evolved from academic exercises to developing technology driver applications such as pen based computers, sign verifiers, developmental tools as well as in home safety using handwritten pattern recognition systems. The concept of a pen-based computer was proposed by Kay. Signature verification refers to the comparison of test signature with reference specimens. The most promising application to be emerged will be related to long distance authorization, personalization, tracking of money and documents and much more. Developmental tools include educational software for teaching handwriting to children, LCD with digitizer, digitized tablets.

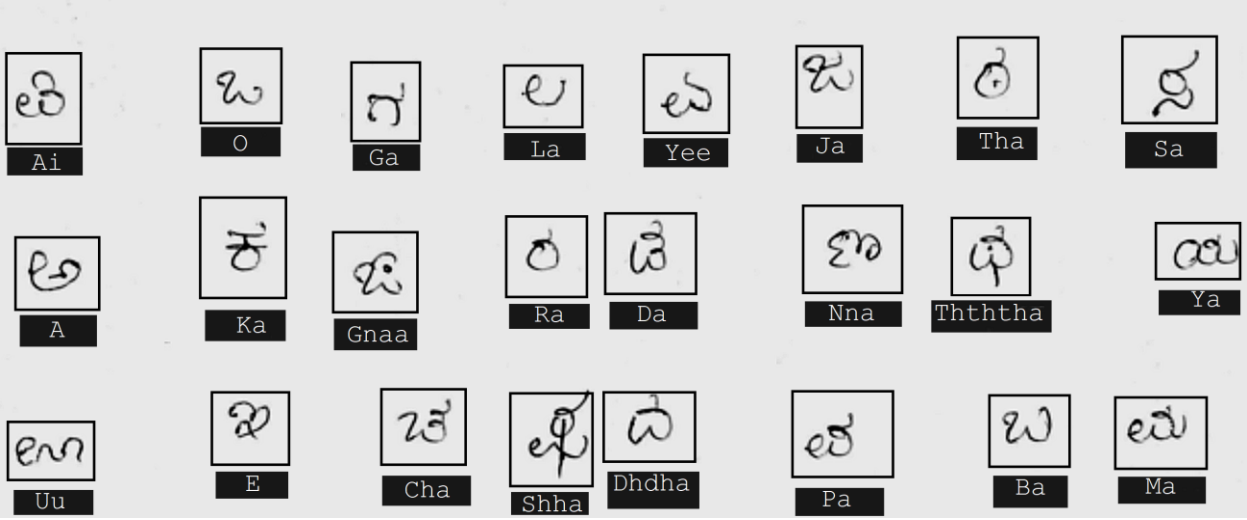

Fig 1.2: Handwritten Character Recognition

1.3 Semi-Supervised learning:

In machine learning, a distinction has traditionally been made between two major tasks: supervised and unsupervised learning. In supervised learning, one is presented with a set of data points consisting of some input x and a corresponding output value y. The goal is, then, to construct a classifier or regressor that can estimate the output value for previously unseen inputs. In unsupervised learning, on the other hand, no specific output value is provided. Instead, one tries to infer some underlying structure from the inputs. For instance, in unsupervised clustering, the goal is to infer a mapping from the given inputs (e.g. vectors of real numbers) to groups such that similar inputs are mapped to the same group.

Semi-supervised learning is a branch of machine learning that aims to combine these two tasks. Typically, semi-supervised learning algorithms attempt to improve performance in one of these two tasks by utilizing information generally associated with the other. For instance, when tackling a classification problem, additional data points for which the label is unknown might be used to aid in the classification process. For clustering methods, on the other hand, the learning procedure might benefit from the knowledge that certain data points belong to the same class.

As is the case for machine learning in general, a large majority of the research on semi-supervised learning is focused on classification. Semi-supervised classification methods are particularly relevant to scenarios where labelled data is scarce. In those cases, it may be difficult

to construct a reliable supervised classifier. This situation occurs in application domains where labelled data is expensive or difficult to obtain, like computer-aided diagnosis, drug discovery and part-of-speech tagging. If sufficient unlabeled data is available and under certain assumptions about the distribution of the data, the unlabeled data can help in the construction of a better classifier. In practice, semi-supervised learning methods have also been applied to scenarios where no significant lack of labelled data exists: if the unlabeled data points provide additional information that is relevant for prediction, they can potentially be used to achieve improved classification performance.

1.4 Kannada language

The Kannada script (Fig 1.3) is an abugida or segmental writing system. Consonant-Vowel sequences are combined as a unit and written where each unit is based on a consonant and there is a secondary vowel notation. The Kannada script consists of 47 characters (with the exception of archaic characters za, la and ru) with 13 vowels called “Swaras” and 34 consonants called “Vyanjanas”.

The characters in each word are written as segmented parts (unlike Devanagari). Many characters differ only by strokes written above or below the base character glyph. Complex characters are formed through the combination of base characters in the Varnamale such as the Kagunitha and Otthakshara which form vowel-consonant sequences and stressed consonant sequences respectively.

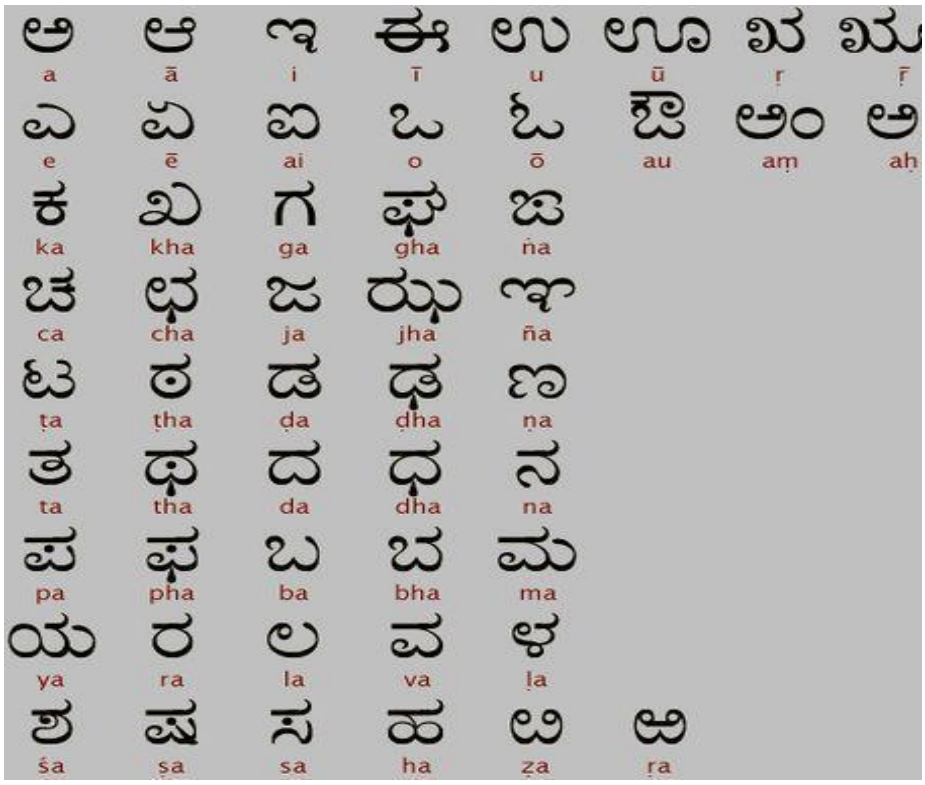

Fig 1.3: Kannada Script

The project is focused on the identification of these characters from various sources of literature where only a few characters are well represented and there is a need to incorporate new characters with limited labelled samples.

We describe the system architecture in Chapter 5 where we describe the episodic framework. We discuss the proposed method in Chapter 7, where Sections 7.1 and 7.2 discuss the theoretical techniques being used in the framework, sections 7.3 and 7.4 discuss the feature extractor architecture, section 7.5 and 7.6 discuss the different steps involved in the proposed method. The implementation is discussed in Chapter 7 where Section 7.1 details the dataset used. We describe the results of experimentation in Chapter 9 where Sections 9.1 to 9.4 show the results of the experiment and compare the different backbone feature extractors. We compare the proposed method to existing work in Section 9.5.

CHAPTER 2

LITERATURE SURVEY

Handwriting recognition is a part of computer vision and revolves around identification and classification of handwritten characters or texts of various different languages. The source of input is from various documents, photographs and other surface devices which enable characters to be written on them. Effort towards advances in captured scene text detection and recognition [5] have been made by Zhu *et al.*, to identify state-of-the-art algorithms and predicted possible research directions.

Major challenges in the process of recognition of characters of Indian languages range from the variety in the set of characters to the similarity between the characters. The solutions proposed are custom designed and no definite generic solution can be provided. Also, the origin of Indian language script being the Brahmi script, the similarity between the characters of two different languages is a challenge open to be solved. Further in Kannada language script, apart from vowels and consonants, the compound characters are represented as a combination of one or more consonants or vowels, with a symbolic representation of the vowel/consonant like a sub/superscript to the character. These problems are confined to the classification of Indian languages and scripts, unlike English. Also, the different styles of writing and varying sizes cause the results to be skewed in some methods of classification.

- Pertaining to the handwritten character recognition of Kannada Kagunita characters, Kagunita characters' moments feature extraction from Gabor wavelets has been emphasized by Ragha *et al*., [6].
- Work towards the provision of benchmark Kannada handwritten document dataset and its segmentation is done by Alaei *et al.*, [7] [8] to solve the unavailability of a standard data set and provide an insight by ground truths for researchers working on the Kannada language.
- Obaidullah *et al.*, [9] have also contributed to the dataset of Kannada handwritten scripts recently, with a page-level document image dataset of 11 Indic scripts, of which Kannada is also part of.

- The complexity and versatility of Kannada script, it being a Dravidian language is quoted by Ramakrishna *et al.*, [10] in their attempt to solve the challenges in the segmentation of on-line handwritten, isolated Kannada words.
- An attempt to classify totally unconstrained handwritten Kannada characters using Fourier transform based Principle Component Analysis and Probabilistic Neural Networks is made obtaining an accuracy of 88.64% by Manjunath *et al*., [11] while Niranjan *et al.*, [12] had approached the same problem of classification of unconstrained handwritten Kannada characters using FLD.
- A newer approach to the same problem has been the usage of ridge-let transforms, by representing mono-dimensional singularities in bidimensional space, is used by Naveena *et al.*, [13]. Contributions towards the sub-field of on-line handwritten Kannada character recognition is also seen.
- A method to choose classifiers for Hierarchical Recognition of Kannada characters in online mode is proposed by Murthy *et al.*, [14]. Further, a combination of online and off-line complementary systems is also introduced to recognize handwritten Kannada characters yielding an increase of online recognizer accuracy by 11%.
- Application of Kannada handwriting recognition techniques has been used to identify writer's text. Dhandra *et al.*, [15] have worked towards writer identification by using texture analysis and a solution towards this problem rectification is done using a feature vector consisting of directional multiresolution spatial features based on Radon transforms, Discrete Cosine transforms and structural features such as aspect ratio and on-pixel ratios.
- Some of the recent advancements in the field have concentrated towards better results by usage of various techniques such as SVMs and Neural Networks. Ramya *et al.*, [16] has used SVM to classify the images of online Kannada handwriting based on various features such as x, y coordinates, pressure and strokes and has achieved an average of accuracy 94.35%.
- Other techniques to dynamically recognize online handwriting for multiple languages was made by Keysers *et al.*, [17] using techniques of generation and scoring of character hypothesis and best path searching. The authors also suggest language-specific adaptations

to international languages such as Latin, Chinese, Japanese and Indian languages such as Gurumukhi, Kannada, Telugu, Malayalam and others.

- Word retrieval using Gabor Wavelets from Kannada documents has been worked upon by Hangarge *et al.,* [15] using the directional energy feature of the word images.
- The preprocessing method of the data set, as an important factor in the process of classifying the Kananda characters, is studied by using signal processing and statistical techniques by Ramya *et al.,* [18]. This clearly shows how the performance of the classifiers such as SVM are affected by the preprocessing and testing methods.
- In off-line Kannada handwritten character recognition, Hallur *et al.,* [19] have proposed a holistic approach-based system to recognize the off-line handwritten Kannada numerals. This system is tested on a database of 147 x 10 Kannada digits contributed by 147 writers. In spite of improper shape in handwritten numerals our proposed system archives accuracy of 95.98% in efficiently recognizing them.
- In [20], the authors have made efforts towards the restoration of degraded Kannada handwritten paper manuscripts or hastapratis using both special local and global binarization techniques, by elimination of uneven background illumination.
- Karthik *et al.,* [21] present a method based on Histogram of Oriented Gradients for the recognition of handwritten Kannada numeral. HOG descriptors, which are proven to be invariant to geometric transformation, are one among the best feature descriptors for character recognition. They have used multi-class SVMs for the classification. The method is experimented by the authors on 4,000 images of isolated handwritten Kannada numerals and an average accuracy of 95% is obtained.
- Further, the use of HOG with ANN (Artificial Neural Networks) and SVM is done by Yadav *et al.,* [22] in their work towards the field. They describe an approach to performing classification on Kannada characters of handwritten origin or those present in natural images, using the HOG descriptors, for feature extraction from the images of the handwritten Kannada characters, and employing a machine learning model (neural networks or support vector machines) for final classification. This effort shows the comparison of classification accuracies between the two classifiers.

- Mirabdollah *et al.,* [23] have proposed the alternative feature vector of DAG, wherein the image is divided into blocks and windows of standard sizes, and the average of the values of pixels in each window is appended to the descriptor to obtain the feature vector.
- The authors Karthik *et al.,* in [24] have grouped the vowels and consonants separately and used 400 images per character to train the deep belief network. They have claimed an average accuracy of 97.04%.
- Even though some of these efforts have yielded considerable accuracies, the introduction of state-of-the-art, robust methods to the field is a clear requirement, in the view of the challenge's issues with these existing methods.
- Ramesh *et al.,* [25] show the use of Convolutional Networks in creating highly accurate handwritten character classifiers. They grouped the vowels and consonants separately and used 400 images per character for training the Convolutional Neural Network. They have claimed the accuracy of 98.7%.

CHAPTER 3

MOTIVATION

The great success of supervised neural network-based machine learning techniques can be attributed to the minimal manual tuning of parameters required as well as the flexibility of the models to learn effective feature representations that work for a diverse range of inputs. However, in order to achieve good generalization capabilities and robustness, supervised neural network models require well curated, large, labelled datasets. This allows the models to effectively learn the different possible variations they may encounter and classify each correctly. The bias created due to unbalanced datasets can cause these models to prefer predicting the classes which were represented with a higher frequency in the training set, resulting in poor performance in recognizing previously unseen classes of characters.

CHAPTER 4

PROBLEM DEFINITION

4.1 Problem statement:

This project aims to solve the “Recognition of Handwritten Kannada Characters in a Few-Shot Learning perspective”, by using a robust, state of the art technique, which provides best in the class accuracies and reliable results. The Kannada alphabet has 47 base characters.

4.2 Objective:

The objectives of this project are:

1. To achieve better accuracy in training and validation on seen classes.
2. To achieve better accuracy in finetuning on unseen classes, using few-shot learning.
3. To compare and choose the feature extractor backbone that provides higher accuracies.
4. To reduce the number of epochs required for the model to converge.
5. To compare 1-shot and 5-shot accuracies and determine the increase in accuracy.

CHAPTER 5

SYSTEM ARCHITECTURE

The architecture utilized in the proposed method is based on the episodic framework for few-shot learning laid out in Fig 5.1. The dataset consists of images of handwritten characters in Kannada with 400 examples, written by multiple writers each for 47 classes with a size of 84x84 for each image. The episodic framework was utilized to evaluate the architecture in a few-shot environment.

5.1 Experimental Steps:

The experiment was carried out with the following steps:

- **Collection of the dataset:** The dataset consists of 47 classes representing each base character of the Kannada abugida. Each class consists of 400 samples obtained from different writers. 50% of the dataset was used for pretraining (24 classes), 2% was used for finetuning (12 classes) and 25% was used for the validation set (11 classes).

- **Preprocessing the images:** The images were rescaled to 84×84px using the PIL library. Bilinear Interpolation was used to achieve this. The images were converted to RGB format.

- **Training the handwritten character classifier:** Two different convolutional networks were used, the Conv4 network and Resnet-12 network. The training consisted of the pretraining phase where the network was trained on the base set. The next phase was the finetuning phase where the network was trained in a episodic fashion on the unseen classes.

- **Analyzing the result:** The accuracy and loss of the two different networks was plotted and compared. Training and Validation accuracy was plotted for the pretraining phase (seen characters) while Test and Validation accuracy was plotted for the finetuning phase. 1-shot and 5-shot finetuning was performed (1 example per class and 5 examples per class respectively)

5.2 Episodic Framework:

The episodic framework was introduced by Vinyals *et al.*, 2016 [1]. It provides a simulation for training a meta-learning model for few-shot classification tasks. In the episodic framework as shown in Fig 5.1, we assume a large labelled dataset C_{train} is present. Our objective is to train the classifier on an unseen set of classes C_{test} where only a few labelled samples are available. In each episode, a small subset of N classes are sampled from C_{train} in order to construct a support set S and query set Q. For N way K shot learning, the number of labelled samples available is K, each task has N classes to be classified. The support set S has K examples from each of the N classes while the query set Q has different examples from the same N classes.

In our method we choose $N = 5$ classes and the size of the query set as 15 examples per class. We choose the 5 classes uniformly over the union of sets $C_{train} \cup C_{test}$ and sample accordingly. A transductive setting is used due to the small size of K in the support set. The entire query set Q can be used for predicting labels rather than predicting each example independently. This helps alleviate the bias caused by the small number of samples while improving generalization.

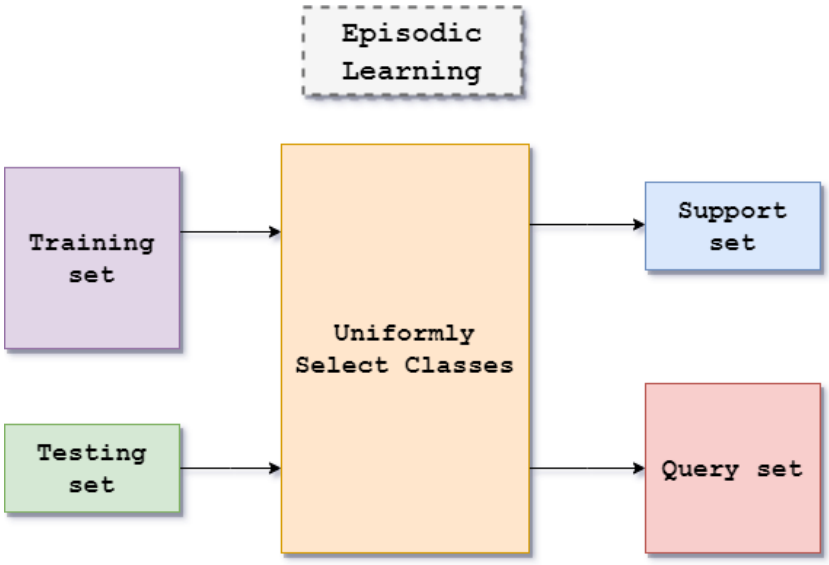

Fig 5.1: Flow diagram of episodic framework

CHAPTER 6

HARDWARE AND SOFTWARE REQUIREMENTS

6.1 Hardware requirements:

An AWS instance possessing 4 NVIDIA K80 GPUs with 8GB RAM each was used.

NVIDIA K80:

The Tesla K80 was a professional graphics card by NVIDIA, launched in November 2014. Built on the 28 nm process, and based on the GK210 graphics processor, in its GK210-885-A1 variant, the card supports DirectX 12. The GK210 graphics processor is a large chip with a die area of 561 mm² and 7,100 million transistors. Tesla K80 combines two graphics processors to increase performance. It features 2496 shading units, 208 texture mapping units, and 48 ROPs, per GPU. NVIDIA has paired 24 GB GDDR5 memory with the Tesla K80, which are connected using a 384-bit memory interface per GPU (each GPU manages 12,288 MB). The GPU is operating at a frequency of 562 MHz, which can be boosted up to 824 MHz, memory is running at 1253 MHz

6.2 Software requirements and libraries:

The programming was done in the Python programming language utilizing the following libraries:

a) **Pytorch:**

PyTorch is a Python machine learning package based on Torch, which is an open-source machine learning package based on the programming language Lua. PyTorch has two main features such as Tensor computation (like NumPy) with strong GPU acceleration and Automatic differentiation for building and training neural networks.

b) **Matplotlib:**

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tkinter, wxPython, Qt, or GTK+

c) Python Image Library:

Python Imaging Library (abbreviated as PIL) (in newer versions known as Pillow) is a free and open-source additional library for the Python programming language that adds support for opening, manipulating, and saving many different image file formats. It offers features such as per-pixel manipulations, masking and transparency handling, image filtering, such as blurring, contouring, smoothing, or edge finding, image enhancing, such as sharpening, adjusting brightness, contrast or color, adding text to images etc.

CHAPTER 7

PROPOSED METHOD

This work uses the combination of Manifold smoothing and Label Propagation to solve the given problem statement. Manifold Smoothing is used to regularize the features extracted for better generalization while Label Propagation allows few-shot inference on unseen classes.

7.1 Manifold smoothing with metric learning

In order to make the decision boundaries of the hidden layer of the model more smooth, resulting in better robustness and generalization, a layer to smoothen the extracted features is used [26]. Given the set of feature vectors $z_i \in R^m$ which are extracted using the Convolutional Neural Network layers, a smoothing function is applied to obtain $\tilde{z}_i$ which are forwarded to the fully connected layer for classification. This smoothing process consists of using a Gaussian similarity function as shown in Fig 7.1 using the L2 norm as the distance function, $d_{ij}^2 = \|z_i - z_j\|_2^2$, for pairs of features z_i, z_j and $\sigma = Var(d_{ij}^2)$ as a measure of the similarity/dissimilarity of the different features.

A similarity matrix is constructed using:

$$A_{ij} = e^{\frac{-d_{ij}^2}{\sigma^2}} \quad (1)$$

The similarity matrix A is normalized using the Laplacian in order to ensure convergence:

$$L = D^{-\frac{1}{2}} A D^{-\frac{1}{2}} , \quad (2)$$

$$\text{where } D_{ii} = \sum_j A_{ij} \quad (3)$$

Power iteration is used to successively increase the weights of the closest features while reducing the weights of the features that are not too close to each other. This is similar to the power iteration needed in label propagation and the propagator matrix P is thus obtained by:

$$P = (I - \alpha L)^{-1} \quad (4)$$

The new feature vectors are calculated as:

$$\tilde{z}_\iota = \sum_j P_{ij} z_j \quad (5)$$

This is similar to a weighted sum of neighbors, resulting in a reduction in the noise present in each feature vector.

7.2 Label Propagation:

The prediction of labels for the query set Q using label propagation is obtained using the similarity matrix that is equivalent to the one used in the manifold smoothing step.

Given the query set Q, the label matrix Y is constructed using:

a) The matrix Y_S of size $(nk \times n)$ corresponding to the support set S.

In each row of Y_S, the column corresponding to the correct label is 1, (Y_{ij} =1) if $y_i = j$. The rest of the elements are 0.

b) The matrix $\boldsymbol{0}$ which is a matrix of 0s of size $(t \times n)$ corresponding to the query set Q

where n is the number of classes, k is the number of samples per class in S , and t is the number of samples in Q.

$$Y = \begin{pmatrix} Y_S \\ 0 \end{pmatrix} \quad (6)$$

Label propagation iteratively determines the unknown labels for the union set $S \cup Q$ [4]:

$$F_{t+1} = \alpha L F_t + (1-\alpha)Y \quad (7)$$

L is the normalized similarity matrix and α is the smoothing factor between 0 and 1. The sequence $\{F_t\}$ converges to

$$F^* = (I - \alpha L)^{-1} Y \quad (8)$$

Where I is the identity matrix. The different features are clustered in a similar fashion to graph spectral clustering as shown in Fig 7.2.

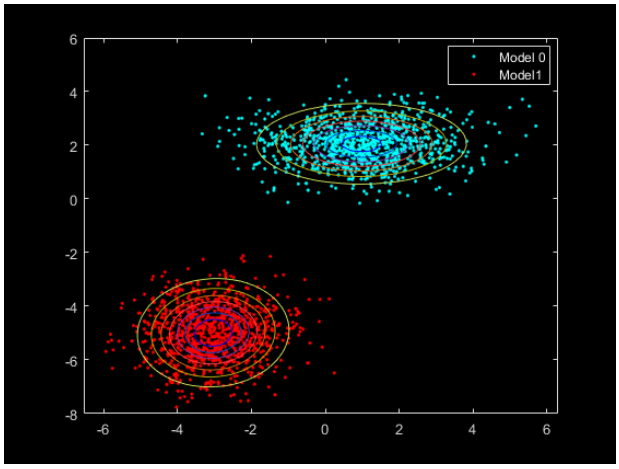

Fig 7.1: Gaussian similarity function

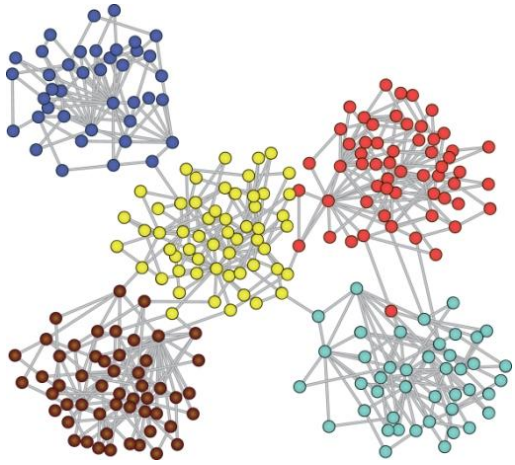

Fig 7.2: Graph clustering from power iteration

7.3 Feature extraction using Convolutional Neural Networks:

The features are extracted from the input images using convolutional neural network layers (CNNs). We experiment with two CNN feature extractors to determine the one with greater efficacy.

a) The first feature extractor is a standard CNN Model with 4 layers as shown in Fig 7.3:

Each layer consists of a convolution (kernel of size 3 × 3), followed by Max-Pooling which reduces the size of the image progressively in each layer. The window of the Max-Pool layer is (2 × 2). The ReLU (Rectified Linear Unit) is used as the activation function which zeroes negative values.

Layer Name	Output shape	Next Layer
Input layer	(84,84,3)	Conv0
Conv0	(42,42,64)	Conv1
Conv1	(21, 21,64)	Conv2
Conv2	(10,10,64)	Conv3
Conv3	(5,5,64)	AvgPool
AvgPool	(64)	Output

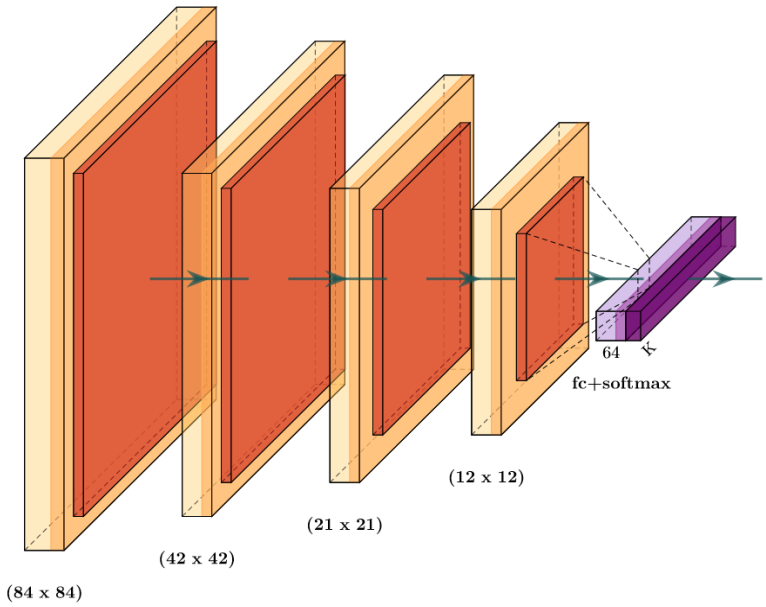

Fig 7.3: Diagram of the Conv-4 Model

b) The second is a Resnet Model with 12 layers [27] as shown in Fig 7.4:

This model is deeper, and each block has an identity shortcut path that helps prevent the vanishing gradient problem that is exacerbated as the number of layers increases. This increased depth improves the feature representation of the model, resulting in greater accuracy.

Layer name	**Output shape**	**Next Layer**
Input layer	(84,84,3)	Block0
Block0	(26,26,64)	Block1
Block1	(9,9,128)	Block2
Block2	(3,3,256)	Block3
Block3	(512)	Output

As shown in Fig 7.4, each block has 3 convolutional layers, a shortcut connection between the first and the third layer and a Max-Pool layer (of window (3 × 3). The shortcut connection adds the output of the first layer and third layer before passing it to the activation function (ReLU again).

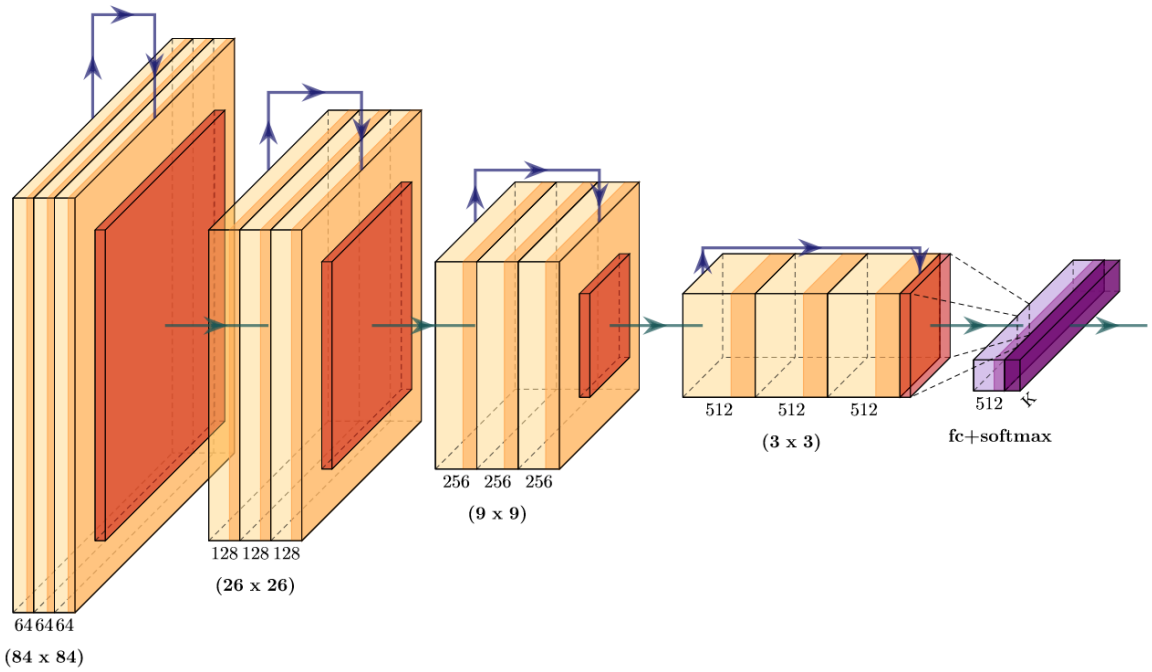

Fig 7.4: Diagram of the Resnet-12 Model

7.4 Pretraining process

The pretraining process is similar to a supervised training schedule. The training set C_{train}, contains classes that have a large number of labelled examples. The objective of the pretraining phase is to learn a good feature representation of the images which can later be fine-tuned to classify unseen classes. Input batches of size 128 are used which improve the efficiency of batch normalization [28], reducing overfitting and improving smoothness of gradients, each image is rotated 4 times for the self-supervision loss [3]. Stochastic Gradient Descent is used to train the network. Algorithm 1 describes the process in pseudocode.

Two fully connected classifiers are trained as shown in Fig 7.5, which use the features extracted by the CNN backbone networks and regularized using the manifold smoothing process as described in Section 7.1.

a) The first classifier C_1 is trained to predict the class labels of the input images. A standard cross entropy loss for classification is used to train this classifier.

The loss function is given by:

$$L_{C1}(x_i, y_i; W_l, \theta) = -ln\, p(y_i|\tilde{z}_i, W_l) \quad (10)$$

W_l is the fully connected layer with softmax activation representing C_1.

b) The second classifier C_2 is utilized to provide a self-supervision type learning signal, where the rotation angle of each input image, (after being rotated by {0°, 90°,180°, 270°}) is predicted. This helps improve the learning signal and provides a certain degree of rotation invariance to the model.

The loss function is given by:

$$L_{C2}(x_i, y_i; W_r, \theta) = -ln\, p(r_i|\tilde{z}_i, W_r) \quad (11)$$

W_r is the fully connected layer with softmax activation representing C_r and r_i is the prediction of the rotation angle.

The overall loss to be minimized is given by:

$$argmin \sum_{i=1}^{128} \sum_{j=1}^{4} L_{C1}(x_i, y_i; W_l, \theta) + L_{C2}(x_i, r_j; W_l, \theta) \quad (12)$$

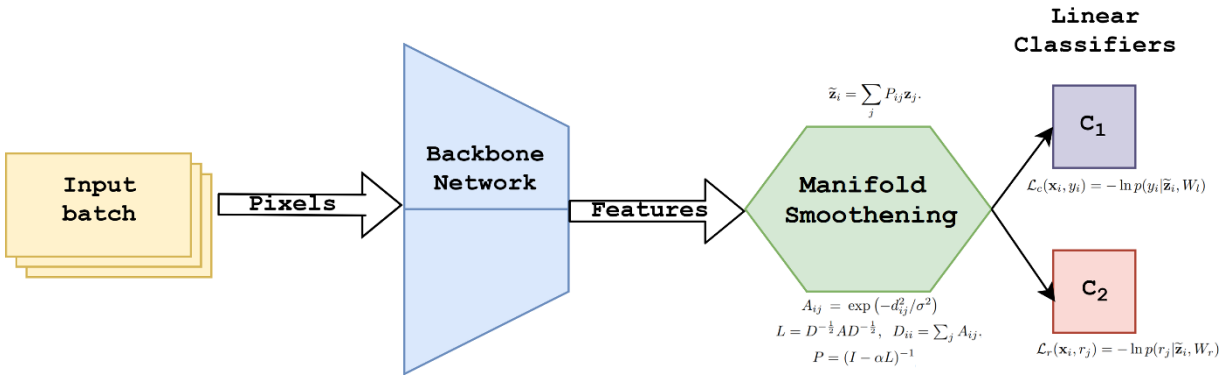

Fig 7.5: Flow Diagram of the pretraining process

Algorithm 1: Pretraining Algorithm

```
Input: Batch of input images
rotated_input_batch = rotate(input_batch, {0,90,180,270}) #Rotating the images
z = backbone_network(rotated_input_batch) # z is the feature representation
A = new matrix(size = z.len * z.len) # Manifold Smoothing
for z_i in z do
    for z_j in z do
    if i == j then
        A[i][j] = 0
    else
        A[i][j] = exp(-(L2Norm(z_i, z_j))^2/Var(L2Norm(z_i, z_j))
A = laplacian(A) # Normalizing the matrix
I = new matrix(size = A.size, type = Identity)
P = matrix_invert(I - α * A) # Smoothing factor α taken as 0.9
z_smooth = P * z
predicted_label = fully_connected_classifier(z_smooth, z_smooth.labels) #C1
predicted_rotation = fully_connected_classifier(z_smooth, rot_labels) #C2
```

7.5 Finetuning process

The finetuning process is performed after the model has been trained on the training set C_{train}. Here, the objective is learning to recognize the unseen classes (part of the test set C_{test}). The label propagation method as described in Section 7.2 is used to find the labels of the unseen classes. Each epoch in finetuning consists of generating an episode as described in Section 6.2, calculating the loss obtained and using backpropagation to adjust the weights accordingly. The finetuning process is defined in Algorithm 2

Two linear classifiers are once again used as shown in Fig 7.6:

a) The classifier $C`_1$ utilizes label propagation to compute the probabilities of the classes in the query set. The logits are converted to class probabilities using the SoftMax function. The loss function is given by:

$$L_{C`1}(x_i, y_i; \theta) = -ln\ p(y_i|\tilde{z}_i, \tilde{Z}, Y_S) \quad (13)$$

b) Since the label propagation loss tends to favor mixing of features, impacting the discriminativeness of the feature representation, a second classifier $C`_2$ is trained with the standard cross entropy loss on the union $S \cup Q$. This helps in preserving the discriminativeness of the feature representation.

The loss function is given by:

$$L_{C`2}(x_i, y_i; W_l, \theta) = -ln\ p(y_i|\tilde{z}_i, W_l) \quad (14)$$

The overall loss to be minimized is the additive combination of the above:

$$argmin \left[\frac{1}{|Q|} \sum_{(x_i, y_i) \in Q} L_{C`1}(x_i, y_i; \theta) + \frac{1}{|S \cup Q|} \sum_{(x_i, y_i) \in S \cup Q} \frac{1}{2} L_{C`2}(x_i, y_i; W_l, \theta) \right] \quad (15)$$

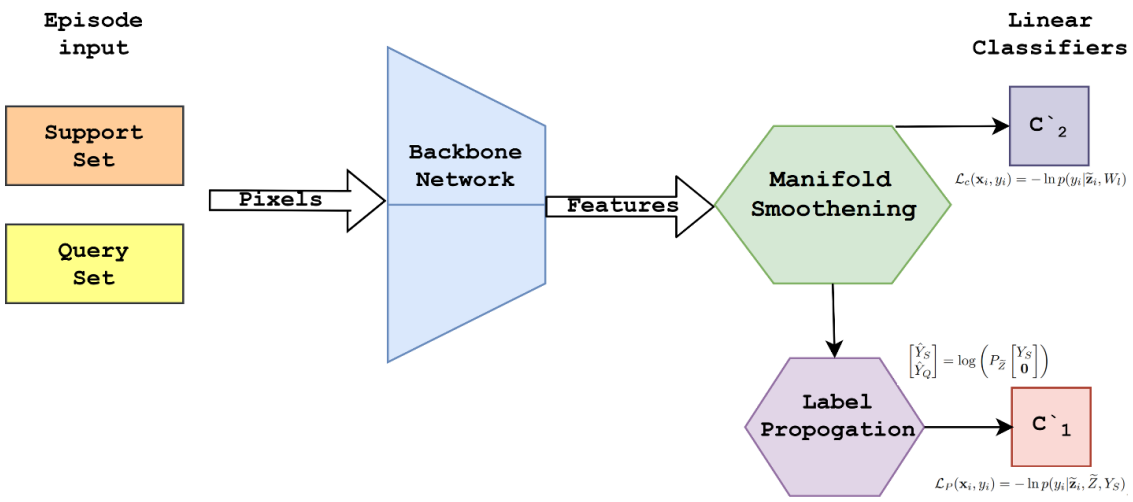

Fig 7.6: Flow Diagram of the finetuning process

Algorithm 2: Finetuning Algorithm

```
Input: Episode of input images
z = backbone_network(rotated_input_batch) # z is the feature representation
A = new matrix(size = z.len * z.len) # Manifold Smoothing
for z_i in z do
        for z_j in z do
            if i == j then
                A[i][j] = 0
            else
                A[i][j] = exp(-(L2Norm(z_i, z_j))^2/Var(L2Norm(z_i, z_j))
A = laplacian(A)
I = new matrix(size = A.size, type = Identity)
P = matrix_invert(I - α * A) # Smoothing factor α taken as 0.9
z_smooth = P * z
lp = label_propagation(z_smooth.support_set, z_smooth.query_set, P)
predicted_unseen = fully_connected_classifier(lp, lp.labels) #Label propagation
predicted_all = fully_connected_classifier(z_smooth, z_smooth.labels) # C`2
```

CHAPTER 8

IMPLEMENTATION

This work uses the dataset described in Section 8.1 to evaluate the model. The components of the network are implemented in Python using the Pytorch library. The Episode Generator is used to create episodic tasks for the finetuning of the network (as described in Section 5.2). The backbone networks are assigned to the GPUs using the CUDA directive. The hyperparameters of the model are given below:

Hyperparameter	Value
Learning Rate (start)	0.1
Optimizer	Stochastic Gradient Descent
CNN Kernel Size	3 × 3
Manifold Smoothing Factor	0.9
Dropout	0.1
Batch Size	128 images
Nesterov Momentum	0.99

8.1 Simulation dataset

The dataset consists of 47 classes representing each base character of the Kannada abugida. Each class consists of 400 samples obtained from different writers. The images were rescaled to 84×84px using the PIL library.

For the purpose of our experiments, we randomly split the 47 classes into 3 sets following the example of [29]:

a) The base set C_{train} (Fig 8.1) consists of 24 classes and has all 400 samples for the supervised pretraining phase. Thus 50% of the dataset is used for the supervised training part. A mixture of vowels and consonants are present in C_{train}. Characters with shapes both simple and complex are represented in the training set.

Fig 8.1: Samples from C_{train}

b) The novel set C_{test} (Fig 8.2) consists of 12 classes which form the unseen set of classes used to test the finetuning approach. This is 25% of the dataset. We can observe that characters both similar in shape to the ones found in C_{train} , as well as uniquely shaped characters can be found in C_{test}.

Fig 8.2: Samples from C_{test}

c) A validation set C_{val} (Fig 8.3) consisting of 11 classes is used to form the validation set used for hyperparameter search and to measure the amount of overfitting. 25% of the dataset is used for this purpose.

Fig 8.3: Samples from C_{val}

8.2 Conv4 Backbone:

```
class Conv4(torch.nn.Module):
    def __init__(self, exp_dict):
        super().__init__()
        self.conv0 = torch.nn.Conv2d(3, 64, 3, 1, 1, bias=False)
        self.bn0 = torch.nn.BatchNorm2d(64)
        self.conv1 = torch.nn.Conv2d(64, 64, 3, 1, 1, bias=False)
        self.bn1 = torch.nn.BatchNorm2d(64)
        self.conv2 = torch.nn.Conv2d(64, 64, 3, 1, 1, bias=False)
        self.bn2 = torch.nn.BatchNorm2d(64)
        self.conv3 = torch.nn.Conv2d(64, 64, 3, 1, 1, bias=False)
        self.bn3 = torch.nn.BatchNorm2d(64)
        self.exp_dict = exp_dict
        if self.exp_dict["avgpool"] == True:
            self.output_size = 64
        else:
            self.output_size = 1600

    def add_classifier(self, no, name="classifier", modalities=None):
        setattr(self, name, torch.nn.Linear(self.output_size, no))

    def forward(self, x, *args, **kwargs):
```

```
        *dim, c, h, w = x.size()
        x = x.view(-1, c, h, w)
        x = self.conv0(x) # 84
        x = F.relu(self.bn0(x), True)
        x = F.max_pool2d(x, 2, 2, 0) # 84 -> 42
        x = self.conv1(x)
        x = F.relu(self.bn1(x), True)
        x = F.max_pool2d(x, 2, 2, 0) # 42 -> 21
        x = self.conv2(x)
        x = F.relu(self.bn2(x), True)
        x = F.max_pool2d(x, 2, 2, 0) # 21 -> 10
        x = self.conv3(x)
        x = F.relu(self.bn3(x), True)
        x = F.max_pool2d(x, 2, 2, 0) # 21 -> 5
        if self.exp_dict["avgpool"] == True:
            x = x.mean(3, keepdim=True).mean(2, keepdim=True)
        return x.view(*dim, self.output_size)
```

8.4 ResNet-12 Backbone:

```
class Block(torch.nn.Module):
    def __init__(self, ni, no, stride, dropout=0, groups=1):
        super().__init__()
        self.dropout = torch.nn.Dropout2d(dropout) if dropout > 0 else lambda x: x
        self.conv0 = torch.nn.Conv2d(ni, no, 3, stride, padding=1, bias=False)
        self.bn0 = torch.nn.BatchNorm2d(no)
        self.conv1 = torch.nn.Conv2d(no, no, 3, 1, padding=1, bias=False)
        self.bn1 = torch.nn.BatchNorm2d(no)
        self.conv2 = torch.nn.Conv2d(no, no, 3, 1, padding=1, bias=False)
        self.bn2 = torch.nn.BatchNorm2d(no)
        if stride == 2 or ni != no:
            self.shortcut = torch.nn.Conv2d(ni, no, 1, stride=1, padding=0)

    def get_parameters(self):
        return self.parameters()
```

```
    def forward(self, x, is_support=True):
        y = F.relu(self.bn0(self.conv0(x)), True)
        y = self.dropout(y)
        y = F.relu(self.bn1(self.conv1(y)), True)
        y = self.dropout(y)
        y = self.bn2(self.conv2(y))
        return F.relu(y + self.shortcut(x), True)


class Resnet12(torch.nn.Module):
    def __init__(self, width, dropout):
        super().__init__()
        self.output_size = 512
        assert(width == 1) # Comment for different variants of this model
        self.widths = [x * int(width) for x in [64, 128, 256]]
        self.widths.append(self.output_size * width)
        self.bn_out = torch.nn.BatchNorm1d(self.output_size)

        start_width = 3
        for i in range(len(self.widths)):
            setattr(self, "group_%d" %i, Block(start_width, self.widths[i], 1,
dropout))
            start_width = self.widths[i]

    def add_classifier(self, nclasses, name="classifier", modalities=None):
        setattr(self, name, torch.nn.Linear(self.output_size, nclasses))

    def up_to_embedding(self, x, is_support):
        for i in range(len(self.widths)):
            x = getattr(self, "group_%d" % i)(x, is_support)
            x = F.max_pool2d(x, 3, 2, 1)
        return x
```

```
def forward(self, x, is_support):
        *args, c, h, w = x.size()
        x = x.view(-1, c, h, w)
        x = self.up_to_embedding(x, is_support)
        return F.relu(self.bn_out(x.mean(3).mean(2)), True)
```

8.5 Episode Generator:

```
class EKannDataset(EDataset):
    h = 84
    w = 84
    c = 3
    split_paths = {"train":"base", "val":"val", "valid":"val", "test":"novel"}
    def __init__(self, data_root, split, input_iterator, size, transforms):
        self.data_root = data_root
        self.split = split
        with open(os.path.join(self.data_root, "fs_lists", "%s.json"
%self.split_paths[split]), 'r') as infile:
            self.metadata = json.load(infile)
        labels = np.array(self.metadata['image_labels'])
        label_map = {l: i for i, l in enumerate(sorted(np.unique(labels)))}
        labels = np.array([label_map[l] for l in labels])
        super().__init__(labels, input_iterator, size, transforms)

    def sample_images(self, indices):
        return
[np.array(Image.open(self.metadata['image_names'][i]).convert("RGB")) for i in
indices]

    def __iter__(self):
        return super().__iter__()
```

```
class EDataset(Dataset):
```

```
    def __init__(self, labels, inp_instance_iterator, size, transforms):
        self.labels = labels
        self.inp_instance_iterator = inp_instance_iterator
        self.labelset = np.unique(labels)
        self.indices = np.arange(len(labels))
        self.transforms = transforms
        self.reshuffle()
        self.size = size

    def reshuffle(self):
        self.clss_idx = [np.random.permutation(self.indices[self.labels == label])
for label in self.labelset]
        self.starts = np.zeros(len(self.clss_idx), dtype=int)
        self.lengths = np.array([len(x) for x in self.clss_idx])

    def gen_fs_t(self, ncls, size):
        classes = np.random.choice(self.labelset, ncls, replace=False)
        starts = self.starts[classes]
        reminders = self.lengths[classes] - starts
        if np.min(reminders) < size:
            return None
        inp_instance_indices = np.array(
            [self.clss_idx[classes[i]][starts[i]:(starts[i] + size)] for i in
range(len(classes))])
        inp_instance_indices = np.reshape(inp_instance_indices, [ncls,
size]).transpose()
        self.starts[classes] += size
        return inp_instance_indices.flatten()

    def inp_instance_t_list(self):
        t_list = []
        t_info = self.inp_instance_iterator.inp_instance()
        ncls, sup_size, q_size, unlabeled_size = t_info
        unlabeled_size = min(unlabeled_size, self.lengths.min() - sup_size -
q_size)
        t_info = FS_VinInpIterator.FS_Vint(ncls=ncls,
```

```
                                                sup_size=sup_size,
                                                q_size=q_size,
                                                unlabeled_size=unlabeled_size)
        k = sup_size + q_size + unlabeled_size
        if np.any(k > self.lengths):
            raise RuntimeError("Requested more inp_instances than existing")
        fs_t = self.gen_fs_t(ncls, k)

        while fs_t is not None:
            t_list.append((t_info, fs_t))
            t_info = self.inp_instance_iterator.inp_instance()
            ncls, sup_size, q_size, unlabeled_size = t_info
            k = sup_size + q_size + unlabeled_size
            fs_t = self.gen_fs_t(ncls, k)
        return t_list
```

```
def __getitem__(self, idx):
        fs_t_info, indices = self.t_list[idx]
        o_aindices = np.argsort(indices)
        o_indices = np.sort(indices)
        ncls, sup_size, q_size, unlabeled_size = fs_t_info
        k = sup_size + q_size + unlabeled_size
        _images = self.inp_instance_images(ordered_indices)
        images = torch.stack([self.transforms(_images[i]) for i in
np.argsort(o_aindices)])
        total, c, h, w = images.size()
        images = images.view(k, ncls, c, h, w)
        del(_images)
        images = images * 2 - 1
        targets = np.zeros([ncls * k], dtype=int)
        targets[o_aindices] = self.labels[o_indices, ...].ravel()
        inp_instance = {"dataset": self.name,
                  "channels": c,
                  "height": h,
                  "width": w,
```

```
                "ncls": ncls,
                "sup_size": sup_size,
                "q_size": q_size,
                "unlabeled_size": unlabeled_size,
                "targets": torch.from_numpy(targets),
                "sup_set": images[:sup_size, ...],
                "q_set": images[sup_size:(sup_size +
                                          q_size), ...],
                "u_set": None if unlabeled_size == 0 else images[(sup_size +
q_size):, ...]}
        return inp_instance


    def __iter__(self):

        self.t_list = []
        while len(self.t_list) < self.size:
            self.reshuffle()
            self.t_list += self.inp_instance_t_list()

        return []

class FS_VinInpIterator():
    FS_Vint = collections.namedtuple("FS_Vint", ["ncls", "sup_size", "q_size",
"unlabeled_size"])
    def __init__(self, ncls, sup_size, q_size, unlabeled_size):
        self.t = self.FS_Vint(ncls, sup_size, q_size, unlabeled_size)

    def inp_instance(self):
        return deepcopy(self.t)
```

8.6 Manifold Smoothing and Label Propagation:

```
def power_iterator(weights, alpha=1, norm_prop=False):
    n = weights.shape[1]
```

```
    identity = torch.eye(n, dtype=weights.dtype, device=weights.device)
    isqrt_diag = 1. / torch.sqrt(1e-4 + torch.sum(weights, dim=-1))
    S = weights * isqrt_diag[None, :] * isqrt_diag[:, None]
    prop = identity - alpha * S
    prop = torch.inverse(prop[None, ...])[0]
    if norm_prop:
        prop = F.normalize(prop, p=1, dim=-1)
    return prop
def sim_matrix(x, rbf_scale):
    b, c = x.size()
    sq_dist = ((x.view(b, 1, c) - x.view(1, b, c))**2).sum(-1) / np.sqrt(c)
    mask = sq_dist != 0
    sq_dist = sq_dist / sq_dist[mask].std()
    weights = torch.exp(-sq_dist * rbf_scale)
    mask = torch.eye(weights.size(1), dtype=torch.bool, device=weights.device)
    weights = weights * (~mask).float()
    return weights

def manifold_smoothing(x, alpha, rbf_scale, norm_prop, prop=None):
    if prop is None:
        weights = sim_matrix(x, rbf_scale)
        prop = power_iterator(
            weights, alpha=alpha, norm_prop=norm_prop)
    return torch.mm(prop, x)

def label_prop(x, labels, nclasses, alpha, rbf_scale, norm_prop, apply_log,
prop=None, epsilon=1e-6):
    labels = F.one_hot(labels, nclasses + 1)
    labels = labels[:, :nclasses].float()  # the max label is unlabeled
    if prop is None:
        weights = sim_matrix(x, rbf_scale)
        prop = power_iterator(
            weights, alpha=alpha, norm_prop=norm_prop)
    y_pred = torch.mm(propr, labels)
    if apply_log:
        y_pred = torch.log(y_pred + epsilon)
```

```
    return y_pred
```

CHAPTER 9

RESULT

State of the art results is achieved using the Label Propagation and Manifold Smoothing model for the problem of "Recognition of Handwritten Kannada Characters in a Few-Shot Learning perspective". This section gives insights of the result obtained in terms of Pretraining Accuracy (seen classes), Finetuning accuracy (seen and unseen classes) using 1-shot and 5-shot learning (support set of 1 example and 5 examples respectively). Comparison of result with the existing work is done here.

9.1 Performance evaluation

Two different feature extractors are evaluated using the episodic framework mentioned in Section 5.2. The average accuracy of classification over 1000 episodes is used as the metric for evaluation. The first feature extractor Conv4, has a faster training and inference time owing to its simplicity, and seems to benefit much more from the finetuning phase as compared to the second feature extractor, the Resnet-12. However, much better accuracies are obtained by the larger Resnet-12 network. This can be attributed to the greater width of the network, which allows a larger number of learnable parameters to be used for classification. Although there is a greater amount of overfitting as evidenced by the difference in test and validation accuracies, the performance on finetuning shows that the framework has good generalization capability.

9.2 Conv4 network:

In Fig 9.1, we observe the convergence of training at 44 epochs, and due to the episodic nature of training, we observe large swings prior to convergence. The loss is monotonically decreasing over a large number of epochs, with a bump close to the convergence point as show in Fig 9.2.

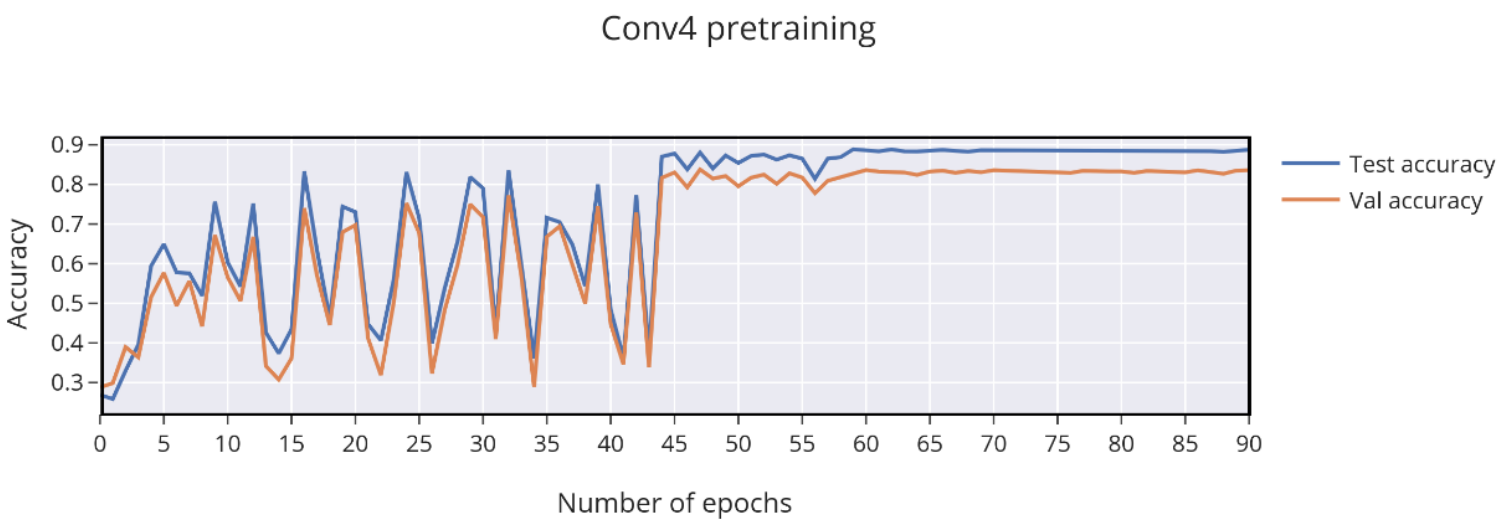

Fig 9.1: Pretraining accuracy vs Number of epochs

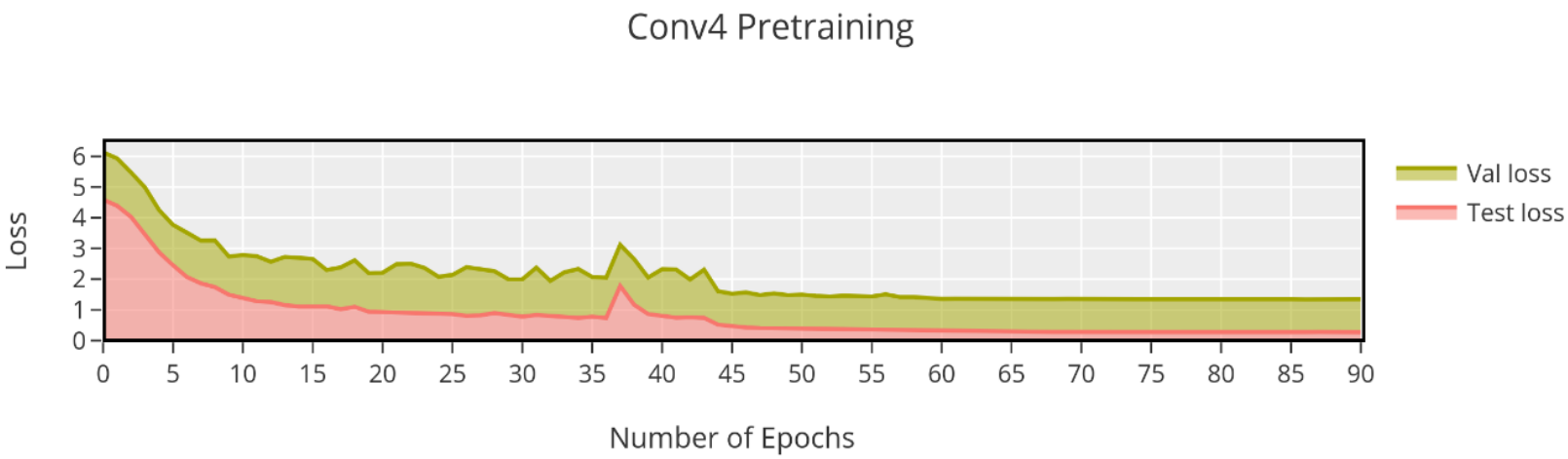

Fig 9.2: Pretraining loss vs Number of epochs

In Fig 9.3, we observe that the pretrained model starts out at 50% accuracy and steadily increases with finetuning epochs until epoch 32 where the network converges to 91.04% accuracy. The loss (Fig 9.4) decreases and stabilizes.

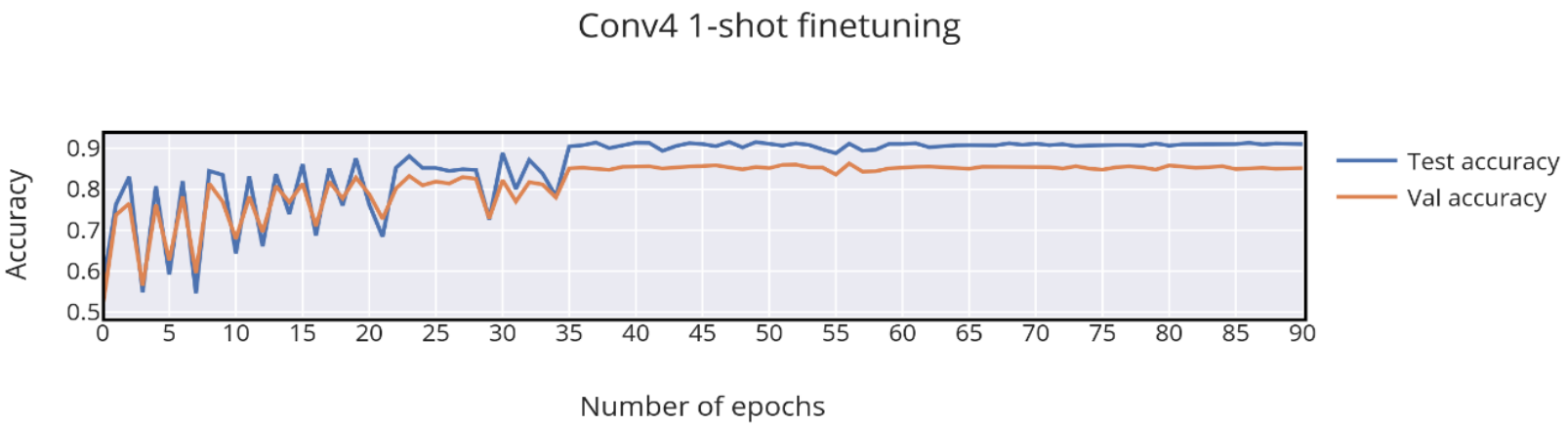

Fig 9.3: 1-Shot Finetuning accuracy vs Number of epochs

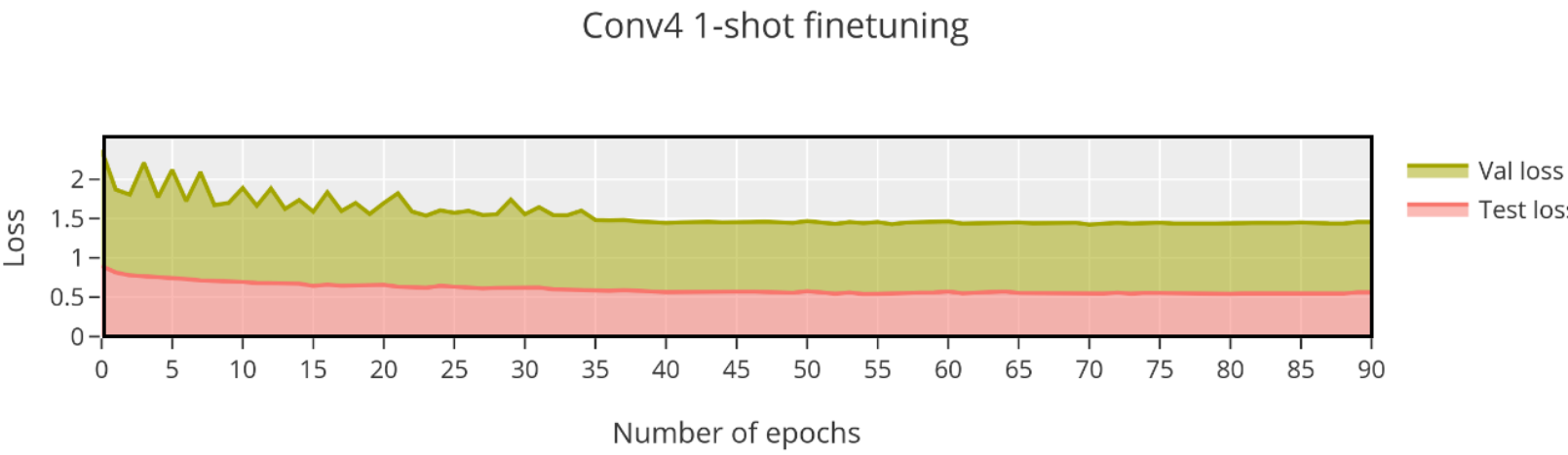

Fig 9.4: 1-Shot Finetuning loss vs Number of epochs

In 5-shot finetuning we observe a higher initial accuracy of 83% accuracy which reduces when more unseen classes are initially encountered, the network finally converges at 37 epochs to an accuracy of 96.88%. We observe the increase in validation loss (Fig 9.6) corresponding to the more difficult episodes.

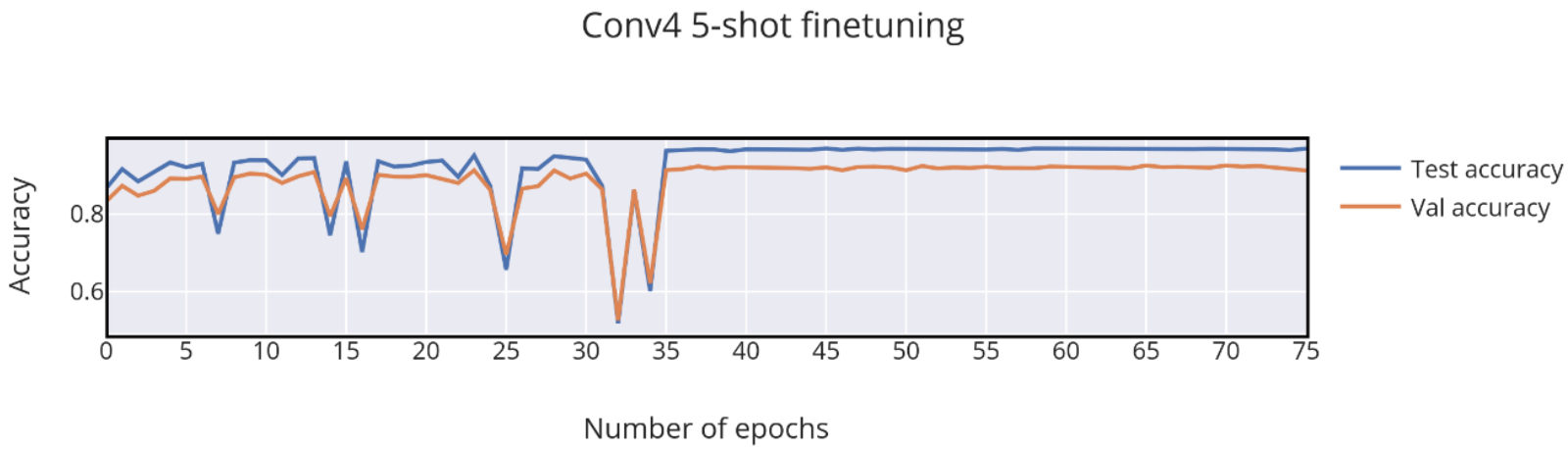

Fig 9.5: 5-Shot Finetuning accuracy vs Number of epochs

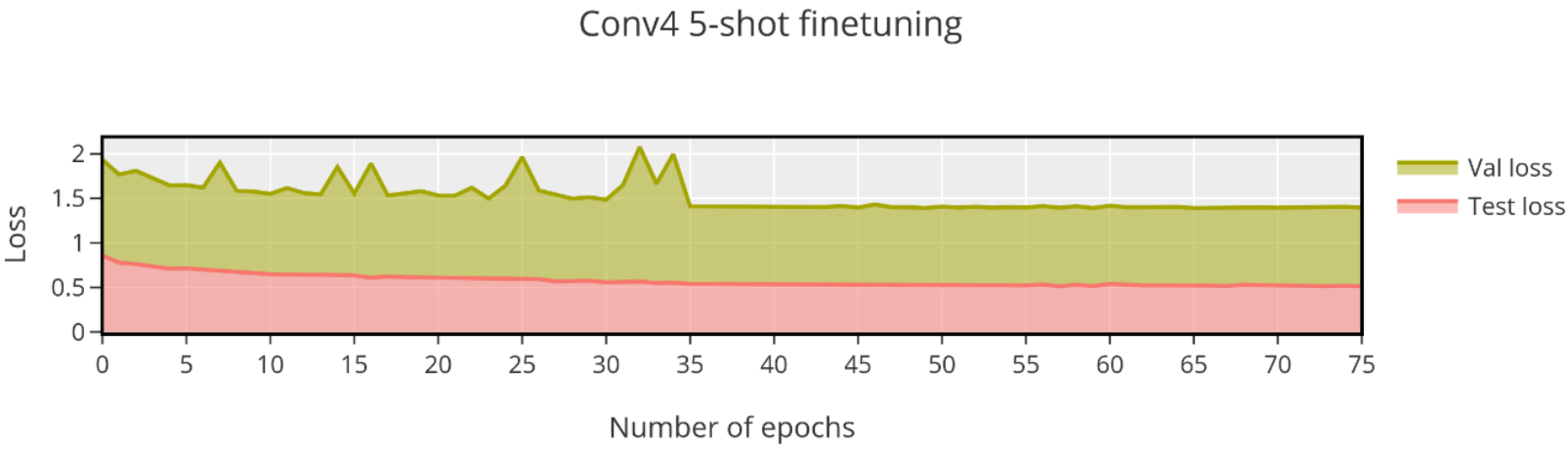

Fig 9.6: 5-Shot Finetuning loss vs Number of epochs

9.3 ResNet-12 network:

Compared to Fig 9.1, we observe a shorter convergence time (35 epochs) and a higher pretraining accuracy being achieved (98.66%) in Fig 9.7. This can be attributed to the increased number of channels (width) and layers (depth) of the backbone network. The loss function decreases faster as well compared to Fig 9.2.

Fig 9.7: Pretraining accuracy vs Number of epochs

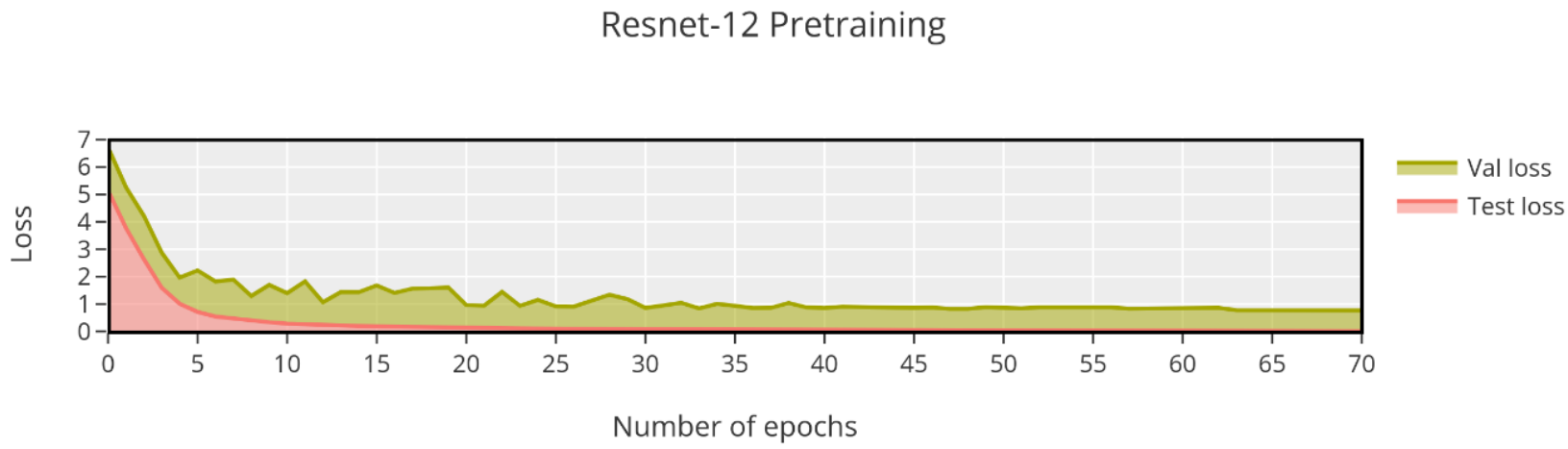

Fig 9.8: Pretraining loss vs Number of epochs

Compared to Fig 9.3, we observe that finetuning does not increase the accuracy of the network by a significant amount. This can be attributed to the stronger convergence during pretraining which allows better inference on novel classes without much finetuning required. The low variance of the loss (Fig 9.10) indicates saturation of the network.

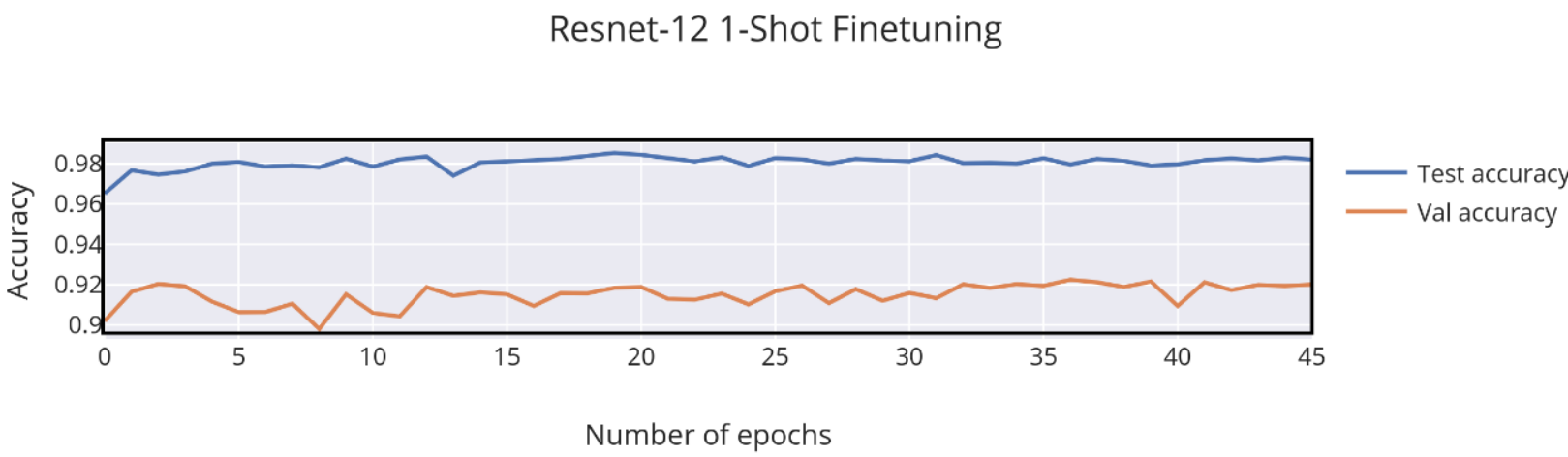

Fig 9.9: 1-Shot Finetuning accuracy vs Number of epochs

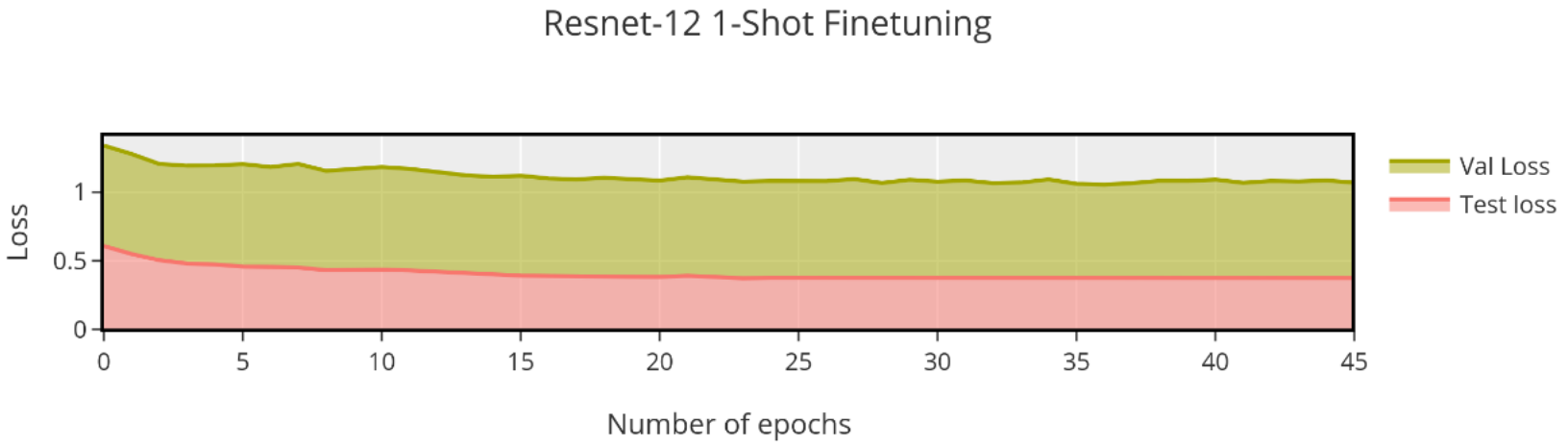

Fig 9.10: 1-Shot Finetuning loss vs Number of epochs

Similar to Fig 9.9 and Fig 9.10, we observe that finetuning doesn't increase the accuracy significantly. Due to the large number of support images (5 compared to 1 in 1-shot), we obtain a higher accuracy (99.13 % compared to 98.17% in Fig 9.9).

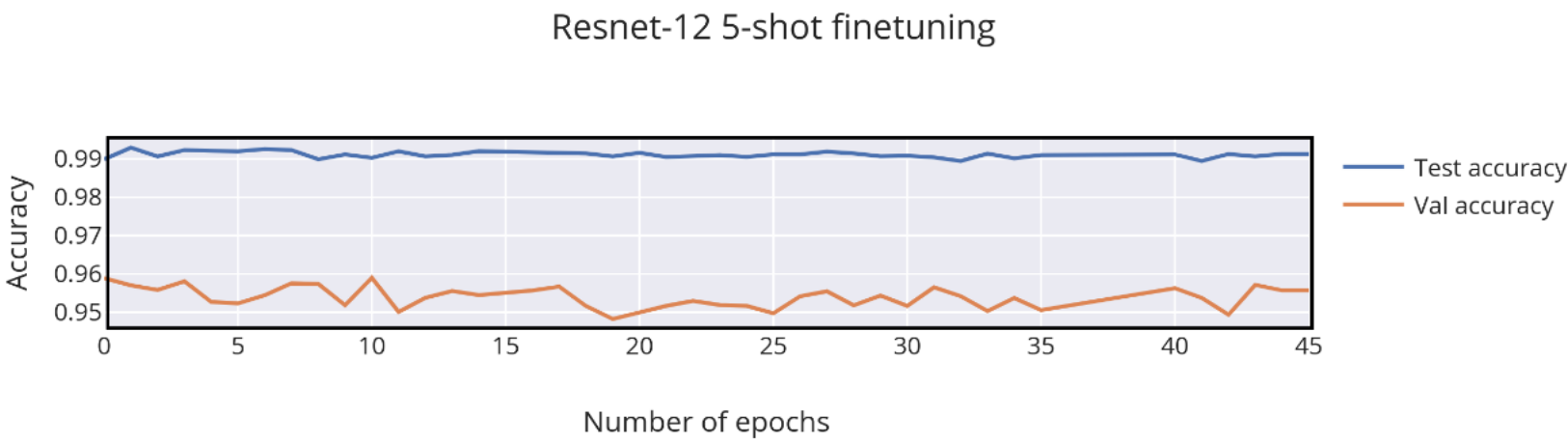

Fig 9.11: 5-Shot Finetuning accuracy vs Number of epochs

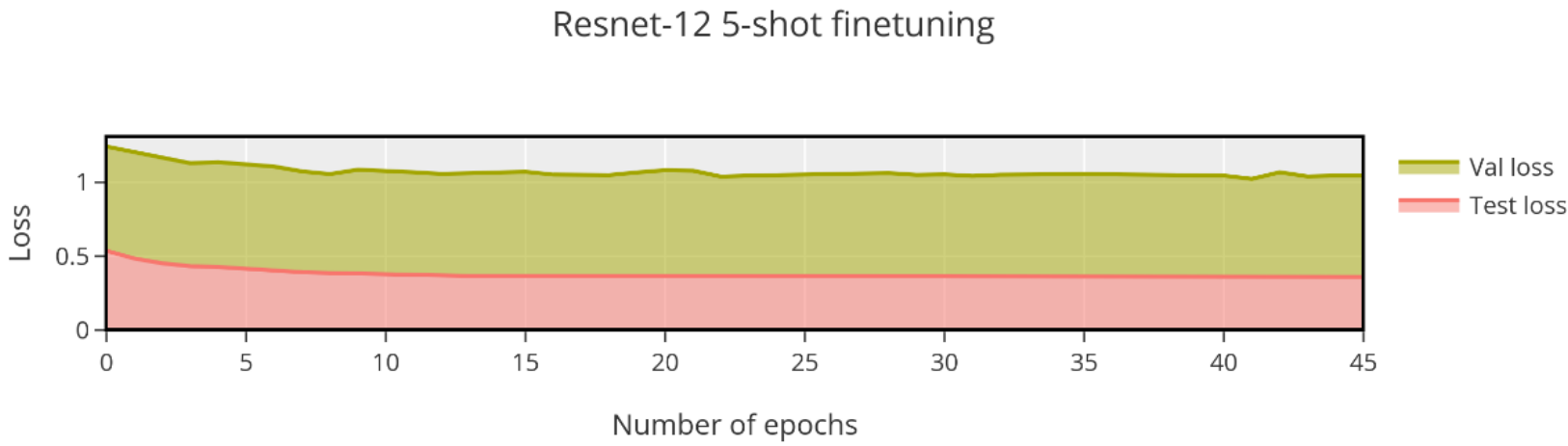

Fig 9.12: 5-Shot Finetuning loss vs Number of epochs

9.4 Comparison between the networks:

The graphs Fig 9.7 and Fig 9.1 show that the Resnet model converges faster in pretraining compared to the Conv4 model. The training is stopped when the learning rate reaches 0.00001. The learning rate is reduced to 10% after every 10 epochs if there is no improvement in the loss (a plateau is reached). A Conv4 model requires a larger number of epochs to converge during the finetuning phase as well compared to the Resnet model. (Fig 9.3, Fig 9.9). We observe that there is a significant increase of test and validation accuracy during finetuning for the Conv4 model (Fig 9.3, Fig. 9.5), while finetuning doesn't increase the accuracy of the Resnet-12 model by a significant amount. (Fig 9.9, Fig 9.11). The increase in the number of support set samples from 1 to 5 provides a boost of ~5% accuracy for the Conv4 model and ~4% for the Resnet-12 model (comparing the validation accuracies). We can infer that by increasing the number of labelled examples for the unseen classes, we can initially expect about a 4% increase in accuracy. The gain per increase of labelled examples should diminish as it converges to supervised learning.

Pretraining:

Backbone network	**Conv4**	**Resnet-12**
Test accuracy	88.74 %	98.66%
Validation accuracy	83.60%	95.69%
Number of epochs	91	71

Finetuning 1-shot:

Backbone network	**Conv4**	**Resnet-12**
Test accuracy	91.04%	98.17%
Validation accuracy	85.14%	91.87%
Number of epochs	89	47

Finetuning 5-shot:

Backbone network	**Conv4**	**Resnet-12**
Test accuracy	96.88%	99.13%
Validation accuracy	91.22%	95.57%
Number of epochs	75	44

9.5 Comparison with previous works:

We compare the test accuracy and validation accuracy of the 5-shot approach with the values obtained by training the Convolutional Neural Network and Capsule Network as provided in [30]:

Classifier model	**Proposed Method with Resnet-12(training + test set)**	**Capsule Network (vowels and consonants)**	**Convolutional Neural Network (average of vowels and consonants)**
Training accuracy	98.66%	97.81%	99.5%
Validation Accuracy	95.57%	89.93%	87%
Number of epochs for convergence	40(as observed in the graph)	50	50

We can observe that the number of epochs required for convergence is similar for all three networks. The amount of overfitting in the Label Propagation network is lower as indicated by the 3% difference between the training and validation accuracies, compared to the 7% difference in the capsule network and 12% difference in the CNN used in [30].

Authors	**Method**	**Accuracy obtained**
Manjunath *et al.*, 2010 [11]	PCA+ANN	88.64%
Karthik *et al.*, 2015 [21]	SVM+HOG	96.41%
Karthik *et al.*, 2018 [24]	Deep Belief Networks	97.04%
Ramesh *et al.*,2019 [30]	Capsule Networks	98.7%
Proposed method (5 shot)	Manifold Smoothing with Label Propagation Network	99.13%

CHAPTER 10

CONCLUSION

We propose a new Offline Handwritten Character recognition framework that has the qualities of robustness to variations in input and easy generalization. The incorporation of unseen character classes into the framework doesn't require the retraining of the entire network to achieve good accuracies. The incorporation is also data efficient as it only requires a small number of labelled samples to learn to classify the newer classes (only 1 example in 1-shot and 5 examples in 5-shot).

The use of Resnet-12(a deep residual CNN), label propagation and manifold smoothing help reduce the effect of training class imbalance bias as well as reducing the overfitting of the network during the pretraining phase. The accuracy obtained on the 5-shot accuracy makes this framework competitive with the supervised learning counterparts, despite the large reduction in the number of labelled samples available (for the novel classes).

The framework can be further enhanced by improving the matrix inversion complexity by introducing block-sparse and sparse inversion techniques, which allows for scalability. The incorporation of the label propagation algorithm into a LSTM and language model system will help in creating few-shot learning-based word, sentence and document optical character recognition systems.

REFERENCES

[1] Oriol Vinyals, Charles Blundell, *et al.* "Matching networks for one shot learning." In *Advances in Neural Information Processing Systems*, pages 3630–3638, 2016.

[2] Alex Nichol, Joshua Achiam, and John Schulman "On first-order meta-learning algorithms". *arXiv preprint* arXiv:1803.02999, 2018.

[3] Gidaris, S., Bursuc, A., Komodakis, *et al.* "Boosting few-shot visual learning with self-supervision." In: *Conference on Computer Vision and Pattern Recognition*. pp. 8059–8068 (2019)

[4] Denny Zhou, Olivier Bousquet, *et al.* "Learning with local and global consistency". In *Advances in Neural Information Processing Systems*, pages 321–328, 2004

[5] Y. Zhu, C. Yao, and X. Bai, "Scene text detection and recognition: Recent advances and future trends," *Frontiers of Computer Science*, vol. 10, no. 1, pp. 19–36, 2016.

[6] L. R. Ragha and M. Sasikumar, "Feature analysis for handwritten kannada kagunita recognition," *International Journal of Computer Theory and Engineering*, vol. 3, no. 1, p. 94, 2011.

[4] A. Amin, "Off-line arabic character recognition: the state of the art," *Pattern recognition*, vol. 31, no. 5, pp. 517–530, 1998.

[7] Alireza Alaei, P. Nagabhushan, and Umapada Pal. 2011. "A Benchmark Kannada Handwritten Document Dataset and Its Segmentation". In *Proceedings of the 2011 International Conference on Document Analysis and Recognition (ICDAR '11)*. IEEE Computer Society, USA, 141–145.(2011)

[8] A. Alaei, U. Pal, and P. Nagabhushan, "Dataset and ground truth for handwritten text in four different scripts," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 26, no. 04, p.1253001, 2012.

[9] S. M. Obaidullah, C. Halder, *et al.* “Phdindic 11: page-level handwritten document image dataset of 11 official indic scripts for script identification,” *Multimedia Tools and Applications*, vol. 77, no. 2, pp. 1643–1678, 2018.

[11] V. Manjunath Aradhya, S. Niranjan, and G. Hemantha Kumar, “Probabilistic neural-network based approach for handwritten character recognition,” *International Journal of Computer & Communication Technology*, vol. 1, no. 2, 3, 4., pp. 9–13, 2010.

[12] Niranjan, S. K., Vijaya Kumar, and Hemantha Kumar. "FLD based unconstrained handwritten Kannada character recognition." In *2008 Second International Conference on Future Generation Communication and Networking Symposia*, vol. 3, pp. 7-10. IEEE, 2008.

[14] A. Majumdar and B. B. Chaudhuri, "Curvelet-Based Multi SVM Recognizer for Offline Handwritten Bangla: A Major Indian Script," *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Parana, 2007, pp. 491-495.

[13] Naveena, C., and VN Manjunath Aradhya. "An impact of ridgelet transform in handwritten recognition: A study on a very large dataset of kannada script." In *2011 World Congress on Information and Communication Technologies*, pp. 618-621. IEEE, 2011.

[14] V. N. Murthy and A. G. Ramakrishnan, “Choice of classifiers in hierarchical recognition of online handwritten kannada and tamil aksharas.” *J. UCS*, vol. 17, no. 1, pp. 94–106, 2011.

[15] B. Dhandra, M. Vijayalaxmi, G. MukarambiHu, and M. HanGarge, “Writer identification by texture analysis based on kannada handwriting,” *Int. J. Comm. Netw. Secur*, vol. 1, no. 4, pp. 80–85, 2012.

[16] Ramya, S., & Shama, K. “Comparison of SVM kernel effect on online handwriting recognition: A case study with kannada script.” In *Data Engineering and Intelligent Computing - Proceedings of IC3T 2016* (Vol. 542, pp. 75-82). (2017)

[17] D. Keysers, T. Deselaers, H. A. Rowley, L.-L. Wang, and V. Carbune, “Multi-language online handwriting recognition.” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1180–1194, 2017.

[18] Ramya, S and Shama, Kumara, “The effect of pre-processing and testing methods on online kannada handwriting recognition: Studies using signal processing and statistical techniques,” *Pertanika Journal of Science and Technology*, vol. 26, no. 2, pp. 671–690, 2018.

[19] V. C. Hallur and R. Hegadi, “Offline Kannada handwritten numeral recognition: Holistic approach,” pp. 632–637, 2014.

[20] P. Bannigidad and C. Gudada, "Restoration of degraded Kannada handwritten paper inscriptions (Hastaprati) using image enhancement techniques," *2017 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore*, pp. 1-6, (2017)

[21] Karthik, S., and K. Srikanta Murthy. "Handwritten Kannada numerals recognition using histogram of oriented gradient descriptors and support vector machines." In *Emerging ICT for Bridging the Future-Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2*, pp. 51-57. Springer, Cham, 2015.

[22] P. Yadav and M. Kumar, “Kannada character recognition in images using histogram of oriented gradients and machine learning,” pp. 265– 277, 2018.

[23] Mirabdollah, M.H., Mohamed, M.A. and Mertsching, B., “Distributed averages of gradients (dag): A fast alternative for histogram of oriented gradients.” In *Robot World Cup* (pp. 97-108). Springer, Cham.(2016)

[24] S. Karthik and K. S. Murthy, “Deep belief network based approach to recognize handwritten kannada characters using distributed average of gradients,” *Cluster Computing*, pp. 1–9, 2018.

[25] Ramesh G, Ganesh N. Sharma, J. Manoj Balaji and Champa H. N. , "Offline Kannada Handwritten Character Recognition Using Convolutional Neural Networks," 2019 IEEE *International WIE Conference on Electrical and Computer E Engineering* , pp. 1-5 (2019)

[26] Lee, W.S., Bartlett, P.L., Williamson, R.C.: “Lower bounds on the vc dimension of smoothly parameterized function classes*.” Neural Computation* 7(5), 1040–1053 (1995)

[27] Kaiming He, Xiangyu Zhang, *et al.* “Deep residual learning for image recognition.” In *Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[28] Sergey Ioffe and Christian Szegedy. “Batch normalization: Accelerating deep network training by reducing internal covariate shift”. In *International Conference on Machine Learning*, pages 448–456, 2015.

[29] Ren, M., Ravi, S., Triantafillou, E., Snell, J, *et al.*” Meta-learning for semi-supervised few-shot classification”. In: *International Conference on Learning Representations* (2018)

[30] Ramesh. G, J. Manoj Balaji, Ganesh. N. Sharma, Champa H.N., “Recognition of Off-line Kannada Handwritten Characters by Deep Learning using Capsule Network.” *International Journal of Engineering and Advanced Technology*. (2019).