ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

DEEP LEARNING BASED DYNAMIC TURKISH SIGN LANGUAGE RECOGNITION WITH LEAP MOTION

M.Sc. THESIS

Burçak DEMİRCİOĞLU KAM

Department of Computer Engineering

Computer Engineering Programme

JULY 2020

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

DEEP LEARNING BASED DYNAMIC TURKISH SIGN LANGUAGE RECOGNITION WITH LEAP MOTION

M.Sc. THESIS

Burçak DEMİRCİOĞLU KAM
(504171505)

Department of Computer Engineering

Computer Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. Hatice Köse

JULY 2020

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DERİN ÖĞRENME TABANLI LEAP MOTION İLE DİNAMİK TÜRK İŞARET DİLİ TANIMA

YÜKSEK LİSANS TEZİ

Burçak DEMİRCİOĞLU KAM
(504171505)

Bilgisayar Mühendisliği Anabilim Dalı

Bilgisayar Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Hatice Köse

TEMMUZ 2020

Burçak DEMİRCİOĞLU KAM, a M.Sc. student of ITU Graduate School of Science Engineering and Technology 504171505 successfully defended the thesis entitled "DEEP LEARNING BASED DYNAMIC TURKISH SIGN LANGUAGE RECOGNITION WITH LEAP MOTION", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Hatice Köse** Istanbul Technical University	.....
Jury Members :	**Prof. Dr. Duygun Erol Barkana** Yeditepe University	.....
	Assoc. Prof. Dr. Gülay Öke Günel Istanbul Technical University	.....
		.....

Date of Submission : **15 June 2020**
Date of Defense : **24 July 2020**

To my husband and my mother who supported me no matter what,

FOREWORD

This master thesis is written under the supervision of Assoc. Prof. Dr. Hatice Köse, at Istanbul Technical University, during the time period from fall 2019 until spring 2020 which also includes the times when the infamous *COVID-19* situation is occurred. On top of being a recently relocated expat to a different country, the health situation made the development of this thesis much harder than it originally was because of being under house lockdown and trying to persist the social distance.

The aim of the thesis is to create a sign language recognition system using deep learning approaches for the dynamic, 2 handed signs which are the key building blocks of the sign languages.

I want to thank my supervisor, Hatice Köse, of being more than great help during the development of this thesis, my mother Güliz Demircioğlu and my husband Uğur Kam of being so supportive and patient with me during this process.

July 2020 Burçak DEMİRCİOĞLU KAM

TABLE OF CONTENTS

Page

ABBREVIATIONS

App : Appendix
ArSL : Arabic Sign Language
ASL : American Sign Language
Auslan : Australian Sign Language
BLSTM-NN : Bidirectional Long Short-Term Memory Neural Networks
CNN : Convolutional Neural Network
CSL : Chinese Sign Language
FC : Fully Connected
HMM : Hidden Markov Model
ISL : Indian Sign Language
k-NN : k-Nearest Neighbour
LDA : Linear Discriminant Analysis
LMC : Leap Motion Controller
LSTM : Long Short-Term Memory
MLP : Multilayer Perceptron
NBC : Naïve Based Classifier
NN : Neural Network
ReLU : Rectified Linear Unit
RF : Random Forest
RGB : Red Green Blue
RNN : Recurrent Neural Network
SDK : Software Development Kit
SGD : Stochastic Gradient Decent
SVM : Support Vector Machine
TID : Turkish Sign Language
UKF : Unscented Kalman Filter

LIST OF TABLES

Page

LIST OF FIGURES

Page

DEEP LEARNING BASED
DYNAMIC TURKISH SIGN LANGUAGE RECOGNITION
WITH LEAP MOTION

SUMMARY

Communication with the society is a challenging issue for deaf and hard-of-hearing people. This project aims to fulfill the need of a robust, usable and cost efficient communication system by using minimal resources such as compact sensor devices in combination with a mobile computational device, like a laptop, tablet or even a mobile phone. The purpose of this thesis is to develop an efficient gesture/sign recognition system using the power of machine learning and deep learning approaches, to achieve this goal. Therefore in this project, sign language recognition is achieved by adopting various traditional machine learning and deep learning techniques after the desired dataset is created. The project can be explained in 2 consecutive steps as; the data collection and processing step and design/implementation and evaluation of these machine learning classifiers and deep learning models with the collected data.

Leap Motion Controller device which is a compact and cost efficient sensor for the hand gesture recognition is used in this project as the sensory hardware to collect the data. For the first step, a data collection tool is developed for *LMC*, to make the data collection process more efficient, easy and fast. Also with this tool a need in the literature is fulfilled, since there is no similar tool developed for the *Leap Motion* device. The tool introduces many benefits for the users who are creating the data samples for the dataset. These benefits can be summarized as; ease of use, automatic labeling, including reference sign video and device status checking. With these benefits in mind, the workflow of the application is as following; user connects the *Leap Motion Controller* to her/his computer, starts the tool via the *.exe* file, chooses a sign from the list provided in the interface, watches the reference video of the sign, pushes the start button, tool checks the device and driver connection and informs the user about their status, user sees herself/himself in the *webcam* viewer, tool waits for the user to show her/his hands to the sensor, user shows both hands to the sensor, tool starts a countdown and starts recording when the countdown ends, user simulates the sign and pushes stop button, tool stops the recording and saves the collected data files into a local folder which are timestamped and labeled with the information from the sign list member that the user selected at the beginning of the process. Also this application designed as highly configurable as possible to make it usable as broadly as possible. For this purpose the sign selection list is constructed from the folder that includes all the reference videos, which means if, for any other research purposes, another set of hand sign samples are desired to be collected, the researcher just needs to change the videos in the reference video folder, no any other change is necessary to make the tool configured to satisfy the new purposes, especially any code change is not required.

For the dataset, 12 dynamic 2 handed word signs are chosen from the Turkish Sign Language (*TID*). These signs are selected based on the results of our previous studies,

which involve the recognition of static hand gestures in *TID*. The chosen signs include one of these static hand gestures, applied in dynamic hand motion patterns.

After the tool is created, for the experiment purposes, total of 200 samples per sign are collected from multiple signers under various different environmental conditions. With these collected samples, a dataset is created with a total of 2400 samples which are evenly distributed across the signs. After dataset is created some preprocessing steps are applied to data to make it more convenient and efficient for the learning modals.

For the recognition step, six different traditional machine learning classifiers as *Gaussian Naïve Bayes*, *Support Vector Machine*, *K-Nearest Neighbour*, *Random Forest*, *Linear Discriminant Analysis* and *Multilayer Perceptron* classifiers and 3 different deep learning models which are *Convolutional Neural Networks*, *Basic Recurrent Neural Network*s and *Long Short-Term Memory Neural Network*s are created and tested. While the experiments are conducted, various different hyperparameters are used with different loss and optimizer functions to increase the accuracy of the system. As the loss function, *Cross Entropy Loss* is used with *Stochastic Gradient Decent* and *Adam* optimizer functions.

The experiments are done in three parts. First, the traditional machine learning methods are experimented on with multiple varying hyperparameters. The most successful method is found as the optimized *LDA* classifier with 99.9% train, 98.2% validation and 95% test accuracy. As the second part of the experiments, deep learning models are experimented on with again multiple varying hyperparameters. All three models are optimized by the analysis of multiple runs and as the result *CNN* became the most successful approach among them with 100% training, 97.7% validation and 96.7% test accuracy. For the last part of the experiments, 5 different train-test set pairs are created with shuffling the collected dataset and all the deep learning methods together with *LDA* are run with these pairs for 5 times to get the average results. As the final best results; 98.8% test accuracy is reached with the optimized *LDA* classifier and *CNN* model while the best accuracy remained at 88.3% for the *RNN* and 85.0% for the *LSTM* models. Also the overall average results became like 97.4% for *LDA*, 96.9% for *CNN*, 82.8% for *RNN* and 79.2% for *LSTM* as the test accuracies. From these results it is concluded that for the purpose of the study and the used dataset, it is seen that the most successful approaches are the *LDA* and *CNN*. As the last evaluation because the test accuracies of these two approaches are pretty close to each other, the confusion matrices of these two approaches are analyzed and it is seen that *CNN*'s sign individual results are better than *LDA* by having more homogeneously distributed right predictions over all the signs.

DERİN ÖĞRENME TABANLI LEAP MOTION İLE DİNAMİK TÜRK İŞARET DİLİ TANIMA

ÖZET

Toplumumuzda, sağır ya da işitme problemi yaşayan kişiler günlük yaşamlarında özellikle işaret dili bilinmeyen ortamlarda çevreleriyle iletişim zorluğu yaşayabilmektedir. Bu tez kapsamında geliştirilen ve küçük boyuttaki bir sensor ve işlemci gücü olan bilgisayar, tablet ve/veya telefon gibi bir alet yardımıyla çalışan iletişim sistemi ile bu problemin çözülmesi hedef olarak belirlenmiştir. Bu hedefi gerçekleştirmek için bu projede verimli bir tanıma sisteminin derin öğrenme metotları kullanılarak geliştirilmesi amaçlanmıştır. Gerekli olan veri seti oluşturulduktan sonra, derin öğrenme yöntemleri kullanılarak işaret dili tanıma yapılmıştır. Bu proje 2 farklı bölüme ayrılabilir. İlk bölümde data toplama ve hazırlama yapılmış, ikinci bölümde ise geleneksel makine öğrenmesi sınıflandırıcıları ve derin öğrenme modellerinin oluşturulup hazırlanan veri seti kullanılarak bu modeller ve sınıflandırıcılar üzerinde deneyler yapılmıştır.

İlk bölümde veri toplama işlemi için bir uygulama geliştirilmiştir. Veri toplama işlemi için *Leap Motion Controller* cihazı sensor olarak seçilmiş ve uygulama bu cihaz ile çalışacak şekilde tasarlanmıştır. Geliştirme yapılırken uygulamanın özellikle kolay kullanılabilir ve anlaşılabilir olmasına ve aynı zamanda da veri toplama işlemini büyük oranda hızlandırmasına önem verilmiştir. Aynı zamanda bu çalışmadan önce benzer bir uygulamanın yapılmamış olmasından dolayı bu çalışma literatürde var olan bir boşluğu da doldurmuştur.

Uygulama Windows Forms uygulaması olarak *.NET* çatısı ile *C#* programlama dili kullanılarak oluşturulmuştur. Uygulamayı geliştirirken *Leap Motion*'ın resmi olarak *C#* programlama dili için yayınladığı kütüphanelerin yanı sıra kamera görüntüsü alıp görüntüleyebilmek için *Windows Media Player* kütüphaneleri ve *Accord.Net* çatısında bulunan kütüphaneler kullanılmıştır. Geliştirilen uygulamanın veri seti oluşturma aşamasını daha efektif ve kolay hale getiren birçok artı noktası bulunmaktadır. Bu noktalar şu şekilde özetlenebilir; kullanım kolaylığı, otomatik etiketleme, referans işaret videosu içerme ve cihaz durum kontrolü.

Uygulamanın kullanım akışı şu şekildedir; kullanıcı *Leap Motion Controller* cihazını bilgisayarına bağlar ve *.exe* uzantılı dosya ile uygulamayı başlatır, uygulamanın ara yüzünde bulunan işaret listesinden veri üretmek istediği işareti seçer, seçilen işaretin referans videosunu izler ve hazır olduğunda başlatma butonuna basar. Uygulama sensor cihazının ve cihaz sürücüsünün durumunu kontrol eder ve kullanıcıya rapor eder, kullanıcı bilgisayarın donanımında bulunan kamera aracılığıyla alınan görüntüyü gösteren bölümden kendini görüntüler ve hareketi doğru yapıp yapmadığını kontrol eder. Uygulama kullanıcının iki elini de sensöre göstermesini bekler, kullanıcı iki elini birden sensöre gösterdiğinde uygulama geri sayımı ve bu geri sayım bittiğinde görüntü ve veri kaydını başlatır. Kullanıcı seçmiş olduğu işareti videoda gösterildiği

şekilde ve *Leap Motion Controller* sensörünün sınırları dışına taşmadan yapar ve durdurma butonuna basar. Uygulama kayıt işlemini durdurur ve toplanan veriler ile oluşturduğu dosyaları oluşturma zamanı bilgisi ve kullanıcının işlemin başında listeden seçmiş olduğu işaretin ismini kullanarak etiketleyip, yerel klasöre kaydeder. Aynı zamanda bu uygulamanın tasarımı yapılırken, kullanım alanının mümkün olduğunca geniş olabilmesi için büyük önem gösterilmiştir. Bu amaç doğrultusunda kullanıcının işaret seçimini yaptığı liste statik olarak veya kod aracılığı ile değil referans videoların bulunduğu klasörün içeriği kullanılarak oluşturulmaktadır. Yani bu uygulama başka bir araştırma için kullanılmak istendiğinde, bu yeni araştırma için belirlenen el işaretlerinin uygulama aracılığı ile doğru bir şekilde toplanabilmesi ve etiketlenebilmesi için, araştırmacının gerekli yeni referans videoları bahsedilen klasöre eklemesi yeterli olacaktır. Uygulamanın yeni amaca uygun çalışabilmesi için herhangi başka bir güncellemeye, özellikle uygulama kodunda yapılacak herhangi bir değişikliğe gerek yoktur.

Bu proje için veri kümesi olarak Türk İşaret Dili'nden 12 dinamik ve iki eli birden içeren kelime işaretleri seçilmiştir. Veri toplama uygulaması geliştirildikten sonra deneylerde girdi olarak kullanılmak üzere birden çok işaretçi tarafından, farklı ortamlarda, işaret başına toplamda 200 adet örnek toplanmıştır. Toplanan bu örnekler ile toplamda 2400 örnek içeren ve işaretler arasında eşit bir dağılıma sahip olan bir veri kümesi oluşturulmuştur. Bu veri kümesi oluşturulduktan sonra öğrenme modelleri tarafından daha iyi kullanılabilirliğini sağlamak amacıyla bazı ön işlemlerden geçirilip veri eğitim işlemi için hazırlanmıştır. Veri kümesindeki örneklerin her biri yüksek miktarda nümerik veri içermektedir. Bu nümerik veriler örnek dosyasına 178 kolon ve veri kaydına göre değişkenlik gösteren satır sayısında kaydedilmiştir. Her bir kolon örneklenen işaretten alınan bir özelliği içermekte her bir satır ise anlık zaman çerçevesini içermektedir. Yani her zaman çerçevesi dahilinde sensörden alınan 178 tane özellik kaydedilmiştir. Her örneğe göre zaman çerçevesi sayısı, her kullanıcının kaydı başlatması ve bitirmesi arasındaki geçen süre farklılık göstereceği için değişkendir. Ancak bu durum verinin işlenmemiş hali ile öğrenme modellerine verilmesine engel olmaktadır çünkü modele verilen örneklerin boyutlarının eşit olması gerekmektedir. Yapılan ön işlemlerden bir kısmı örneklerin hepsinin boyutunu eşitleyerek bu sorunu gidermekte, kalan kısmı ise alınan verinin ve etiketlerin modeller tarafından anlaşılabilir hale getirilmesini sağlayan standart işlemlerden oluşmaktadır.

Projenin ikinci bölümünde 6 farklı geleneksel makine öğrenmesi sınıflandırıcısı: *Gauss Naif Bayes (Gaussian Naïve Bayes)*, *Destek Vektor Makinesi (Support Vector Machine)*, *K-En Yakın Komşu (K-Nearest Neighbour)*, *Rassal Orman (Random Forest)*, *Doğrusal Ayrımcılık Analizi (Linear Discriminant Analysis)* ve *Çok Katmanlı Algılayıcılar (Multilayer Perceptron)*; ve 3 farklı derin öğrenme modeli: *Evrişimsel Sinir Ağları (Convolutional Neural Networks-CNN)*, *Tekrarlayan Yapay Sinir Ağları* (*Recurrent Neural Networks-RNN*) ve *Uzun Kısa Vadeli Hafıza Ağları* (*Long Short-Term Memory-LSTM*) yaratılmış ve yüksek tanıma başarısına ulaşabilmek için bu sınıflandırıcılar ve modeller üzerinde deneyler yapılmıştır.

Bu sınıflandırıcılar ve modeller literatürde benzer araştırmalar değerlendirilerek, aynı zamanda tanınmak istenen verinin yapısı düşünülerek seçilmiştir. Örnek başına düşen veri miktarı oldukça fazla ve karmaşık olduğu için sezgisel yöntemler yerine öğrenmeye dayalı yöntemlerin kullanılması kararlaştırılmıştır. Derin öğrenme yöntemleri arasından seçim yapılırken; *Evrişimsel Sinir Ağları*'nın seçilmesinin

nedeni bu tip sinir ağlarında resim verilerinin daha başarılı sonuç vermesinin göz önünde bulundurulması ve bu projedeki verinin 2 boyutlu oluşu nedeniyle yapısal olarak tek renkli bir resim verisine benzetilmesi, *Tekrarlayan Yapay Sinir Ağları*'nın seçilmesinin nedeni ise projedeki verinin zamansal bilgi içermesi ve bu derin öğrenme yönteminin zamansal boyutta değerlendirme yapabilme yeteneğinin yüksek olmasıdır. *Uzun Kısa Vadeli Hafiza Ağları*'nin seçilme nedeni ise zamansal verinin boyutunun büyük olmasıdır. Bahsedilen yöntemler ışığında modeller *Python* programlama dili ve açık kaynak kodlu bir derin öğrenme kütüphanesi olan *Pytorch* çatısı sınıflandırıcılar ise *sklearn* kütüphanesi kullanılarak oluşturulmuştur. Öğrenme ve tanıma işleminin başarıdan ödün vermeyerek mümkün olduğunca hızlı tutulabilmesi için modeller olabildiğince basit mimariler kullanılarak tasarlanmış ve kullanıma sunulmuş, *Microsoft Azure: Bulut Bilişim Hizmetleri* üzerinde yaratılmış olan ve grafik işlemci birimleri (*Graphics Processing Unit-GPU*) içeren sanal bir makine *(Virtual Machine)* üzerinde yürütülmüştür. Sistemin performansını yükseltmek için oluşturulan modeller üzerinde farklı hiper parametre kombinasyonları farklı kayıp ve iyileştirici fonksiyonlar ile birlikte denenmiştir. Kayıp fonksiyonu olarak *Cross Entropy* fonksiyonu iyileştirici olarak *Stochastic Gradient Decent* ve *Adam* fonksiyonları ile birlikte kullanılmıştır. İyileştirici fonksiyonların farklı kombinasyonları denenen parametreleri öğrenme oranı ve momentumdur.

Deneyler üç bölüm halinde yapılmıştır. İlk olarak geleneksel makine öğrenmesi metodları üzerinde hiper parametreler değiştirilerek farklı eğitimler yapılmış ve sınıflandırıcılar optimize edilmiştir. Bu metodlar arasında en iyi sonucu veren optimize edilen *LDA* sınıflandırıcısı olmuştur. Bu sınıflandırıcı ile %99.9 eğitim, %98.2 doğrulama ve %95 test doğruluk oranlarına erişilmiştir. Deneylerin ikinci bölümünde derin öğrenme metodları üzerinde deneyler yapılmış ve yine hiper parametreler değiştirilerek modeller optimize edilmiştir. Sonuç olarak bu modeller arasında en başarılı olan *CNN* modeli olmuştur. Bu model ile %100 eğitim, %97.7 doğrulama ve %96.7 test doğruluk oranlarına ulaşılmıştır. Deneylerin son bölümü olarak toplanmış olan veri kümesi karıştırılarak 5 farklı eğitim-test kümesi oluşturulmuştur. Oluşturulan 5 farklı veri kümesi üzerinde derin öğrenme modelleri ve *LDA* sınıflandırıcısı 5 kez yürütülmüş ve ortalama sonuçlar hesaplanmıştır. Ulaşılan en iyi sonuçlar; *LDA* sınıflandırıcısı ile %98.8, *CNN* modeli ile %98.8, *RNN* modeli ile %88.3, *LSTM* modeli ile %85.0 test doğruluk oranıdır. Ulaşılan genel ortalama sonuçlar ise *LDA* sınıflandırıcısı ile %97.4, *CNN* modeli ile %96.9, *RNN* modeli ile %82.8, *LSTM* modeli ile %79.2 test doğruluk oranıdır. Elde edilen sonuçlar göz önüne alındığında çalışmanın amacı ve kullanılan veri kümesi için en iyi metodların *LDA* ve *CNN* olduğu görülmüştür. Son bir değerlendirme olarak bu iki metodun test doğruluk oranlarının birbirine oldukça yakın olması nedeniyle hata matrisleri analiz edilmiştir. İşaretlerin bireysel sonuçları göz önüne alındığında *CNN* modelinde doğru işaret tahminlerinin daha homojen olması dolayısıyla *CNN* metodunun *LDA*'den daha başarılı olduğu görülmüştür. Aynı zamanda bu projenin devamı niteliğinde, geliştirilmiş olan model kullanılarak gerçek zamanlı tanımanın yapılabileceği bir uygulamanın geliştirilmesi kararlaştırılmış ve planlanmıştır.

1. INTRODUCTION

1.1 Purpose of Thesis

The main purpose of this thesis is to create a deep learning based dynamic sign language recognition system via hand gestures. In this system *Leap Motion Controller (LMC)* is chosen for gathering hand gesture data. To achieve this purpose also an efficient data collection system is created which is highly adaptable for any data collection purpose with *LMC*. As a future work, this thesis can be extended to an application for real time sign language recognition which can serve to hearing-impaired people as a translator.

This study is a part of *RoboRehab: Assistive Audiology Rehabilitation Robot* project where we aim to develop an effectively aware robotic rehabilitation platform to assist hearing-impaired children in hospitals with their audiology tests. This system can be used by the children as another modality to interact with the humanoid robot Pepper in the project. [1]

1.2 Literature Review

There are several studies on the recognition of Sign Languages using depth sensors such as *Leap Motion Controller* and *Microsoft Kinect*. Traditional machine learning methods, and lately, deep learning based approaches are employed for the recognition process. Even though some studies include time-series data, most of the studies in the literature include static single hand gesture data which consist of one labeled data frame per sign which is similar to the predecessor study [7] [8] of this thesis.

Sign languages are visual languages consist of gestures of face, upper body, arm, hand and fingers. Since *Leap Motion Sensor* can only detect hands, and fingers, it is employed to recognize only the alphabet, or hand signs of the sign languages.

[1] This project is funded under the grant TUBITAK 118E214.

Table 1.1 : Summary of the studies in the literature

Reference	Data Collection	Data Type	Dataset	Approach	Accuracy
[9]	LMC	26 static, single	ASL Alph.	k-NN	72.78%
				SVM	79.83%
[10]	LMC	28 static, single	ArSL Alph.	NBC	98.3%
				MLP	99.1%
[11]	LMC	28 static	ArSL Alph. + Digits	SVM	91%
[12]	LMC	31 static, double	ISL Alph.	Eucl. Dist.	88.39%
				Cos Sim.	90.32%
[13]	LMC	24 static	ASL Alph.	heu. DT + GA	82.71%
[14]	LMC	28 single + 3D drawn	ISL + Latin	SVM + BLSTM-NN	63.57%
[15]	LMC		ASL	RNN	over 96%
[16]	LMC	-	ISL	DTW + IS Algo.	-
[17]	LMC	10 Gestures	-	ANFIS and SVM algo.	98.66%
[18]	LMC	3D dynamic		SVM-LSTM based hybrid NN	96.4%
[19]	LMC	6 Dynamic Gestures	-	LSTM + CNN	95.9%
[20]	LMC	3D drawn	-	HMM	86.88%
				BLSTM-NN	81.25%
[21]	LMC	recog. and ver. 3D signatures	-	k-NN	2D to 3D = +6.8% (recog.) +9.5% (verif.)
				HMM	2D to 3D = +9.9% (recog.) +6.5% (verif.)
[22]	2 LMCs	28	ArSL Alph.	LDA-fusion at FE	97.7%
				LDA- fusion at Class.	97.1%
[23]	LMCs	0-9 digit Gestures	ASL	HMM	93.14%
[24]	LMC + Kinect	28 static	ArSL Alph.	Supervised ML	22/28 => 100%
[25]	LMC + Kinect	10 static	ASL Alph.	multi-class SVM	91.28%
[26]	LMC + Kinect	10 gesture	ASL	SVM	96.5%
				1 expl. RF	94.7%
[27]	LMC + Kinect	51 dynamic single + double	ISL	HMM	96.05% single - 94.27% double
				HMM + IBC	97.89% single - 96.87% double
[28]	LMC + Kinect	50 single + double	ISL	HMM + BLSTM-NN	97.85% (single) + 94.55% (double)
[29]	Kinect	5, 100 sent. continuous	CSL	LB-HMM	86.6%
				LB-Fast-HMM	87.8%
		5, 100 sent. isolated	CSL	HMM	86.6%
				DTW	51% + 28.5% = 79.5%
[30]	Kinect	9 Gestures	-	SVM	95.8 %
				K-nn	79.5%
[31]	Camera	30 isolated	ArSL	Eucl. Dist. Class.	97% + new occlusion resolving = +2.56%
[32]	Camera	15, isolated, double, video	ASL -> RWTH-BOSTON-50	median + mode + serial particle filt.	87.33%
[33]	Camera	Continuous Sentences	ASL -> SIGNUM	Statistical Approach + HMM	90.0% single signer 47.0% multi-signer
			ASL -> RWTH-PHOENIX-Weather	Statistical Approach + HMM	65.7% single-signer 47.0% multi-signer
[34]	Wearable Devices	9 Dynamic Gestures	Cambridge Gesture DS	CNN + RNN	77.21%
	Sony SmartWatch	20 Dynamic Gestures	SmartWatch Gestures DS	LSTM	88.57%
[35]	SoftKinect DS325 + DUO 3D	25 Dynamic Gestures		R-3D-CNN	83.8%
[36]	Flex	Dynamic Gestures	4-dimensional signals	LSTM	
[37]		Dynamic Gestures		NIUKF-LSTM	

There are several studies which are recognizing the static sign language handshapes via *LMC* on various different sign languages. In the study [9], 26 static alphabet signs of American Sign Language are recognized with *Leap Motion Controller*. The signs are single hand gestures and two different machine learning methods such as *k-Nearest Neighbor(k-NN)* and *Support Vector Machine(SVM)* are employed to recognize this data. For both methods 4-fold cross validation is used and the success rate of this study is reflected as 72.78% for *k-NN* and 79.83% for *SVM*. As another sign language, 28 static alphabet signs of Arabic Sign Language (*ArSL*) are recognized with *Leap Motion* in another study [10]. These signs are single hand gestures, too. Two different machine learning methods, *Naïve Based Classifier(NBC)* and *Multilayer Perceptron (MLP)* for the classification of the data are used and the success rates reflected are 98.3% for *NBC* and 99.1% for *MLP*. There is also another study [11] on *ArSL* which proposes a pattern recognition based *SVM* method for the recognition of the signs which are gathered with *LMC*. Total of 28 static hand gestures are used from the alphabet and 0-9 digit signs to evaluate the approach. 91% accuracy of classification rate is reached as the result of the study. Similarly Indian Sign Language is studied in [12] where a sign language recognition system is developed using *Leap Motion Controller* on a 10 degree inclined surface, to extract the depth information properly. As the data, 26 alphabet signs and 5 digit signs of Indian Sign Language, which includes both hands, is chosen to be recognized and this data is collected from 10 different signers. Two different methods are used for the recognition; with *Euclidean Distance Method* 88.39% average accuracy is reached while 90.32% average accuracy is reached with the *Cosine Similarity Method*. Apart from the supervised learning methods, there are also heuristic approaches present in the literature like [13]. In this particular study, a heuristic decision tree is implemented to recognize the American Sign Language data acquired via *Leap Motion Controller*. The data includes 24 static alphabet finger spelling signs. The decision tree includes 16 kinds of decisions which focus hand and finger characteristics. To sort these decisions genetic algorithms are experimented to achieve the most efficient order for the recognition. As the result the achieved, accuracy is pointed out as 82.71%.

Apart from studies which focus on static hand shapes, there are a couple of studies on dynamic gestures of sign languages. In [14], 56 different annotations are included

in the recognition process which are gathered with *LMC* and are separated into two groups; 28 of them are single-handed-isolated sign gestures of Indian Sign Language and remaining 28 are Latin Language fingerspelling words (air writing). *Support Vector Machine(SVM)* classifier is used as the first step of the recognition to classify the signs according to these two groups. After the separation is done, two *Bidirectional Long Short-Term Memory Neural Networks(BLSTM-NN)* classifiers are used for the recognition. These two networks have different approaches per group; the approach is sequenced classification for the manual signs and sequence transcription for the fingerspelling. In real-time recognition 63.57% accuracy is reached in this study. In a similar study, [15] the hand gestures chosen from American Sign Language(*ASL*) are recognized by an *RNN* which is trained with the data of finger bone angles acquired via *Leap Motion Controller*. The recognition accuracy of the system is reflected as over 96%. The method is also used with *SHREC* dataset, which includes semaphoric hand gestures, where it is stated that the method outperformed compared with the current approaches in the literature. In another one, a system, which combines *DTW* with IS algorithm is proposed to convert the hand gestures of Indian Sign Language (*ISL*) gathered with *Leap Motion Controller* device into text [16]. *IS* algorithm is used to detect the dynamic environment changes and DTW is used for the gesture transformation with the map of the similar patterns.

While there are just a little amount of studies in the literature which are focusing on dynamic sign language gestures using *LMC*, there are a lot of studies for the recognition of general dynamic hand gestures, like waving or clasping which are also gather data via *LMC*. In [17], a robot control system is developed which is based on these kinds of 10 hand gestures. The data for the hand gestures are acquired via *Leap Motion Controller*, with using noise suppression, coordinate transformations and inverse kinematics. For the classification of the gestures *ANFIS* and *SVM* algorithms are implemented in the system. The recognition accuracy of the approach is given per sign in the study which reaches to 98.66% in average. In another study [18], including similar dataset, a 3D dynamic gesture interaction system is developed by using *SVM-LSTM* based hybrid neural network model. The gesture data is captured by *Leap Motion*. The start and end points of the gestures are automatically defined by the model and 96.4% accuracy is reached by the system. It is also reflected that

the prediction of the gestures in the system is taking 0.15 seconds in average. Also in [19] 3D hand positions and velocities gathered from *LMC* are used to recognize 6 types of dynamic hand gestures with a *Long Short-Term Memory* and *Convolutional Neural Networks* based techniques. In the real time experiments, 97% F-measure is achieved and the recognition accuracy is found as 95.9%. As another set of used hand gestures, which do not belong to sign languages, some of the studies are focused on finger drawing kind of dataset which are specifically using the motion of the hand rather than the hand shape. In [20] 3D finger drawn text gathered with *LMC* is recognized. The data is gathered as sentences and a heuristic analysis conducted for the word segmentation by paying attention to the stroke lengths between the words. In the segmentation part 78.2% accuracy is achieved. As the continuation of the study, sequential classifiers are used for the recognition. The achieved accuracy became 86.88% with *Hidden Markov Model (HMM)* classifier and 81.25% with *Bidirectional Long Short-Term Memory Neural Networks (BLSTM-NNs)*. As another study which uses similar data, a system to recognize and verify 3D signatures is proposed in [21]. The 2D features gathered from *Leap Motion* is extended to 3D with using instantaneous pressure of the writing. A dataset with 2000 signatures is created with these features. The recognition is done by applying *k-NN* and *HMM*. With the 3D dataset higher accuracies are recorded for recognition and verification via both classifying methods. With *k-NN* 6.8% and with *HMM* 9.9% accuracy increases are recorded for recognition while the increases are 9.5% and 6.5% for verification respectively. Also same methods are also experimented with benchmark datasets and similar results are recorded. It is concluded that this new addition to the data can improve the accuracy of the biometric systems and for the existing biometric setups, *Leap Motion* can be an alternative.

In some studies multiple sensors are used to increase the success of the recognition. This is due to the fact that, sometimes one hand or fingers can be occluded by other hand and the signs cannot be recognized correctly. Multiple sensors can be employed from different angles and the data could be fused to increase the recognition success rate. In some of these studies, multiple *LMC*s are used to recognize static handshapes of sign languages. As an example to this category in [22], 28 Arabic Alphabet Signs are recognized using the data collected by using two *LMC*s. The data coming from these sensors are fused either in the feature extraction or the classification phase where,

Linear Discriminant Analysis classifier is employed. The accuracy reflected as 97.7% when the fusion is made at the feature extraction level and as 97.1% when the fusion is made at classifier level. In the study it is also pointed out that using two *LMC*s resulted with a better performance, then using a single one. In another study [23], also multiple *Leap Motion Controller*'s are used for the recognition of American Sign Language's 0-9 digit signs. The data coming from both the devices are fused together and *Hidden Markov Models (HMMs)* are used for the recognition. As the result of the experiments, it is pointed out that using multiple sensors can result with higher recognition accuracy than the systems which use single sensor. The accuracy achieved with multiple sensors is 93.14% in average.

Rather than using multiple *LMC*s some of the studies use various different sensors together to include different kinds of features into the dataset. The most commonly used sensors are *Leap Motion Controller* together with *Microsoft's Kinect* sensor. These specific sensors are used in [24] while a supervised machine learning model is developed to recognize Arabic Sign Language's (*ArSL*) hand gestures. The sensors are used together to collect depth images from 28 static alphabet signs of the sign language, that are used for the evaluation of the model. In the study recognition accuracy of 22 out of 28 signs reached to 100%. In a similar study, multi-class *SVM* classifier is used to recognize 10 hand gestures of American Manual Alphabet by using *Leap Motion Controller* together with *Kinect* sensor [25]. The fingertip position and orientation data coming from *Leap Motion* is combined with the depth information coming from *Kinect* sensor and the combined data is used to create an ad hoc feature set. With the proposed approach the achieved recognition accuracy is reflected as 91.28%. Apart from a couple of studies which use these sensors on static hand shapes, there are more studies which used this pair for dynamic gestures. For example, in [26], *Leap Motion Controller* is used together with *Microsoft Kinect* sensor jointly to achieve a real-time recognition system. By adding *Microsoft Kinect* sensor to the study, additional features are collected as hand contours and distance between hand samples from the centroid. In this study 10 American Sign Language gestures are recognized by using multi-class *Support Vector Machine* classifier and one exploiting *Random Forest* classifier together with 3 different feature selection methods; *F-Score*, *Sequential* and *Random Forests*. The achieved accuracy is 96.5% for *SVM* and 94.7%

for *Random Forest* classifier. In another study [27] the data coming from *Leap Motion Controller* is combined with the facial data of the signer which is gathered with *Kinect* simultaneously. 51 dynamic sign word gestures (31 double hand - 20 single hand) of Indian Sign Language are collected as the dataset and the recognition is done with *Hidden Markov Model (HMM)*. After the *HMM*, *Independent Bayesian Classification Combination* approach is used to improve the performance of recognition. For single hand gestures recognition rate is reflected as 96.05% and for double hand gestures, the rate reflected is 94.27%. The study is concluded with a comparison of unimodal and multimodal network where multimodal network's rates show gains of 1.84% for single and 2.60% for double hand gestures. The same pair of sensors are also used in [28] to gather Indian Sign Language data to be recognized by implementation of two different sequential classifier models namely; *Hidden Markov Model (HMM)* and *Bidirectional Long Short-Term Memory Neural Network (BLSTM-NN)*. The data included 7500 total samples of 50 different sign-word gestures, which include both single and double-handed signs. To improve the recognition accuracies both *HMM* and *BLSTM-NN* results are combined and 97.85% for single and 94.55% for double hand overall accuracies are reached.

In some of the studies, it is preferred to use just one *Kinect* sensor rather than using its combination with a *LMC* like in [29] where a fast and less complex algorithm based on *Hidden Markov Model (HMM)* is proposed to calculate the similarity between the sign and the sign sequence . Also to improve the accuracy of the recognition, grammar and sign length constraints are added together with a proposed coarse segmentation method to the system. As the data, a *Kinect* dataset of 100 different sentences of Chinese Sign Language which are composed from 5 signs are used. For the continuous (sentences) sign recognition, with the *LB-HMM* approach the recognition accuracy found as 86.6% while it raised to 87.8% with *LB-Fast-HMM*. For the isolated sign recognition *HMM* and *DTW* are experimented on where the accuracies found as 97.8% and 95.1% respectively. In a related study [30] *Support Vector Machine* and *K-Nearest Neighbour Classifier* is used to recognize 9 hand gestures which are not chosen particularly from any sign language. The features used in the recognition is handcrafted by using the *Kinect*'s Skeleton Data's simulated signatures. Also data augmentation is done to create synthetic samples by using synthetic minority oversampling technique. The

achieved classification rate is 71.1% for the *SVM* and 51% for the *k-Nearest Neighbour*. Also to improve the accuracy action pair based one vs. one classification layer weights are used and the improvement is seen as 24.7% and 28.5% respectively.

As an alternative to depth sensors such as *LMC* and *Kinect*, regular *RGB* cameras are also used in several studies in the literature. As it is used in [31] 30 isolated words from Arabic Sign Language are used to evaluate the proposed Automatic Sign Language Recognition System. The sign language data is gathered with a single camera. 83% of these words were having the different occlusion states. There were 4 stages in the study; hand segmentation, tracking, feature extraction and classification. A skin detector is used for the hand segmentation and a proposed skin-blob tracking technique is used for tracking. In the experiments with *Euclidean Distance Classifier* 97% recognition rate is achieved which is independent of the signer. Also to improve the recognition of the similar gestures an occlusion resolving technique is proposed which requires specification of the position of the hands and the head but also results with 2.56% improvement. In some other studies, the public databases are used for the experiments which consist of the same kind of data gathered with camera. As an example in a study [32], a methodology for hand tracking and feature extraction is proposed for the recognition of sign gestures which are sampled from the American Sign Language. 15 signs (black-white videos) which are simulated with both hands are chosen from *RWTH-BOSTON-50* dataset as the evaluation data for the proposed approach. The method showed 87.33% recognition accuracy in the experiments conducted. The methodology includes feature covariance matrix based serial particle filter for the isolated hand gestures. Before the feature matrix is constructed for the detection, median and mode filters are combined to extract the foreground for better recognition. As an addition to the proposal, to reduce tracking errors, serial tracking of the hands are suggested rather than parallel tracking. By combining these two proposals, the region around the hands is extracted to create the covariance matrix which represents the hand position, and as a result decreases the number of features compared to the original data. Also it is reflected that the feature matrix is adaptable to new signs without any retraining process because it is able to integrate multiple correlated features. In a similar study [33], a statistical approach is presented for continuous sign language recognition with changing signers. It is

reflected that it is important to use both landmarks from hand and face is important for the recognition of the sign languages. *CMLLR* adaptation and class language models are used to improve the recognition. Two different datasets are used to evaluate the proposed system. As the lab data, *SIGNUM* database is used (Gathered with camera, 25 signers, 455 sign vocabulary, 19k sentences) and as the 'real-life' data, *RWTH-PHOENIX-Weather* database is used (Gathered with camera, 9 signers, 1081 sign vocabulary, 7k sentences). While evaluating the approach for the recognition method, *HMM* is used and with the lab-data data 10.0%/16.4% word error rates are achieved and with the 'real-life' data 34.3%/53.0% word error rates are achieved for single signer/multi-signer setups.

There are also some other studies which are using various different sensors to recognize general dynamic hand gestures which are not based on sign languages. In such a study [34] two recognition techniques based on *Recurrent Neural Network (RNN)* are developed to recognize dynamic hand gestures which are from *Cambridge Gesture Dataset* and *SmartWatch Gestures Dataset*. In *Cambridge Gesture Dataset* there are 9 classes which include 3 different handshapes combined with 3 different movements. For the recognition of this dataset *Convolutional Neural Network (CNN)* is combined with *RNN*. The data in *SmartWatch Gestures Dataset* includes 20 arm gestures which are gathered with *Sony SmartWarch* accelerometer and for the recognition *LSTM* technique is used. An optimization is made to decrease the required power in the implementation, considering both hardware and software, a fixed-point optimization is used to quantize the weights into two bits which results with decrease in the needed storage size for the weights. With combined neural network approach (*CNN + RNN*) the accuracy is shown as 77.31% and with the *RNN* approach the accuracy is shown as 88.57%. In another study, 25 dynamic hand gestures are captured with depth, color and stereo-IR sensors to be recognized by recurrent three-dimensional *Convolutional Neural Network* [35]. The depth and color information is gathered with *SoftKinect DS325* sensor and stereo-IR data is gathered with *DUO 3D* sensor. The proposed recognition system achieved 83.8% recognition accuracy where it is pointed out that the human accuracy is 88.4% for the same dataset. It is also reflected that this performance achieves state-of-the-art performance on *SKIG* and *ChaLearn2014* benchmarks. As an example of using completely different sensor

device, which is also presented in the study, in [36], *Flex* sensor is used, which is an armband including four flex resistance sensors to collect data from forearm muscles. The device provides 4-dimensional signals from the muscles which can be used to recognize the gestures of the hand it is worn on. Also with the data including 4 different gestures and additional noise gestures as plus 1 gathered via *Flex*, a *Long Short-Term Memory Network (LSTM)* is trained for the real-time recognition of the hand gestures and the classification accuracy is achieved as 93.4%. In a similar study [37], a nested interval *Unscented Kalman Filter(UKF)* with *Long Short-Term Memory(NIUKF-LSTM)* network is proposed for the hand joints-based gesture recognition systems which use sequential skeletal datasets. This kind of datasets includes joint identification using hand pose estimations which are generally noisy and error-prone. The proposed method changes the distribution of the sigma points between two intervals which enable the system to revise the noise and improves the accuracy of the recognition.

Apart from the studies which are experimenting on the kinds of hand and/or hand movement recognition, there are also a lot of studies which are reviewing the *LMC* according to its sensing abilities for gathering hand data, if it is rather sufficient to recognize sign language gestures or not. A review of the *Leap Motion Controller*, and the analysis of its abilities for the recognition of Australian Sign Language (*Auslan*) are presented in [38]. It is reflected that the device is able to track the hands and fingers accurately while the accuracy decreases for some particular cases. These cases are pointed out as; when the position of the hands obstructs the device's view and when the elements of the hands, like fingers, are brought together. It is concluded that the device has potential but further development in the *Leap Motion API* is needed to be used for the recognition of *Auslan*.

All the studies that are examined above are especially focused on the recognition but how the data is collected and annotated, which is a crucial part of these studies is not covered and reported in detail. Importance of the data collection and annotation process can be considered from the different aspects. For the approaches based on machine learning methods, especially deep learning models, a huge amount of data is needed to achieve the results with acceptably high success rates, where generally, more data leads to better recognition. As another aspect, because large amount of

data is needed, collection of the data and labeling processes are very time consuming. Especially if the method or tool used for the data collection is not easy to use and is not efficient, this causes additional problems. Lack of efficiency may also lead to spending more time in the pre-processing step of the data. To overcome all of the difficulties on these aspects, the interface which is used in the process should be easy and it should include demo videos for the participants who will generate and the annotate the data, since they might not be fluent in the sign language(s). The demo videos are also necessary for the possible conflicts due to different sayings, accents of the same sign. Unfortunately, there are just a few studies which are especially focused on the data collection and annotation of the sign languages with *Leap Motion Controller* device. In a related study [39], a data collection tool is created to gather data via *Microsoft Kinect 2* together with *Leap Motion Controller* by synchronizing them together. In the study the data is collected as video and data tables, by using *C++* framework, and the official *SDK*s of *Microsoft Kinect 2* and *Leap Motion API*s. The tool also includes a command-line interface, and a *Matlab GUI* to initiate, inspect, and load *Kinect 2* recordings. This *GUI* doesn't include data gathering from LMC.

1.3 Hypothesis

There are several research questions related to the thesis, and a hypothesis build upon these questions. We collected dynamic sign data (hand gestures) using the efficient and broadly usable application that we designed and developed, and tested several approaches including traditional machine learning methods and deep learning based methods (*CNN*, *RNN* and *LSTM*) to see which approach is more successful in recognizing these dynamic hand gestures. We come up with the following hypothesis based on these studies:

Hypothesis 1 (H1): The *CNN* based approach will show higher success rate then *RNN* and LSTM approaches in the recognition of the selected set of dynamic signs from *TID* (the 2-dimensional labelled time-series data of Turkish Sign Language's dynamic hand signs which is collected with this application).

Hypothesis 2 (H2): The deep learning based methods will show a higher success rate than traditional machine learning methods, in the recognition of the selected set of

dynamic signs from *TID* (the 2-dimensional labeled time series data of Turkish Sign Language's dynamic hand signs which is collected with this application).

2. DATA ACQUISITION

2.1 Motivation

The data acquisition is the first step of the study, which is the most time-consuming part and very important for the success of the study. This part should be efficient in terms of resource consumption, modular and adaptable for the collection of similar data types, for broader usage. To achieve this aim, the *Leap Motion Controller* is chosen as the sensor, and *Windows Forms Application* is chosen for the graphical user interface library to be used for the development process of the data acquisition tool.

2.1.1 Leap Motion Controller (LMC)

Leap Motion Controller is a small and portable commercial device which has two infrared cameras and three *LED*s in its hardware. It is connected to the computer from a *USB* port and it has its own *SDK* which recognizes hands and hand movements and gives the directional and positional vector data for hands it senses, after it makes uncompromised calculations on the raw data coming from its cameras. It also has some default applications as well as the basic utilities (like pause, reset, calibrate, troubleshoot the device) supplied by its driver which helps to visualize the observed data. The most important of these applications is the *Visualizer* which can be started from the dock icon of the driver. The *Visualizer* is a screen that shows a coordinate template including the real time skeleton and the data points of the hands which are recognized by the device.

In the most recent official updates of the *SDK* of the controller, the main application area is mostly shifted to the virtual reality applications, where the device can also be combined with virtual reality equipment like *Oculus Rift* and *HTC Vive*. On the other hand, the older official *SDK*'s are in distribution for the applications which don't consist of virtual reality, even though these *SDK*s are not supported anymore.

2.1.2 Windows Forms Application

Windows Forms Application is a *Graphical User Interface (GUI)* library which belongs to *.NET Framework*. It is easy to use to create applications for *Windows* desktop computers. The design and coding phases can be done by *Visual Studio Application*. The development of the application can be done either by the toolbox supplied by *Visual Studio*, or by *C#* language. In either case data binding to the interface can be done by coding. The most important thing to consider is the thread safety of the application. The *GUI* thread has to differ from the other possible calculations or any other operations to make the *GUI* always responsive for the user.

2.2 Design of the Application

The application is developed by using *Leap Motion SDK v2* which is published officially by *Leap Motion Company*, and *C#* is used as the programming language. Additionally *Windows Forms Application* libraries are used for *GUI* in combination with *Windows Media Player* libraries and *NuGet* packages from *Accord.Net Framework*. *Windows Media Player* libraries are used to display the reference sign videos and the *Accord* libraries are used for obtaining and displaying the sequence of image frames (as video) from the *Webcam*.

The screenshot of the application can be seen on the upper part of the Fig: 2.1. As it is seen in the application screen, the design of the application interface is divided into 3 vertical parts. These parts are ordered from left to right by mostly considering the workflow of the usage. The leftmost part is the first part that user has to interact with the application. From this part, user chooses which sign she/he wants to demonstrate and watches the sign she/he chose from the list. After the video is finished, the user pushes the start button and then her/his attention is directed to the *Webcam* screening which is the next vertical part of the interface, as well as to the text box on the right most side of the application which includes information about the recording states which is updated along the process. Also the start and stop buttons' sizes kept relatively large to increase the ease of the usage while their colors are chosen as green for start and red for stop which are commonly accepted colors for these specific behaviors.

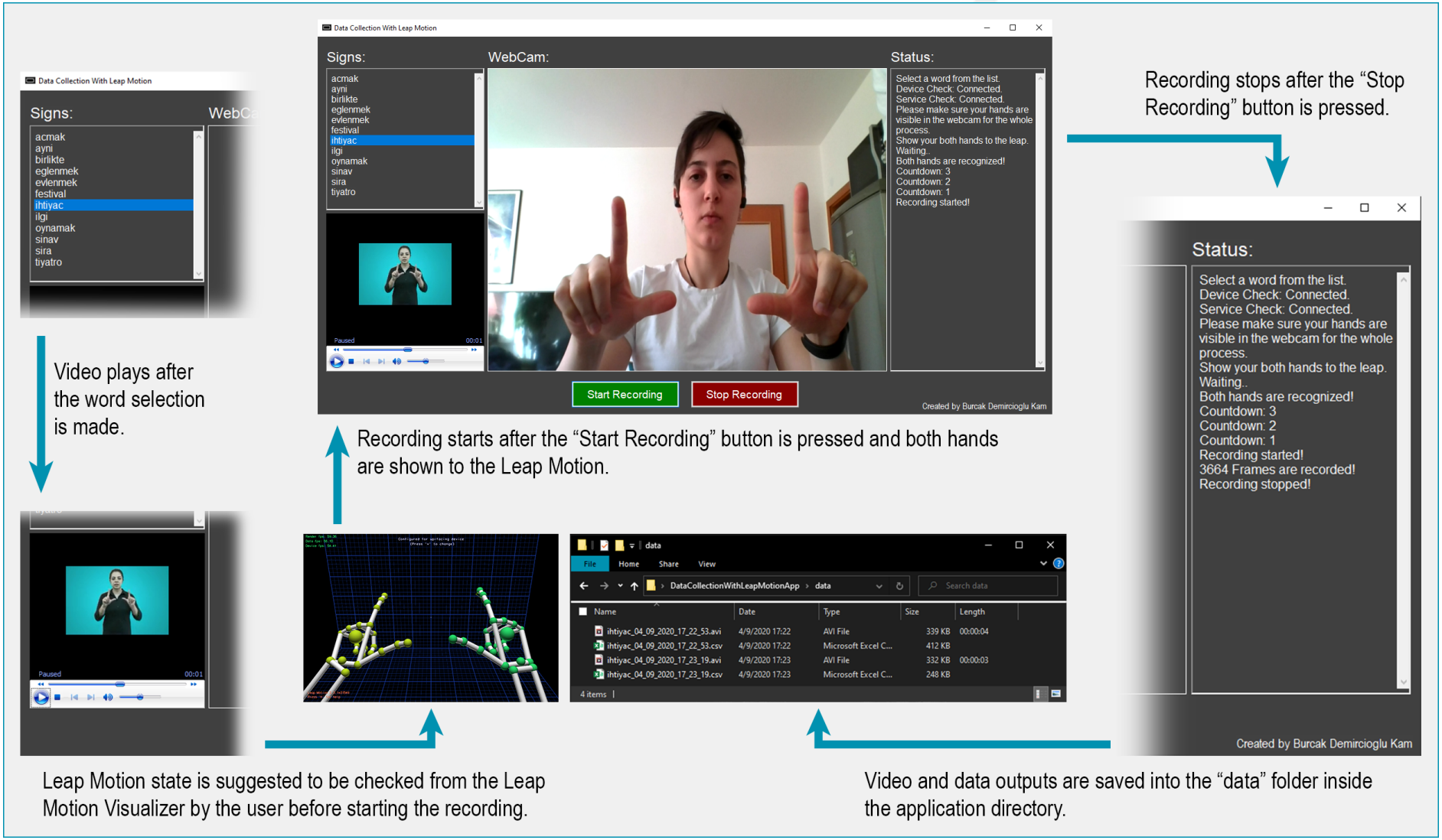

Figure 2.1 : Usage workflow of the data collection application.

2.3 Usage of the Application

To use the application, the only external requirement is a *Leap Motion Controller* with its driver installed on a *Windows* computer. The first step to use the application is to choose the sign to be simulated from the word list, as seen on upper left side of the Fig: 2.1. After a sign is chosen, as shown in Fig: 2.1, as the next step, the sign's simulation example video plays on the player at the left bottom side of the application, to show the user how to generate the sign. As an optional step, to be sure that if the *LMC* is working and able to provide the data to the application, the *Visualizer* application can be used which is included in the *Leap Motion*'s driver. After watching the video, the user has to press the "Start Recording" button to start the data collection. The application checks for the controller and driver service's status and also shows the real time *Webcam* video output as the reaction to the button press. If the checks result with successful connections then the application waits for the user to show her/his both hands to the controller to make the data collection's start point more consistent for every simulation. After both hands are shown by the user, as in the Fig: 2.2, and recognized by the device, a count down starts in the text box located right side of the application. When countdown reaches to 0 then the recording starts for both data file and reference video file. To start the recording, the application waits for the user to press "Stop recording" button. After the button is pressed, the recording stops and the application shows the data frame amount recorded to the data file in the text box, which can be seen on the right side of the Fig: 2.1. This process can be done from start to end repeatedly as much as the user desires until she/he explicitly presses the window close button of the application. After each recording process, the recorded video and data file are saved under the "data" folder. The example local "data" folder screenshot can be seen in the Fig: 2.1 as the last step of the flow.

2.4 Collected Data

Two types of data are collected with this application: vector data coming from the *Leap Motion Controller* and video reference data. Both files are saved under the "data" folder which is in the same location with the application's executable file after the

Figure 2.2 : An example setup to use the application.

recording is terminated. Also both files are labeled with the sign name automatically which is taken from the chosen sign from the list in the application which can be seen in the left side of the Fig: 2.1. The label is included in the name of the files together with the timestamp taken at the instance when the recording is started.

The data coming from *LMC* is saved as a *.csv* file. It consists of 178 features which can be seen in the Appendix A. These features are the columns of the data file. The features include various directional and positional vectors for palms and fingers as well as other specific *boolean* data related to the positions of the limbs like if the fingers are extended or not. While recording the data floating data points are recorded without any change while the *boolean* data converted to binary to make it more efficient to use in the further usage. In some edge cases, it is observed that, sometimes *LMC* is not recognizing both hands and it just sends the data of one hand. To overcome this inconsistency, the second hand's expected data is filled with 0s (*NULL* value) while recording the data into the file. With this padding strategy, the data size kept consistent even though the data coming from *LMC* is not consistent.

2.4.1 Sign Recognition

The dataset used in the evaluation of the system includes 12 word signs of Turkish Sign Language whose example videos are collected from *TID*'s online dictionary [40]. While choosing these specific signs, the results from the previous studies [7] [8] are

adopted. The most successfully recognized 4 different hand shapes are chosen based on the adopted information and then for each one, 3 different signs (words) are selected, which include different movements of the chosen hand shape. The dataset is limited with just 4 hand shapes because in this study, main purpose is to recognize the hand shape with its movement rather than just the hand shape itself. Therefore all the signs in the dataset are dynamic signs which include multiple time frames. While the recognition of the static hand shapes are experimented in the previous studies, it is realized that for the full recognition of any sign language the static recognition is not enough because every hand shape may result with multiple different meanings with its different movements. Therefore, this study is more challenging and complex compared to the previous studies.

On the other hand, compared with static hand shape data recognition of dynamic signs are significantly harder. In the dynamic dataset, the input size of one sample is significantly higher than static signs, because of the included time frames. In the dynamic dataset, the number of features per time frame, per sample is equal to the number of features per sample in the static dataset. This means a static sample is just a snapshot of a dynamic sample at a particular instance and in dynamic datasets, to find the actual input size to the learning model per one sample, the number of features has to be multiplied by the amount of the time frame in the specific sample while the input size for the static dataset sample remains equal to the number of features per sign. Therefore the system has to recognize the sign's movements as well as the static hand shape it includes in each time frame.

The created dataset for the study includes only features which are collected from both hands, and excludes the features from any other body parts like face or torso. This may seem as a disadvantage of this study because with the other body parts' position and movements, the meanings may change in the sign languages but to keep the system efficient, easy and commercially usable by the end user, the best sensor to use is the *LMC* which unfortunately only collects the data from hands. The sensors which can collect data from other body parts are all either bulky and can not be integrated to the daily usage like *Kinect* or just gives very raw data (like just image or video) which is very hard to extract the specific finger information from like standard *RGB* cameras.

3. DATA PREPROCESSING

The sign/gesture data collected by the data collection tool explained in the chapter 2 is processed and fed to the neural network models. This processing step is required because the frame sizes are not same in all the data even though the number of features are the same. Therefore as the frames are forming the rows and the features are forming the columns, all the data has consistent column size but varying row size which is not acceptable by the neural networks in most of the cases. To overcome this situation a padding operation is done to fix the row size to the maximum, acquired from the whole data.

$$y = 10^{-(\lceil |log(\min x \in |data|)| \rceil + 1)} \tag{3.1}$$

To minimize the effects of the padding, some precautions are taken. For the padding a very small value which is close to zero and does not exist in the data set is used, not to interfere with the real data values. For this processing, first the real data point closest to the 0 is found in all of the data, after the absolute value of this number is achieved, the logarithm of this number is taken with base 10. Then the absolute value and ceil of this result is taken in order. Then the reached result is increased by 1 to create a safety margin. The result of this calculation is chosen to be the necessary negative power of 10 to create the required smallest number. These calculations can be seen in Eq: 3.1. After this number is found, the data is padded by filling the missing rows with this number until all the sample lengths (row amounts) reached to the maximum. With these processes the data is prepared to be used as the input for the neural networks with a fixed and consistent size and also the possible negative effects of the alterations are minimized.

Although there are a lot of different padding approaches in the literature, like interpolation, getting average size, etc. previously mentioned effective and simple approach is taken because it doesn't need high computing power and it is a lot faster than most of the others. An approach with a need for low computing power is necessary

because the ultimate goal of this study is to develop a real-time sign language translator. This translator has to be usable on also mobile devices like phones which don't have high computer power generally and speed of the prediction is very important to make the product easily usable as natural speaking.

As the additional processing the data is divided into two parts as train and test. Test percentage is chosen to be 10% including same amount of sample for each label in the dataset. While dividing the data, the samples are shuffled. This process is done to keep the test set absolutely separate then the rest of the dataset to keep the test samples same for each trained model and each trial to achieve maximum precision on the comparisons.

4. MACHINE LEARNING BASED SIGN RECOGNITION MODULE

4.1 Motivation

In the previous studies, traditional machine learning approaches are designed and tested for static hand signs [7] [8]. In the first trials of this work, similar approaches are tested with the new data and compared with the deep learning models.

4.1.1 Gaussian Naïve Bayes Classifiers

Naïve Bayes approach takes the Bayes's Theorem as the base of its classification algorithms. The most important characteristic of this classifier is its assumptions on the features. The classifier evaluates each feature independent from each other and gives equal importance to every single of them which doesn't actually reflect the truth in the real life where most of the features are connected to each other for an event to happen. This classifier evaluates the features and concludes with the results by calculating the probability of the results (posterior probabilities) based on the known data just like in the Bayes's Theorem on Eq: 4.1 where it shows the condition for the probability of event A to happen where event B is already happened. In this generalization event A can be seen as the label and event B can be seen as the feature vector. To reach the result, the classifier calculates probabilities of each possible label with this formula where P(B|A) is the conditional probability of the features, also called as likelihood, while P(A) and P(B) are the prior probabilities. Then the classifier decides on the label which has the highest probability as the result output. This approach is useful when the data is discrete because it actually counts the feature values while calculating the P(B|A) (conditional probability) which is not the case in this study.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.1}$$

To make this approach work for continues values in features, an additional assumption is made which constructs the Gaussian Naïve Bayes Classifiers. In this approach the continuous feature values are assumed to be distributed according to a Gaussian

distribution and the probability equation of this distribution is used while calculating the conditional probabilities. The probability equation can be seen in Eq: 4.2, where σ is standard deviation and μ is the mean of the distribution.

$$P(B|A) \Longrightarrow P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i-\mu_y)^2}{2\sigma_y^2}\right) \tag{4.2}$$

4.1.2 Support Vector Machine (SVM)

For Support Vector Machine (SVM) approach (also the classifier trained using this approach is referred as Support Vector Classifier, SVC) the objective is to find a hyperplane in the N-dimensional space, where N equals to the feature amount, which can classify the labels distinctly. While finding the optimal hyperplane there are a lot of possibilities but the one which has a maximum margin between the data points are chosen as the optimal. These chosen hyperplanes are used as the decision boundaries while making predictions on the labels. As some simple examples; when there are two features the hyperplane can be shown as a line (1D) where if there are three features the hyperplane becomes a plane (2D) rather than a line. Therefore more broadly it can be said that when the feature amount is N, then the hyperplane dimension becomes N-1. While training this algorithm hinge loss is used to optimize the algorithm while maximizing the hyperplane margin. When the predicted value is true then no loss is calculated in this approach but when the predicted value is false then the loss is calculated and a regularization parameter is added to balance the loss with the margin maximization goal. After the loss is calculated the back propagation is done to train the model. While training this model various kernel functions can be used for analysis of the patterns in the data. The kernel options are Radial Basis Function (RBF) Kernel, Polynomial (Poly) Kernel, Linear Kernel and LinearSVC Kernel. The difference between these kernels can be seen in Fig: 4.1.

4.2 K-Nearest Neighbours (k-NN)

K-Nearest Neighbours classifier has one of the most basic algorithms among the machine learning approaches. The algorithm doesn't make any analysis on the distribution of the features. The approach is purely standing on the Euclidean distance calculations (Eq: 4.3). When the classifier is predicting a label for a data it calculates

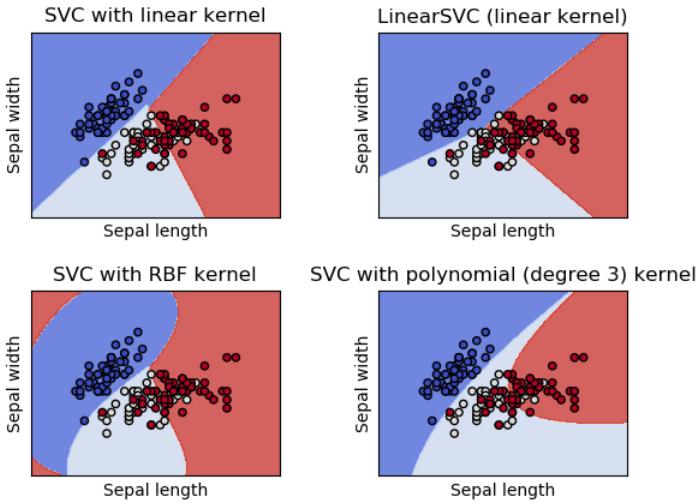

Figure 4.1 : SVC Graphs trained with different kernels. [1]

the distances between the known points (q) and the current data point (p) by using the features (i) as the dimensions. After doing the calculations, it takes k amount of known points which are nearest to the current point and returns the majority label of these nearest points as the decided label for the current data.

$$distance(p,q) = distance(q,p) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{4.3}$$

4.3 Random Forest

Random Forest Classifiers are based on multiple decision trees which are constructed with the features. A sample decision tree can be seen in Fig: 4.2. While prediction is made the algorithm goes through all of the constructed decision trees, gets individual predictions from each of them and uses the one with the most voted. The key point of this algorithm is the low correlation between the models(trees) because when their correlation is low then they are protecting them from each other's errors. While creating the trees to keep the correlation low, the algorithm follows two approaches. The first approach to reach the low correlation is sampling the dataset randomly with replacement while creating the trees. While this sampling the sample size stays same but the content of the sample set changes because of the replacement. This approach makes the trees significantly different because the decision trees are very sensitive to the data they use. The second approach is the randomized choice of features. In normal decision trees while a splitting has to be always made the feature with the ability of

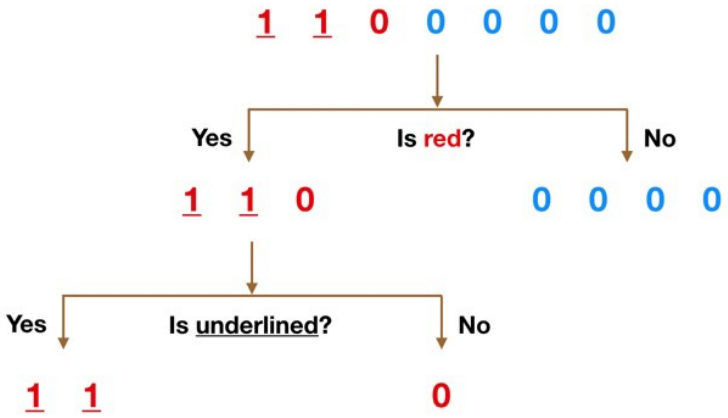

Figure 4.2 : A sample decision tree. [2]

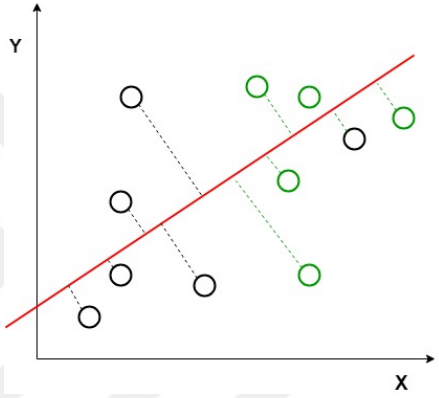

Figure 4.3 : 2D Coordinate System with the straight line which is created by *LDA* for dimensionality reduction. [3]

the largest separation is chosen but in *Random Forest* algorithms, because the sample set is differentiated for each tree, feature order changes for the decision points which creates diversification as well as low correlation.

4.4 Linear Discriminant Analysis (LDA)

LDA is a dimensionality reduction technique. In this technique the features can be seen as the dimensions. For the ease of the explanation assume that there are 2 features which can be shown as a 2D coordinate system and the data points are marked on this graph as the points with the color labels. This approach tries to find a straight line on the graph which can be the new 1D graph with the projections of the points on the line. While finding this straight line, two criteria must be followed in this approach: the distance between the means of the classes has to be maximized and the variation within each class has to be minimized. The process can be seen in the Fig: 4.3

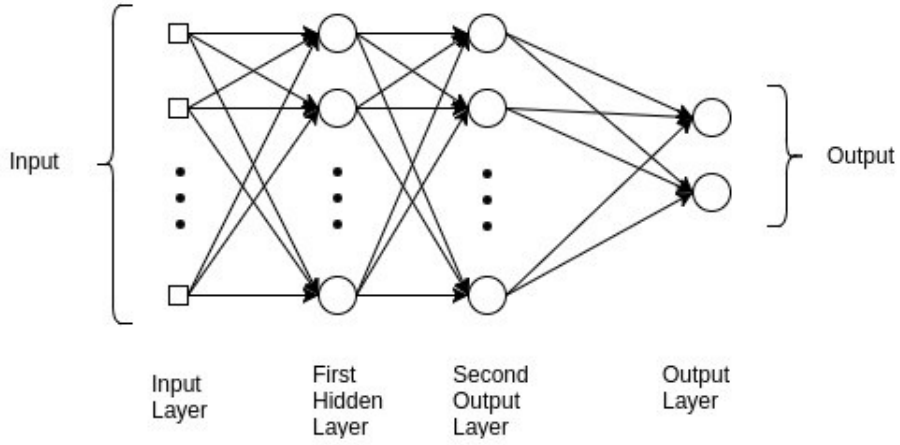

Figure 4.4 : A Sample MLP Structure. [4]

While using this approach, three different solvers can be used: *Singular Value Decomposition (SVD)*, *Least Squares Solution (LSQR)* and *Eigenvalue Decomposition (EIGEN)*. Among those solvers just *SVD* is not computing the covariance matrix which makes it able to be used with large number of features while others are unacceptably slow for such usage.

4.5 Multilayer Perceptron (MLP)

Multilayer Perceptron is the most typical and straight forward neural network model which consists of just multiple fully connected linear layers as it can be seen in Fig: 4.4. The linear layers use the function in Eq: 4.4 to calculate the outputs where f is the activation function, W is the weights, x is the input vector and b is the bias vector. Activation functions are used to increase the non-linearity to increase the model's flexibility while finding the relations between data. The common activation functions are *Sigmoid*, *ReLU* and *Tanh*.

$$y = f(WxT + b) \tag{4.4}$$

The last layer of this algorithm outputs the probabilities for each label option and the prediction is reflected as the one label which's probability is the highest among. After the output is created, a loss function is used to back propagate the model and optimize the weights.

5. DEEP LEARNING BASED SIGN RECOGNITION MODULE

5.1 Motivation

Deep learning based approaches are designed and implemented for the recognition of the sign language data that is collected by the application explained in Chapter 2, since deep learning based approaches are generally more efficient than the traditional heuristic methods to process the big and bulky data. In this study since the signs will be dynamic and feature extraction might be a costly process, three different deep learning based approaches are trained and tested in this study to get the best recognition results. These are: *Convolutional Neural Network (CNN)*, *Recurrent Neural Network (RNN)* and *Long Short-Term Neural Network (LSTM-NN)*.

5.1.1 Convolutional Neural Network (CNN)

Convolutional Neural Network is the most commonly used deep learning method for recognition and classification. It is generally used on big data sets involving images. *CNN*s can be examined by units consist of several layers. In a basic unit of *CNN*, the first layer is the convolutional layer which makes the main computation in the network. The layer has filters or kernels which are moved on the input data, as a sliding window, and calculations are processed by combining both the data surfaces (image and filter) to create the output data. The output data size of the layer changes according to the filter's size and filter's sliding step size used in the layer; as the n-sized filter is on one location of the input, it outputs just one data point to the next layer rather than n. After the convolutional layer the second layer is the activation layer in the unit. This layer is the data rectifier part of the unit to increase non-linearity of the network. The commonly used activation functions are *ReLU*, *Leaky ReLU* and *ELU*. The third layer is the pooling layer which makes down sampling on the data. Also these three part units of the network can be added to the network multiple times sequentially, which causes the network to grow in depth. After the layer units, at the end of the network,

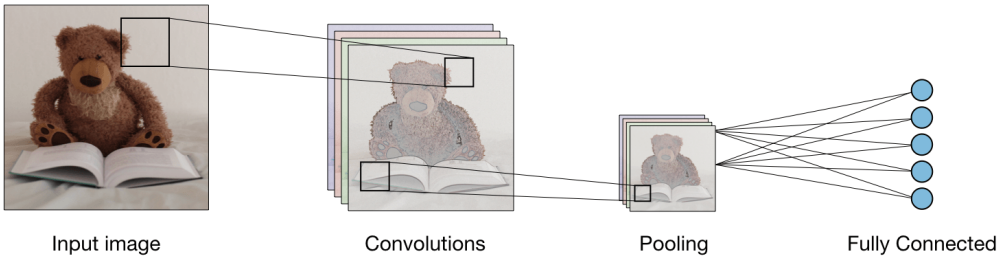

Figure 5.1 : *CNN* workflow [5]

one or multiple *Fully Connected (FC) Layer*s has to be added to the process to make the data even narrower, as 1 dimensional. To get the label predictions from this last layer a *Softmax* function is applied as the last step which converts the data coming from *FC* layer to vector of output probabilities, where the highest probability represents the predicted classification label. The basic architecture with one unit can be seen in the Fig: 5.1.

5.1.2 Recurrent Neural Networks (RNNs)

*RNN*s are the most common approaches for processing the data which include time series, such as the data in this study. They are fed forward networks with an internal memory which make them give the outputs, depending on the previous ones. This approach is very important for the times series data because this kind of data is highly dependent on all the time frames it includes in that particular order. This kind of networks give the partial outputs to the next layers (as shown in Fig: 5.2) as the additional inputs. The general architecture of *RNN*s can be seen in the Fig: 5.2. In the figure t is the time step, a is the activation and y is the output. These variables can be shown with the Eq: 5.1 and Eq: 5.2. W_{ax}, W_{aa} and W_{ya} are the weight coefficients, b_a and b_y are the bias coefficients g_1 and g_2 are the activation functions which are same in every neuron. The commonly used activation functions are *Sigmoid*, *Tanh*, and *ReLU* for *RNN*s. For this kind of neural networks, the loss function is calculated by summing up all the previous step losses in the network.

$$a^{<t>} = g_1(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a) \tag{5.1}$$

$$y^{<t>} = g_2(W_{ya}a^{<t>} + b_y) \tag{5.2}$$

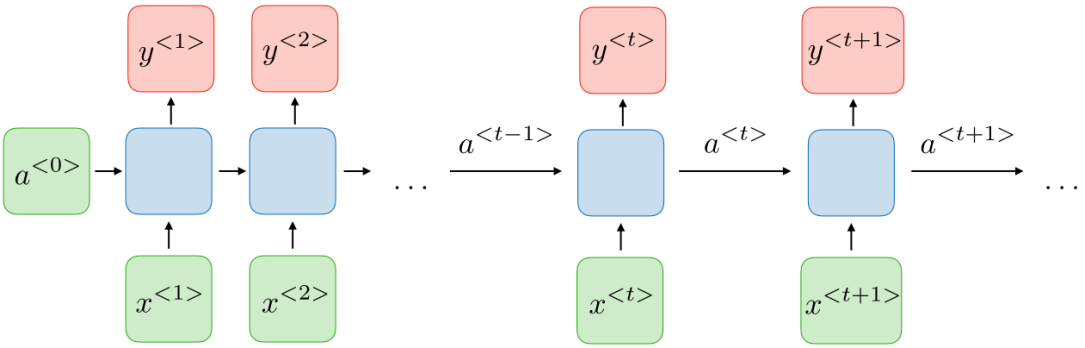

Figure 5.2 : General Architecture for *RNN* [6]

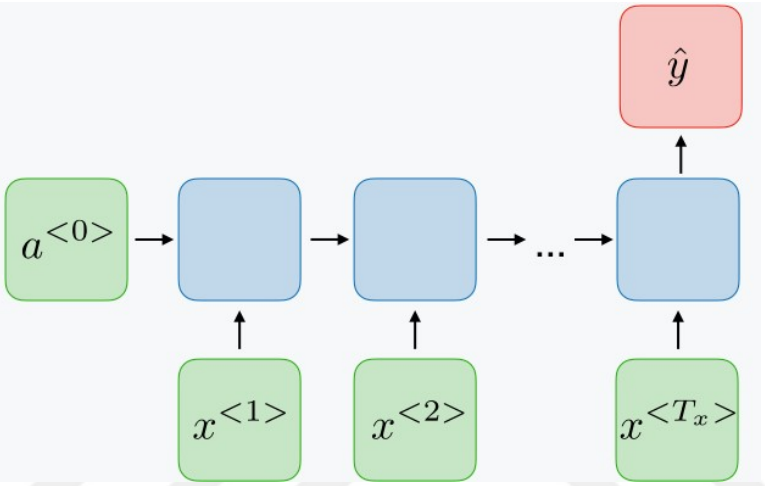

Figure 5.3 : Many-to-one case Architecture for *RNN* [6]

These networks can be used in various different cases according to the desired input and output sizes. The types of the cases are one-to-one, one-to-many, many-to-one, and many-to-many. In this study the most applicable type is the many-to-one case since the data have multiple inputs as the time frames and it desires one output as the sign label. The related structure can be seen on the Fig: 5.3

Basic *RNN*s have a common problem named vanishing or exploding gradient problems caused by the activation functions. During the back propagation, since the loss function is the sum-up of all previous losses and it creates a multiplicative gradient, if the time step amount is very large then derivative of the activation function causes the weights to vanish, because its value is smaller than 1. In the other case, if the weights become very large then the value of the gradient becomes too small to affect the weights which results in the exploding gradient problem. The data in this study is prone to have the vanishing gradient problem because the data size is fairly large. Therefore *Long*

Short-Term Memory approach is chosen to be the next step for better results since it solves these problems with additional gates in the neurons.

5.1.2.1 Long Short-Term Memory Neural Network (LSTM-NN)

*LSTM-NN*s are special *Recurrent Neural Network*s which resolves the vanishing/exploding gradient problems that make them able to have longer memories. This solution is achieved by having various gates in the neurons which are update gate (Γ_u) to decide how much past should matter, relevance gate (Γ_r) to decide if the previous information should be dropped, forget gate (Γ_f) to decide if current info has to be erased and output gate (Γ_o) to decide how much current output should be revealed. The architecture of one neuron of the *LSTM-NN* can be seen in the Fig: 5.4 The equations for the gates can be seen in Eq: 5.3 where W, U and b are the specific coefficients for the gates and σ is the *Sigmoid* function. Also the additional functions for *LSTM* can be seen in Eq: 5.4, Eq: 5.5 and Eq: 5.6 where $\odot$ is used for element-wise multiplication between two vectors.

$$\Gamma = \sigma(Wx^{<t>} + Ua^{<t-1>} + b) \tag{5.3}$$

$$\tilde{c}^t = \tanh(W_c[\Gamma_r \odot a^{<t-1>}, x^{<t>}] + b_c) \tag{5.4}$$

$$c^t = \Gamma_u \odot \tilde{c}^{<t>} + \Gamma_f \odot c^{<t-1>} \tag{5.5}$$

$$a^t = \Gamma_o \odot c^{<t>} \tag{5.6}$$

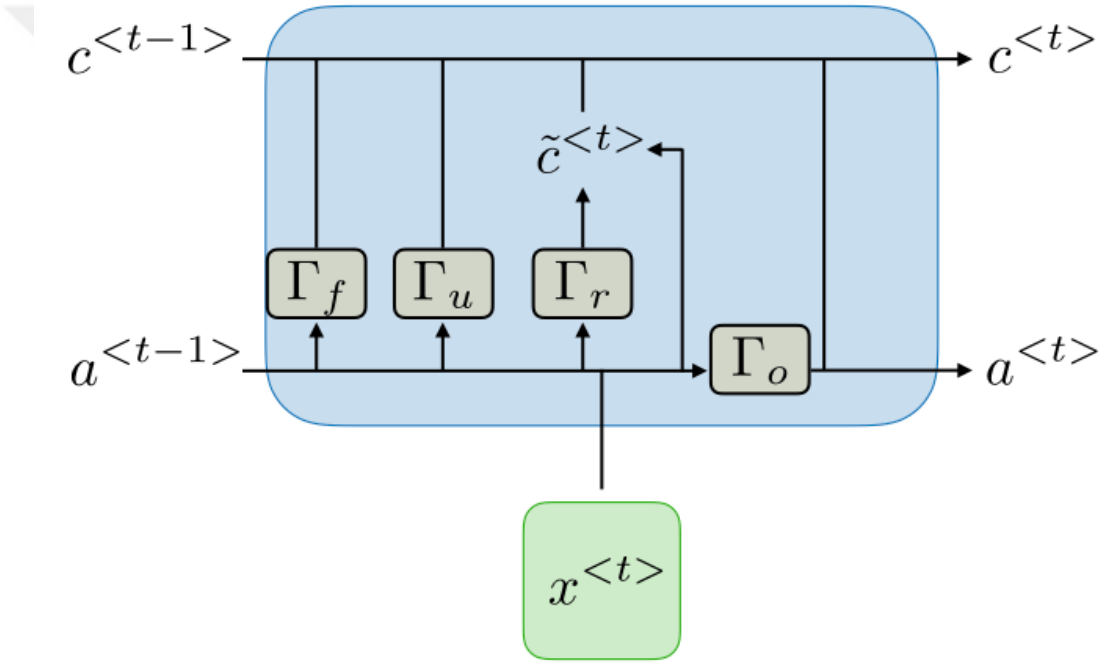

Figure 5.4 : An LSTM neuron [6]

6. EXPERIMENTS

6.1 Experimented Approaches

In this study six different traditional machine learning approaches and three different deep learning neural networks are used. These approaches are summarized as seen in Table: 6.1.

As the machine learning approaches; *Gaussian Naïve Bayes*, *SVM*, *k-NN*, *Random Forest*, *LDA* and *MLP* are experimented. The explanations of these approaches can be seen in Chapter 4. For all of these methods, 10 fold cross-validation is used to validate the classifiers to keep the validation subset size same with the deep learning approaches. As the hyperparameters 'priors' parameter of the Gaussian Naïve Bayes method is set to 'None' while other parameters are kept as their default values in *sklearn.scikit* library. For *SVM* approach all 5 different kernels (rbf, poly, linear, sigmoid, linearSVC) are experimented on individually. All the hyperparameters are used with their default values except gamma which is set to be 0.001 for all of the kernels which require gamma value. Also for the *k-NN* approach all hyperparameters are used with the default values where neighbour amount is set to 5. In the *Random Forest* approach maximum depth hyperparameter is set to 2 and random state is set to 0 while other parameters are left with the default values where estimator amount is set to 100. For *LDA* classifiers the solver is chosen as 'svd' and tol parameter is set to 0.0001 while other parameters are used with the default values. To consider the other solver options 'lsqr' option is tried but, because of its unacceptable slowness a result cannot be achieved. The cause of this slowness is explained in the Chapter 4. While training *MLP* none of the default values for hyperparameters are changed again.

As it is explained in the Chapter 2 and 3, the data in this project consist of 2 dimensional time series values. Even though it is a time series problem, because it is 2 dimensional it seems very similar to a monochromatic image data. Considering this similarity as the first deep learning approach, a *CNN* modal is created and trained. As

Table 6.1 : Experimented machine learning and deep learning approaches.

Method	Purpose
Gaussian Naïve Bayes Support Vector Machine K-Nearest Neighbour Random Forest Linear Discriminant Analysis Multilayer Perceptron	Adopted from previous studies [7] [8].
CNN	Image Like
RNN	Timeseries
LSTM	Large Timeseries

the activation layer *ReLU* is used after the convolutional layers and as the loss function *Cross Entropy Loss* is used. As the optimizer *Stochastic Gradient Decent (SGD)* and *Adam* functions are used with varying learning rates and varying momentum (for SGD). Also 3 *Fully-Connected (FC)* linear layers are added to the end of the network with a *Softmax* functions at the end to get the predicted label.

As the second deep learning approach, a basic *RNN* model is created and trained because the data includes time series. The model created with one *RNN* layer and 3 *FC* linear layers with a *Softmax* function as in the previous *CNN* approach.

As the third deep learning approach *LSTM-NN* model is created and trained because the data consists of around 300 time frames as one input which would be the cause of *RNN*'s low results because of its characteristic of not being able to keep enough memory caused by its vanishing gradient problem. Therefore an *LSTM-NN* model created with one *LSTM* layer with dropout, and three *Fully Connected* linear layers together with a *Softmax* function at the end. As the loss function same *Cross Entropy Loss* function is used and as the optimizer *SGD* and *Adam* functions are used with varying learning rates as in other deep learning approaches.

6.2 Experiments

For the experiments, first of all, a sign dataset is created with 2400 samples that are collected via the developed tool that is explained in Chapter 2. While collecting the data multiple, cross-gender and cross-age signers used the tool to give the samples. All the data is collected in indoor environments which are under various light conditions;

Table 6.2 : Machine Learning Classifier Accuracies

ML Model	Train Acc %	Val Acc %	Test Acc %
Gaussian Naïve Bayes	85.926	78.19	74.583
SVM (kernel: rbf)	1	8.33	8.333
SVM (kernel: poly)	1	79.81	77.083
SVM (kernel: linear)	1	88.56	85.833
SVM (kernel: sigmoid)	0.231	0.32	0
SVM (LinearSVC)	1	88.38	82.916
Kneighbours	89.768	81.67	78.75
Random Forest	74.166	69.491	72.083
LDA	99.907	98.24	95
MLP	98.889	85.88	75.417

naturally lighted, artificially well lighted, artificially mildly lighted, artificially less lighted. The dataset includes 200 samples per each word sign.

The experiments are implemented with *Python* language using *sklearn* for traditional machine learning classifiers and *Pytorch* for deep learning models, which is an open source machine learning framework. Before starting the training, the data processing steps in Chapter 3 are performed.

For the traditional machine learning approaches the datasets are used as they are and 10 fold cross validation is performed to evaluate the classifiers before testing. Although various different hyperparameters are tried in multiple runs of training the best hyperparameters are found as mostly the defaults as it is explained in the Section 6.1. The results of the classifiers can be seen in Table 6.2. The best test accuracy is achieved with LDA classifier which is significantly higher than other machine learning approaches. Which means although the classifier is a very simple one, while being robust, it is very successful on the test set because the data used in this experiment is separable with linear boundaries adequately to reach 95% test accuracy. Also the confusion matrix of this classifier can be seen in Fig: 6.1.

After the machine learning experiments also the deep learning approaches are experimented on with the same dataset but, after the preprocessing, the train dataset is randomly divided into two as training and validation sets by keeping sample amount equal for each sign for each partition. For the training 90% of the train dataset is used which means, it consists of 180 % 90 = 162 samples for each sign, as a total of 2160 % 90 = 1944 samples. For the validation datasets, the remaining of the train dataset

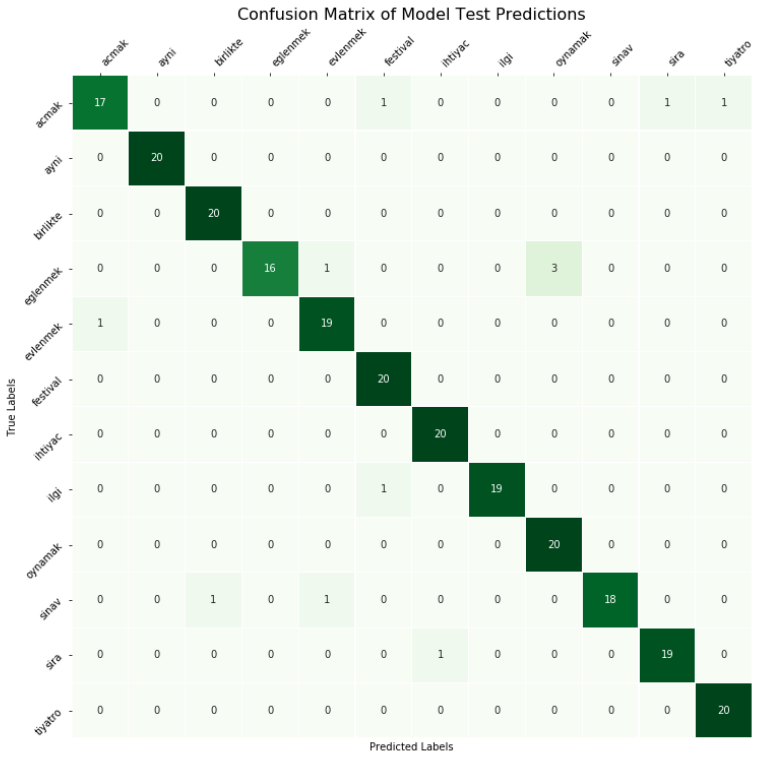

Figure 6.1 : *LDA* Confussion Matrix

is used, which means it includes 180 - 162 = 18 samples for each sign as a total of 2160 - 1944 = 216 samples. By partitioning the dataset like this, the validation set size kept equal with the validation set size of the traditional machine learning approaches which creates validation sets with 10 fold cross validation. To use these datasets in the neural network models, data loaders are generated for each of them with shuffling feature enabled to keep the bias of the data order as small as possible.

After the data loaders are prepared, a simple CNN is created to keep the neural network as shallow as possible to make the training process faster with high accuracy results. In this *CNN*, two convolution layers are used with *Rectified Linear Unit (ReLU)* and *MaxPool2d* functions consecutively, continuing with two consecutive *Fully Connected (FC)* layers paired with again *ReLU* function and an additional *FC* and *Softmax* layer at the end to output the label together with a dropout layer. The architecture of the neural network can be seen in the Fig: 6.2. With this model *Cross Entropy Loss* is used as the loss function, and *Stochastic Gradient Decent* (without momentum) with "learning rate" feature is set to 0.001 are used. The training is run for 100 epochs with a batch size of 5 by using the *Graphical Programming Units (GPUs)* that *Microsoft Azure*

```
CNNNet(
  (conv1): Conv2d(1, 6, kernel_size=(5, 5), stride=(1, 1))
  (conv2): Conv2d(6, 16, kernel_size=(5, 5), stride=(1, 1))
  (pool): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  (fc1): Linear(in_features=39360, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=80, bias=True)
  (fc3): Linear(in_features=80, out_features=12, bias=True)
  (dropout): Dropout(p=0.3, inplace=False)
)
```

Figure 6.2 : *CNN* architecture.

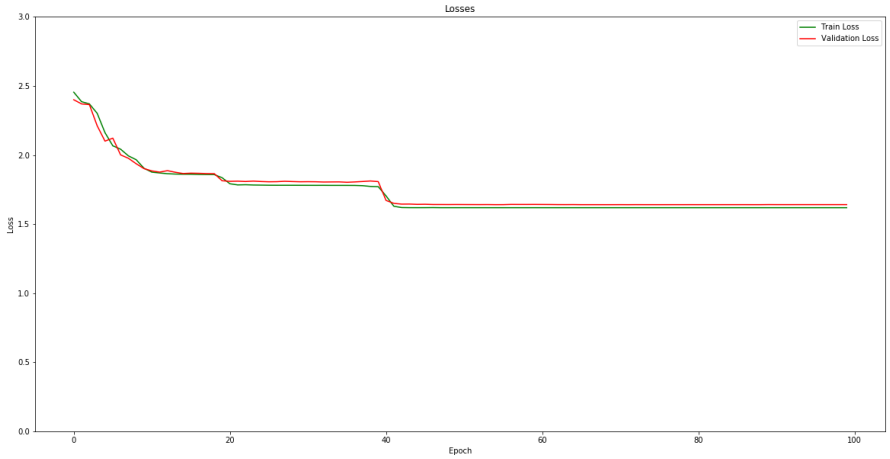

Figure 6.3 : *CNN* losses graph.

Virtual Machine includes to reach the results faster than the local computer. After several trials with different hyperparameters, achieved best accuracy result became as 100% for training, 97.7% for validation and 96.7% for test. The loss and accuracy graphs can be seen in the Fig: 6.3 and Fig: 6.4 respectively. Also the confusion matrix can be seen in the Fig: 6.5.

As the second deep learning model experiment, an *RNN* model is created with 1 *RNN* layer, 1 droupout layer, 2 *FC* layers combined with *ReLU* function and one final *FC* layer following with a *Softmax* layer to get the outputs as labels which are exactly same with the previous CNN model. The architecture can be seen in the Fig: 6.6. With this model the same *Cross Entropy Loss* is used as the one used with the *CNN* but the optimizer is changed to *Adam* function after some trials with *SGD* because it gave better results for this neural network. The *Adam* function is used with a "learning rate" feature which is set to 0.001. This neural network also trained for 100 epochs with batch size set to 1 with same provided *GPU*s. 1 is the optimized value of the batch size where some other values are also experimented on. After several results with different

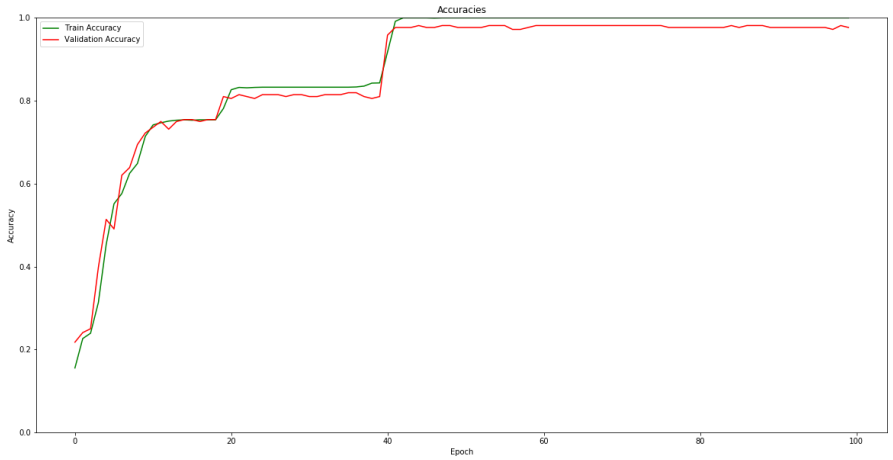

Figure 6.4 : *CNN* accuracies graph.

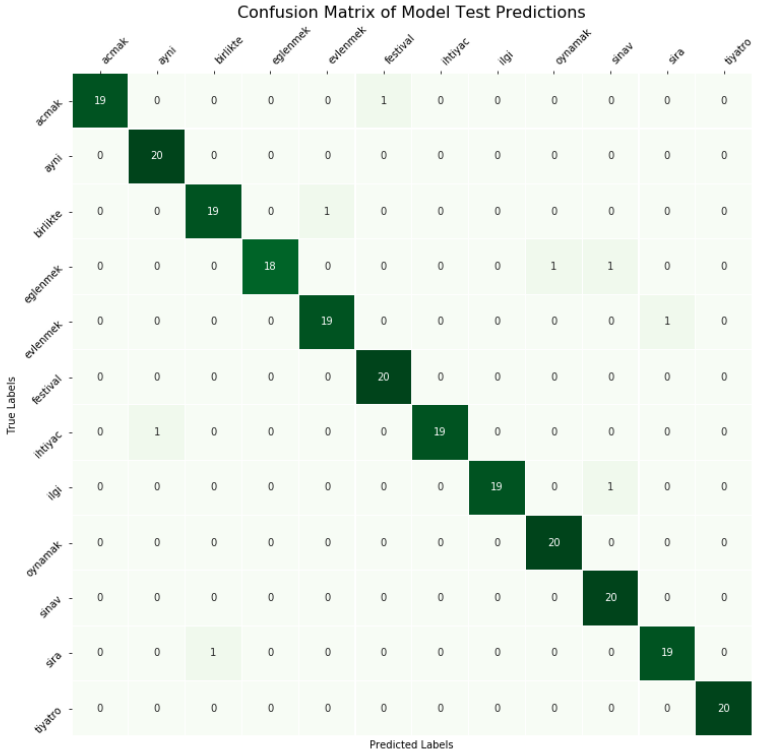

Figure 6.5 : *CNN* confusion matrix.

```
RNNNet(
  (rnn): RNN(44856, 250, batch_first=True, dropout=0.3)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc1): Linear(in_features=250, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=80, bias=True)
  (fc3): Linear(in_features=80, out_features=12, bias=True)
)
```

Figure 6.6 : *RNN* architecture.

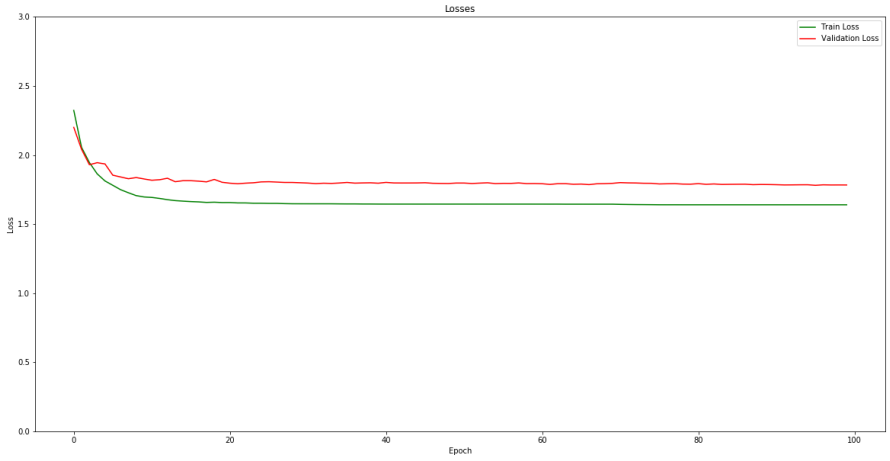

Figure 6.7 : *RNN* losses graph.

hyperparameters the best achieved accuracy results are as 97.8% for training, 84.3% for validation and 85.4% for test. The loss and accuracy graphs can be seen in the Fig: 6.7 and Fig: 6.8. Also the confusion matrix can be seen in the Table: 6.9.

As the third deep learning model experiment, an *LSTM* model is created with 1 *LSTM* layer with 2 hidden layers, 1 droupout layer, 2 *FC* layers combined with *ReLU* function and one final *FC* layer following with a *Softmax* layer to get the outputs as labels which are exactly same with the previous models. The architecture can be seen in the Fig: 6.10. With this model the same *Cross Entropy Loss* is used as the one used with the previous models and for the optimizer both *SGD* and *Adam* functions experimented on but as it is seen with the *RNN Adam* function gave better results with this model too. The *Adam* function is used with a "learning rate" feature which is set to 0.001. This neural network also trained for 100 epochs with optimized batch size (after multiple trials) as 1 with same provided *GPU*s. After several results with different hyperparameters the best achieved results are as 97.1% for training, 80.6%

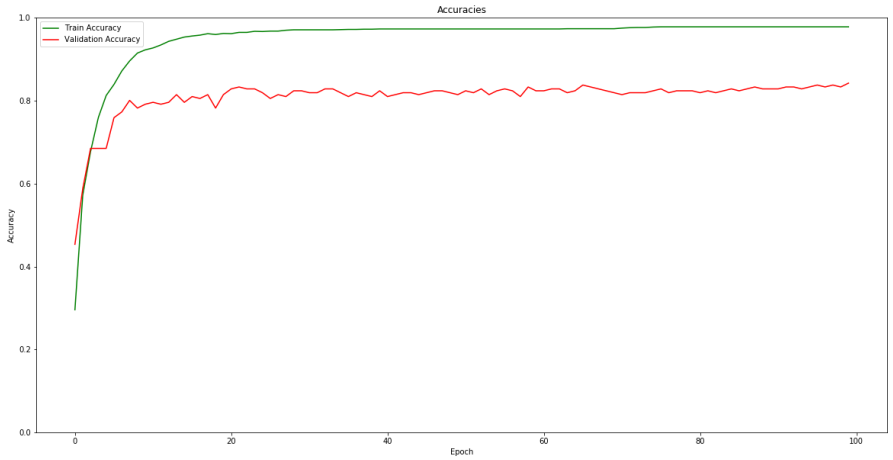

Figure 6.8 : *RNN* accuracies graph.

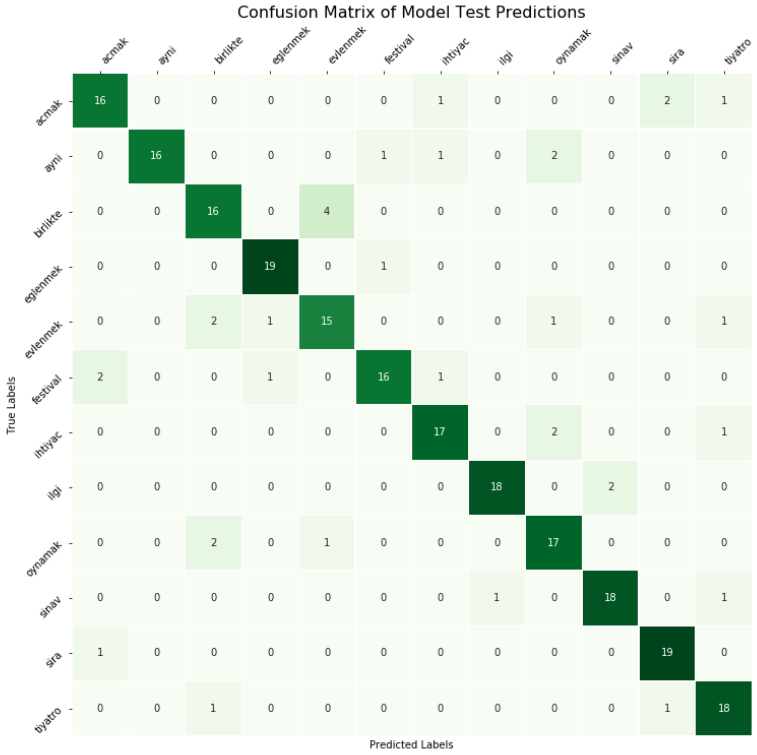

Figure 6.9 : *RNN* confussion matrix.

```
LSTMNet(
  (lstm): LSTM(44856, 250, num_layers=2, batch_first=True, dropout=0.3)
  (dropout): Dropout(p=0.3, inplace=False)
  (fc1): Linear(in_features=250, out_features=120, bias=True)
  (fc2): Linear(in_features=120, out_features=80, bias=True)
  (fc3): Linear(in_features=80, out_features=12, bias=True)
)
```

Figure 6.10 : *LSTM* architecture.

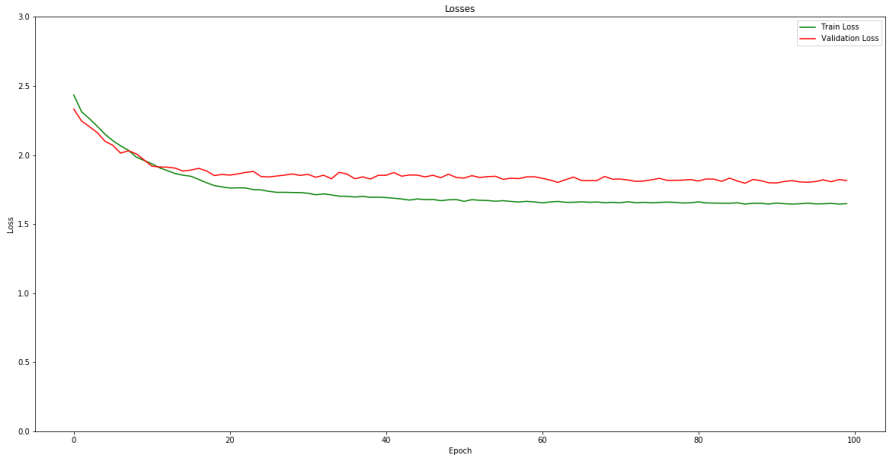

Figure 6.11 : *LSTM* losses graph.

for validation and 80.4% for test. The loss and accuracy graphs can be seen in the Fig: 6.11 and Fig: 6.12. Also the confusion matrix can be seen in the Table: 6.13.

After the best hyperparameters of the models are found for the current dataset one more experiment set is conducted on LDA (which gave the best results among the traditional machine learning approaches), CNN, RNN and LSTM to do cross validation on the whole dataset for all models with the same train-test dataset pairs and get the average results. For this part of the experiments, 5 different train-test pairs are created with the whole dataset. All pairs have same number of samples for each dataset as it is explained in the previous part of the experiment. Therefore difference between the dataset pairs are; they include different samples for each sign for test and train. When the whole dataset is divided into two as train and test, the sample picking is done randomly for each 5 pairs while keeping the sample amount same for each different sign (label). After the dataset pairs are created, for each pair the models are run for 5 times to get the average results for more precise resolutions. The results can be seen in Fig: 6.14. The highlighted parts in the table reflects the best run for the particular dataset with particular model among the 5 trials. As it can be seen from the table, the

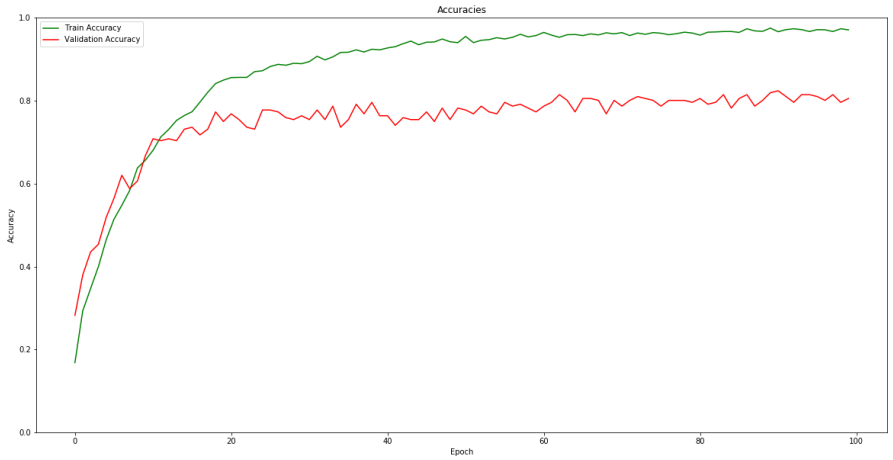

Figure 6.12 : *LSTM* accuracies graph.

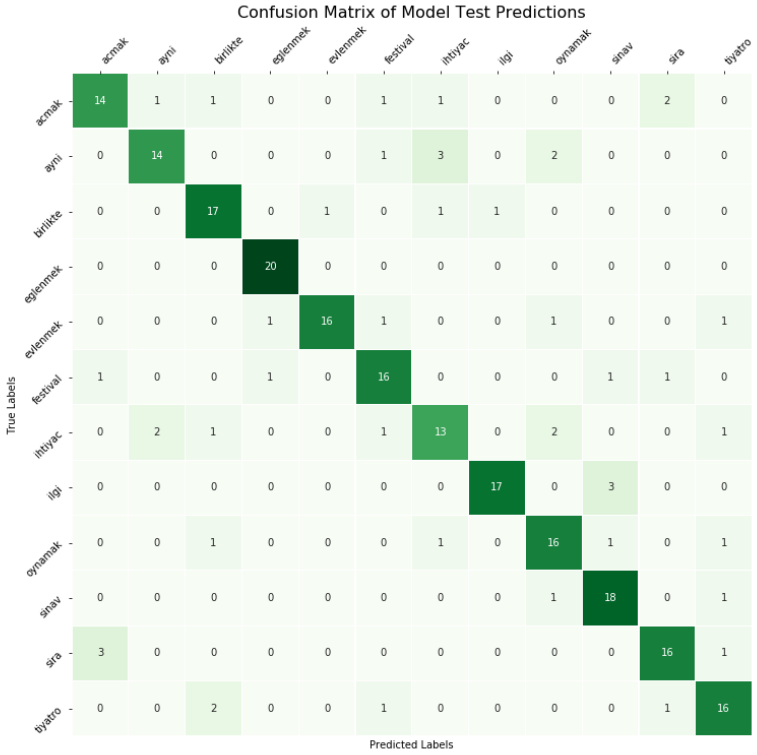

Figure 6.13 : *LSTM* confussion matrix.

best test accuracies are 98.8% for LDA, 98.8% for CNN, 88.3% for RNN and 85.0% for LSTM while the best average test accuracies are 98.8% for LDA, 97.9% for CNN, 94.9% for RNN and 82.3% for LSTM. Also the average of all paired datasets' test accuracy average results are as following: 97.4% for LDA, 96.9% for CNN, 82.8% for RNN and 79.2% for LSTM.

		LDA					CNN					RNN					LSTM				
	Trial	Train Acc	Val Acc	Test Acc	Train Time	Test Time (ms)	Train Acc	Val Acc	Test Acc	Train Time	Test Time (ms)	Train Acc	Val Acc	Test Acc	Train Time	Test Time (ms)	Train Acc	Val Acc	Test Acc	Train Time	Test Time (ms)
Dataset 1	1	0.9991	0.9824	0.9500	00:00:35.173	64.1130	1.0000	0.9769	0.9667	00:09:09.684	196.6200	0.9676	0.7963	0.8292	00:47:38.073	342.0140	0.9630	0.7917	0.8083	02:23:57.727	538.7300
	2	0.9991	0.9824	0.9500	00:00:21.450	56.6720	1.0000	0.9907	0.9625	00:09:14.623	202.7060	0.9753	0.8194	0.8333	00:47:52.480	337.8730	0.9676	0.7917	0.7958	02:24:19.958	621.7060
	3	0.9991	0.9819	0.9500	00:00:34.312	66.0250	1.0000	0.9769	0.9458	00:09:12.752	209.2850	0.9624	0.7963	0.8125	00:45:10.310	327.9760	0.9769	0.7639	0.7750	02:23:58.132	554.8100
	4	0.9991	0.9847	0.9500	00:00:34.545	61.9720	0.9995	0.9769	0.9667	00:09:12.655	198.6300	0.9784	0.8426	0.8542	00:45:41.000	327.9640	0.9712	0.8056	0.8042	02:38:55.098	676.8970
	5	0.9991	0.9824	0.9500	00:00:33.179	65.8000	0.9995	0.9769	0.9375	00:09:11.533	202.6060	0.9666	0.8519	0.8167	00:45:14.000	323.7560	0.9686	0.8194	0.7875	02:38:41.457	661.7000
	Avg.	**0.9991**	**0.9828**	**0.9500**	00:00:31.732	**62.9164**	**0.9998**	**0.9796**	**0.9558**	00:09:12.249	**201.9694**	**0.9701**	**0.8213**	**0.8292**	00:46:19.173	**331.9166**	**0.9694**	**0.7944**	**0.7942**	02:29:58.474	**610.7686**
Dataset 2	1	0.9986	0.9819	0.9875	00:00:21.356	57.8980	0.9990	0.9907	0.9750	00:09:28.444	205.6720	0.9769	0.8472	0.8708	00:45:11.304	355.4980	0.9722	0.8102	0.8250	02:38:17.349	632.4940
	2	0.9986	0.9778	0.9875	00:00:21.403	58.5970	0.9995	0.9954	0.9875	00:09:28.297	203.3750	0.9769	0.8333	0.8625	00:45:15.000	342.4290	0.9691	0.7315	0.8125	02:38:46.290	649.0540
	3	0.9986	0.9782	0.9875	00:00:21.354	56.9710	0.9995	0.9954	0.9750	00:09:28.858	197.3900	0.9722	0.8194	0.8375	00:45:13.349	364.3080	0.9691	0.7963	0.8292	02:39:35.887	700.0160
	4	0.9815	0.9986	0.9875	00:00:21.452	58.5930	1.0000	0.9815	0.9875	00:09:30.627	198.8250	0.9743	0.8750	0.8167	00:45:18.249	328.7840	0.9748	0.7824	0.8208	02:39:49.000	672.7950
	5	0.9986	0.9819	0.9875	00:00:21.397	56.4840	1.0000	0.9815	0.9708	00:09:30.564	201.6130	0.9650	0.8333	0.8583	00:45:24.464	334.5540	0.9743	0.7917	0.8250	02:39:52.744	689.4990
	Avg.	**0.9952**	**0.9837**	**0.9875**	00:00:21.392	**57.7086**	**0.9996**	**0.9889**	**0.9792**	00:09:29.358	**201.3750**	**0.9730**	**0.8417**	**0.8492**	00:45:16.473	**345.1146**	**0.9719**	**0.7824**	**0.8225**	02:39:16.254	**668.7716**
Dataset 3	1	0.9995	0.9801	0.9875	00:00:34.989	60.8630	0.9995	0.9769	0.9750	00:09:29.363	204.7990	0.9630	0.8241	0.8250	00:45:00.676	331.2030	0.9676	0.8519	0.7750	02:38:36.003	689.5450
	2	0.9995	0.9829	0.9875	00:00:21.574	58.8940	0.9990	0.9990	0.9708	00:09:31.457	205.7250	0.9681	0.8241	0.8208	00:45:25.727	326.5150	0.9707	0.8241	0.7958	02:40:22.231	687.8980
	3	0.9995	0.9829	0.9875	00:00:21.000	60.7650	0.9995	0.9815	0.9625	00:09:30.595	203.9030	0.9841	0.8750	0.8333	00:45:17.240	325.0110	0.9645	0.7222	0.7833	02:40:27.056	634.0330
	4	0.9995	0.9833	0.9875	00:00:21.682	62.7020	0.9995	0.9491	0.9667	00:09:32.688	212.6510	0.9733	0.8148	0.8167	00:45:29.897	332.0840	0.9655	0.7593	0.7542	02:40:18.769	618.3550
	5	0.9995	0.9829	0.9875	00:00:21.683	62.9760	1.0000	0.9815	0.9792	00:09:33.293	210.3820	0.9748	0.8657	0.7917	00:45:20.355	325.4030	0.9650	0.7917	0.7708	02:24:43.104	639.1390
	Avg.	**0.9995**	**0.9824**	**0.9875**	00:00:24.186	**61.2400**	**0.9995**	**0.9776**	**0.9708**	00:09:31.479	**207.4920**	**0.9726**	**0.8407**	**0.8175**	00:45:18.779	**328.0432**	**0.9667**	**0.7898**	**0.7758**	02:36:53.433	**653.7940**
Dataset 4	1	1.0000	0.9819	0.9792	00:00:22.000	57.3260	1.0000	0.9722	0.9833	00:09:33.070	211.0150	0.9774	0.8241	0.8667	00:45:09.035	323.4870	0.9774	0.8009	0.8333	02:38:17.788	714.2210
	2	1.0000	0.9782	0.9792	00:00:21.618	57.2880	1.0000	0.9583	0.9833	00:09:31.745	202.6150	0.9702	0.8426	0.8667	00:45:28.756	327.9930	0.9727	0.7824	0.7958	02:23:21.678	614.8460
	3	1.0000	0.9806	0.9792	00:00:23.000	65.9730	0.9995	0.9722	0.9667	00:09:33.470	208.2390	0.9748	0.8194	0.8417	00:45:27.416	335.6840	0.9691	0.7685	0.8083	02:23:21.174	605.4230
	4	1.0000	0.9824	0.9792	00:00:21.798	59.9810	0.9995	0.9722	0.9667	00:09:33.470	208.2390	0.9753	0.8380	0.8833	00:45:20.407	322.6980	0.9743	0.8056	0.8500	02:23:06.511	631.2310
	5	1.0000	0.9824	0.9792	00:00:21.543	57.1130	1.0000	0.9815	0.9792	00:09:31.415	203.3960	0.9671	0.8380	0.8750	00:45:15.680	331.4760	0.9624	0.7824	0.7917	02:23:35.619	547.2480
	Avg.	**1.0000**	**0.9811**	**0.9792**	00:00:21.992	**59.5362**	**0.9998**	**0.9713**	**0.9758**	00:09:32.634	**206.7008**	**0.9729**	**0.8324**	**0.8667**	00:45:20.259	**328.2676**	**0.9712**	**0.7880**	**0.8158**	02:26:20.554	**622.5938**
Dataset 5	1	0.9986	0.9792	0.9708	00:00:21.670	64.0880	0.9995	0.9491	0.9625	00:09:28.806	207.1230	0.9825	0.8426	0.8417	00:45:13.486	323.3590	0.9707	0.8102	0.7583	02:38:17.000	652.2040
	2	0.9986	0.9796	0.9708	00:00:21.587	63.9980	0.9969	0.9769	0.9708	00:09:31.315	202.8750	0.9681	0.8102	0.8042	00:45:18.983	318.6400	0.9769	0.7731	0.7250	02:24:10.932	587.1640
	3	0.9986	0.9819	0.9708	00:00:21.673	58.4200	0.9995	0.9491	0.9667	00:09:29.362	200.2320	0.9748	0.8287	0.7792	00:45:15.805	321.8420	0.9645	0.7963	0.8000	02:23:52.081	556.2860
	4	0.9986	0.9824	0.9708	00:00:21.760	56.8160	0.9995	0.9676	0.9667	00:09:32.627	205.1920	0.9666	0.8148	0.8083	00:45:35.745	334.0180	0.9753	0.8704	0.7875	02:25:24.575	619.2640
	5	0.9986	0.9815	0.9708	00:00:21.558	57.5850	1.0000	0.9861	0.9583	00:09:31.695	204.0550	0.9810	0.8333	0.8417	00:45:23.493	351.1380	0.9712	0.8241	0.7958	02:24:02.253	543.9910
	Avg.	**0.9986**	**0.9809**	**0.9708**	00:00:21.650	**60.1814**	**0.9991**	**0.9657**	**0.9650**	00:09:30.761	**203.8954**	**0.9746**	**0.8259**	**0.8150**	00:45:21.502	**329.7994**	**0.9717**	**0.8148**	**0.7733**	02:27:09.368	**591.7818**

Figure 6.14 : Cross validation results of LDA, CNN, RNN and LSTM.

7. CONCLUSIONS AND RECOMMENDATIONS

The aim of this study is to recognize dynamic hand gestures and signs using *Leap Motion* sensor. For this aim, several methods based on traditional machine learning and deep learning based approaches are developed and tested on 12 words from the Turkish Sign Language, which have dynamic two-handed signs. *Leap Motion Controller* is employed as the depth sensor in this study, since it is compact, cost-efficient and suitable for mobile use.

In the first step of the project, an efficient and broadly usable tool is created to collect sign gesture data samples via *Leap Motion Controller*. This tool collects multiple time frames with 178 different features for each sample simulation data and automatically labels the data at the end of the recording. By using this tool, 2400 evenly distributed data samples are collected. After the dataset is created some preprocessing steps are applied to the data to make it usable for the deep learning approaches. In the second step of the project six different traditional machine learning and three different deep learning approaches are adopted in combination with multiple optimizers.

As the traditional machine learning approaches, Gaussian Naive Bayes, Support Vector Machine, K-Nearest Neighbour, Random Forest, Linear Discriminant Analysis and Multilayer Perceptron methods are used and classifiers are created with these methods. With different parameters multiple experimental runs are conducted with these classifiers and they are optimized. According to the results, it is concluded that *LDA* is the most successful method among the machine learning approaches. *LDA* is followed by *SVM* with linear kernels as it is seen in the test results. These results show that the dataset is linearly separable, but also contains high similarities between labels. Both of these approaches are very successful for linearly separable datasets but *LDA* does better predictions when the data is highly linear because *SVM* does more complex operations on the data even though it uses linear kernel. To verify this high linearity in the dataset, first the variance of the feature components are analyzed with *LDA*. This analysis can be seen on the Fig: 7.1. Then again with *LDA*, a dimensionality reduction

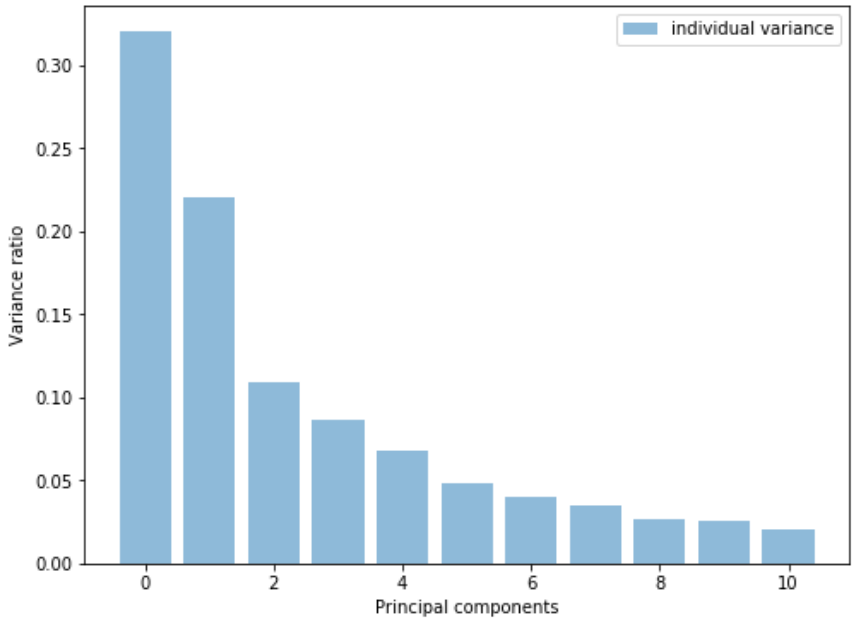

Figure 7.1 : *LDA* variance graph.

is applied to create a 2 dimensional graph to visualize the data as seen on Fig: 7.2. From the graph it can be observed that the dataset is mostly linear separable, therefore using *SVM* on this dataset resulted an overfitting problem and lower test accuracy.

As the first deep learning approach, a *Convolutional Neural Network* model is created with *Cross Entropy Loss* and *Stochastic Gradient Decent* optimizer. The hyperparameters are altered between each training runs and the model optimized according to the training and validation result analysis. As the second approach a *Basic Recurrent Neural Network* model is created with *Cross Entropy Loss* and *Adam* optimizer and the model is optimized by altering the hyperparameters and analyzing the results. As the third approach, a *Long Short-Term Memory Neural Network* model is created with again *Cross Entropy Loss* and *Adam* optimizer. This model is also optimized by altering the hyperparameters between training runs and analyzing the training and validation results. After both of the models are optimized a test sample set, which is evenly distributed across the signs and is not used for training or validation, is given to the models to get the test results.

As the final part of the experiments, 5 different train-test dataset pairs are created from the whole dataset to do precise cross validation on all of the chosen models. On these 5 dataset pairs optimized LDA, CNN, RNN and LSTM models are trained and tested again for 5 times and average values are collected for train, validation and test accuracies with the train time and test time. The average of all dataset pairs' average

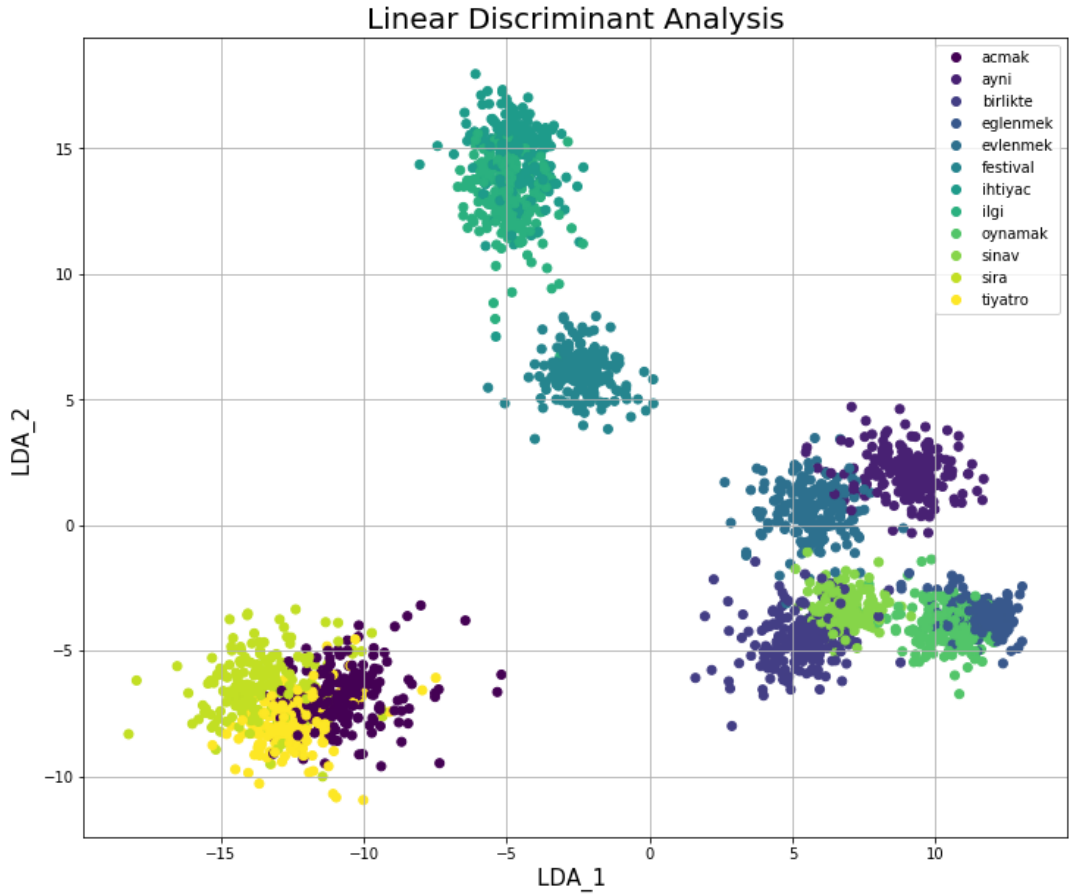

Figure 7.2 : Feature reduction to 2 dimensions with *LDA* graph.

test accuracy results are achieved as following: 97.4% for LDA, 96.9% for CNN, 82.8% for RNN and 79.2% for LSTM.

As the conclusion, it is shown that for the recognition of the chosen dataset *CNN* approach is more successful than both *RNN* and *LSTM* approaches while it is slightly less than the *LDA* approach. These results verify H1 where *CNN* shows a higher success rate than other deep learning approaches but because *LDA* is as successful as *CNN*, H2 cannot be verified with these results.

These results lead to the following conclusions. As the first and most apparent conclusion; although the dataset consists of time series and the most popular approaches are *RNN*s for this kind of datasets, both *RNN* and *LSTM* had significantly worse results than both *LDA* and *CNN* approaches which are known to be simpler learning methods. The most probable cause of this is the size of the dataset because this kind of neural networks generally need huge amount of data to learn the relations in between the data and the labels. Another highly probable cause is the structure of the data. *RNN* kind neural networks need properly structured data for high success rates but to construct the data as properly as they need costs high computation power for preprocessing and time which is not desirable when the ultimate aim is the real-time recognition. Also as it can be seen on the Fig: 6.14 the test times are very high for these models which means models are more complex and there is latency in the recognition. Therefore with their low test accuracies and high test times these two models are dropped for model considerations of following studies where similar dataset will be used.

As the second conclusion; the collected dataset is linearly separable for accurate classification of 12 chosen labels and this characteristic of the dataset is resulted with a successful *LDA* classifier for the recognition. On the other hand, this characteristic of the dataset is highly dependent on the label amount. Because this study aims to cover whole sign language in the future, 12 labels can be accepted as just the initiation of the sign language recognition system and although *LDA* is very successful for this amount of label its success will most probably drop as the label amount increases. While this dependency exists for *LDA* and it is logically apparent, *CNN* also gives nearly as successful results without any similar apparent dependency on the dataset. With this

consideration for the future experiments, *CNN* seems like a better approach for similar datasets with more labels.

For the third conclusion, the confusion matrices of *LDA* and *CNN* approaches are considered. As it can be seen from the confusion matrices in Fig: 6.1 and Fig: 6.5, the worst result is for "eglenmek" label for both methods. *CNN* is more successful on this sign with 18/20 correct predictions while *LDA* gives just 16/20 correct predictions for the same label and it confuses the label mostly with the label "oynamak". Moreover although overall accuracy results are slightly better for *LDA* it has 2 signs which's prediction accuracies are less than 18/20 where *CNN* correctly predicted all of the labels for more than or equal to 18/20. Therefore this conclusion also supports the previous one as *LDA*'s success highly varies with the different labels especially when the data for multiple labels gets similar to each other which most probably means they lose the linear separability characteristic. As the results of this evaluation, H2 can be verified while *CNN* can be accepted as more successful than *LDA* with this point of view.

Even though this study only includes 12 signs, the results are very promising, and the study can be extended to cover as much of the Turkish Sign Language as possible, to fulfill the needs of the deaf and hard-of-hearing people, which is the ultimate purpose of this research.

To reach better results several different approaches can be adopted for the future studies. These approaches can be summarized as; increasing the sample dataset size as much as possible (at least 500 samples per sign), increasing the number of features by using multiple sensors which are located at the different angles, applying a feature selection strategy to prevent the noise in the data additional to making the process faster.

Also as a future study of this project, a new application for the real-time recognition of signs, is going to be developed by using the proposed model. This new application is going to work cross platform and also across mobile devices and it will be

user-friendly, to be used in daily life communication of deaf and hard-of-hearing people.

REFERENCES

[1] **learn developers, S.**, Support Vector Machines, https://scikit-learn.org/stable/modules/svm.html, reference date: 2020-06-16.

[2] **Yiu, T.**, Understanding Random Forest, https://towardsdatascience.com/understanding-random-forest-58381e0602d2, reference date: 2020-06-16.

[3] **raman_257**, ML | Linear Discriminant Analysis, https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/, reference date: 2020-06-16.

[4] **Kain, N.K.**, Understanding of Multilayer perceptron (MLP), https://medium.com/@AI_with_Kain/understanding-of-multilayer-perceptron-mlp-8f179c4a135f, reference date: 2020-06-16.

[5] **Amidi, A. and Amidi, S.**, CS 230 - Deep Learning - Convolutional Neural Networks cheatsheet, https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks, reference date: 2020-05-03.

[6] **Amidi, A. and Amidi, S.**, CS 230 - Deep Learning - Recurrent Neural Networks cheatsheet, https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks, reference date: 2020-05-03.

[7] **Demircioglu, B.**, **Bulbul, G. and Kose, H.** (2016). Recognition of Sign Language Hand Shape Primitives With Leap Motion, *7th Workshop on the Representation and Processing of Sign Languages: Corpus Mining, Language Resources and Evaluation Conference (LREC 2016)*, volume 1, pp.47–52.

[8] **Demircioglu, B.**, **Bulbul, G. and Kose, H.** (2016). Turkish Sign Language recognition with Leap Motion, *2016 24th Signal Processing and Communication Application Conference (SIU)*, pp.589–592.

[9] **Chuan, C.H.**, **Regina, E. and Guardino, C.** (2014). American Sign Language Recognition Using Leap Motion Sensor, *Machine Learning and Applications (ICMLA), 2014 13th International Conference on*, pp.541–544.

[10] **Mohandes, M.**, **Aliyu, S. and Deriche, M.** (2014). Arabic sign language recognition using the leap motion controller, *Industrial Electronics (ISIE), 2014 IEEE 23rd International Symposium on*, pp.960–965.

[11] **Bassem Khelil, H.A.** (2016). Hand Gesture Recognition Using Leap Motion Controller for Recognition of Arabic Sign Language, *3rd International Conference on Automation, Control, Engineering and Computer Science (ACECS'16)*, pp.233–238.

[12] **Mapari, R.B. and Kharat, G.** (2015). Real time human pose recognition using leap motion sensor, *2015 IEEE International Conference on Research in Computational Intelligence and Communication Networks (ICRCICN)*, pp.323–328.

[13] **Makiko Funasaka, Yu Ishikawa, M.T. and Joe, K.** (2015). Sign Language Recognition using Leap Motion Controller, *Int'l Conf. Par. and Dist. Proc. Tech. and Appl. | PDPTA'15 |*, p.263.

[14] **Kumar, P.**, **Saini, R.**, **Behera, S.K.**, **Dogra, D.P. and Roy, P.P.** (2017). Real-time recognition of sign language gestures and air-writing using leap motion, *2017 Fifteenth IAPR International Conference on Machine Vision Applications (MVA)*.

[15] **Avola, D.**, **Bernardi, M.**, **Cinque, L.**, **Foresti, G.L. and Massaroni, C.** (2019). Exploiting Recurrent Neural Networks and Leap Motion Controller for the Recognition of Sign Language and Semaphoric Hand Gestures, *IEEE Transactions on Multimedia*, *21*(1), 234–245.

[16] **P.Karthick, N.Prathiba, V. and S.Thanalaxmi** (2014). Transforming Indian Sign Language into Text Using Leap Motion, *International Journal of Innovative Research in Science, Engineering and Technology*, volume 3, pp.10906–10910.

[17] **et al., C.Y.** (2016). Human–Robot Interaction Interface, *Advanced Technologies in Modern Robotic Applications*, pp.257–301.

[18] **Tao Wang, Xiaolong Cai, L.W. and Tian, H.** (2018). Interactive Design of 3D Dynamic Gesture Based on SVM-LSTM Model, *International Journal of Mobile Human Computer Interaction*, p. 15.

[19] **Naguri, C.R. and Bunescu, R.C.** (2017). Recognition of Dynamic Hand Gestures from 3D Motion Data Using LSTM and CNN Architectures, *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pp.1130–1133.

[20] **Kumar, P.**, **Saini, R.**, **Roy, P.P. and Dogra, D.P.** (2016). Study of Text Segmentation and Recognition Using Leap Motion Sensor, *IEEE Sensors Journal*.

[21] **Behera, S.**, **Dogra, D. and Roy, P.** (2017). Analysis of 3D signatures recorded using leap motion sensor, *Multimedia Tools and Applications 77*, p.14029–14054.

[22] **Mohandes, M.**, **Aliyu, S. and Deriche, M.** (2015). Prototype Arabic Sign language recognition using multi-sensor data fusion of two leap motion controllers, *2015 IEEE 12th International Multi-Conference on Systems, Signals & Devices (SSD15)*.

[23] **Fok, K.**, **Ganganath, N.**, **Cheng, C. and Tse, C.K.** (2015). A Real-Time ASL Recognition System Using Leap Motion Sensors, *2015 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery*, pp.411–414.

[24] **Miada A. Almasre, H.A.N.** (2016). A Real-Time Letter Recognition Model for Arabic Sign Language Using Kinect and Leap Motion Controller v2, *International Journal of Advanced Engineering, Management and Science*, *2*.

[25] **Marin, G.**, **Dominio, F. and Zanuttigh, P.** (2014). Hand gesture recognition with leap motion and kinect devices, *2014 IEEE International Conference on Image Processing (ICIP)*, pp.1565–1569.

[26] **Marin, G.**, **Dominio, F. and Zanuttigh.** (2015). Hand gesture recognition with jointly calibrated Leap Motion and depth sensor, *P. Multimed Tools Appl (2016)*.

[27] **Pradeep Kumar, Partha Pratim Roy, D.P.D.** (2018). Independent Bayesian classifier combination based sign language recognition using facial expression, *Information Sciences*, volume428, pp.30–48.

[28] **Pradeep Kumar, Himaanshu Gauba, P.P.R. and Dogra, D.P.** (2017). A multimodel framework for sensor based sign language recognition, *Neurocomputing*, volume259, pp.21–38.

[29] **Wenwen Yang, Jinxu Tao, Z.Y.** (2016). Continuous sign language recognition using level building based on fast hidden Markov model, *Pattern Recognition Letters*, pp.28–35.

[30] **Jiayi Li, Aman Shrestha, J.L.K. and Fioranelli, F.** (2019). From Kinect skeleton data to hand gesture recognition with radar, *The Journal of Engineering IET International Radar Conference (IRC 2018)*, pp.6914–6919.

[31] **Nada B. Ibrahim, Mazen M. Selim, H.H.Z.** (2018). An Automatic Arabic Sign Language Recognition System (ArSLRS), *Journal of King Saud University – Computer and Information Sciences*, volume 30, pp.470–477.

[32] **Kian Ming Lim, AlanW.C. Tan, S.T.** (2016). A feature covariance matrix with serial particle filter for isolated sign language recognition, *Expert Systems With Applications*, volume 54, pp.208–218.

[33] **Oscar Koller, Jens Forster, H.N.** (2015). Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers, *Computer Vision and Image Understanding*, volume141, pp.108–125.

[34] **Shin, S. and Sung, W.** (2016). Dynamic hand gesture recognition for wearable devices with low complexity recurrent neural networks, *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp.2274–2277.

[35] **Molchanov, P.**, **Yang, X.**, **Gupta, S.**, **Kim, K.**, **Tyree, S. and Kautz, J.** (2016). Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks, *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp.4207–4215.

[36] **Eckhardt, C.**, **Sullivan, J. and Pietroszek, K.** (2017). Flex: Hand Gesture Recognition Using Muscle Flexing Sensors, *Proceedings of the 5th Symposium on Spatial User Interaction*, SUI '17, Association for Computing Machinery, New York, NY, USA, p.164, `https://doi.org/10.1145/3131277.3134360`.

[37] **Ma, C.**, **Wang, A. and Chen, G.e.a.** (2018). Hand joints-based gesture recognition for noisy dataset using nested interval unscented Kalman filter with LSTM network, *The Visual Computer*, volume 34, p.1053–1063.

[38] **Potter, L.E.**, **Araullo, J. and Carter, L.** (2013). The Leap Motion controller: a view on sign language, *OzCHI '13: Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration*, pp.175–178.

[39] **Kooij, J.F.** (2016). SenseCap: Synchronized Data Collection with Microsoft Kinect2 and LeapMotion, *Proceedings of the 24th ACM International Conference on Multimedia*, MM '16, Association for Computing Machinery, New York, NY, USA, p.1218–1221, `https://doi.org/10.1145/2964284.2973805`.

[40] Güncel Türk İşaret Dili Sözlüğü, `http://tidsozluk.net/`, reference date: 2020-04-08.

APPENDICES

APPENDIX A.1 : Data Features Collected From *Leap Motion Controller*

APPENDIX A.1: Data Features Collected From *Leap Motion Controller*

Data features (columns) are as following (in order):

- Hand_Left_PalmNormal_X
- Hand_Left_PalmNormal_Y
- Hand_Left_PalmNormal_Z
- Hand_Left_PalmNormal_PITCH
- Hand_Left_PalmNormal_YAW
- Hand_Left_PalmNormal_ROLL
- Hand_Left_PalmVelocityl_X
- Hand_Left_PalmVelocity_Y
- Hand_Left_PalmVelocity_Z
- Hand_Left_PalmVelocity_PITCH
- Hand_Left_PalmVelocity_YAW
- Hand_Left_PalmVelocity_ROLL
- Hand_Left_Direction_X
- Hand_Left_Direction_Y
- Hand_Left_Direction_Z
- Hand_Left_Direction_PITCH
- Hand_Left_Direction_YAW
- Hand_Left_Direction_ROLL
- Hand_Left_Arm_Direction_X
- Hand_Left_Arm_Direction_Y
- Hand_Left_Arm_Direction_Z
- Hand_Left_Arm_Direction_PITCH
- Hand_Left_Arm_Direction_YAW
- Hand_Left_Arm_Direction_ROLL
- Hand_Left_Finger_TYPE_THUMB_IsExtended

- Hand_Left_Finger_TYPE_THUMB_Direction_X
- Hand_Left_Finger_TYPE_THUMB_Direction_Y
- Hand_Left_Finger_TYPE_THUMB_Direction_Z
- Hand_Left_Finger_TYPE_THUMB_Direction_PITCH
- Hand_Left_Finger_TYPE_THUMB_Direction_YAW
- Hand_Left_Finger_TYPE_THUMB_Direction_ROLL
- Hand_Left_Finger_TYPE_THUMB_TipVelocity_X
- Hand_Left_Finger_TYPE_THUMB_TipVelocity_Y
- Hand_Left_Finger_TYPE_THUMB_TipVelocity_Z
- Hand_Left_Finger_TYPE_THUMB_TipVelocity_PITCH
- Hand_Left_Finger_TYPE_THUMB_TipVelocity_YAW
- Hand_Left_Finger_TYPE_THUMB_TipVelocity_ROLL
- Hand_Left_Finger_TYPE_INDEX_IsExtended
- Hand_Left_Finger_TYPE_INDEX_Direction_X
- Hand_Left_Finger_TYPE_INDEX_Direction_Y
- Hand_Left_Finger_TYPE_INDEX_Direction_Z
- Hand_Left_Finger_TYPE_INDEX_Direction_PITCH
- Hand_Left_Finger_TYPE_INDEX_Direction_YAW
- Hand_Left_Finger_TYPE_INDEX_Direction_ROLL
- Hand_Left_Finger_TYPE_INDEX_TipVelocity_X
- Hand_Left_Finger_TYPE_INDEX_TipVelocity_Y
- Hand_Left_Finger_TYPE_INDEX_TipVelocity_Z
- Hand_Left_Finger_TYPE_INDEX_TipVelocity_PITCH
- Hand_Left_Finger_TYPE_INDEX_TipVelocity_YAW
- Hand_Left_Finger_TYPE_INDEX_TipVelocity_ROLL
- Hand_Left_Finger_TYPE_MIDDLE_IsExtended
- Hand_Left_Finger_TYPE_MIDDLE_Direction_X
- Hand_Left_Finger_TYPE_MIDDLE_Direction_Y
- Hand_Left_Finger_TYPE_MIDDLE_Direction_Z

- Hand_Left_Finger_TYPE_MIDDLE_Direction_PITCH
- Hand_Left_Finger_TYPE_MIDDLE_Direction_YAW
- Hand_Left_Finger_TYPE_MIDDLE_Direction_ROLL
- Hand_Left_Finger_TYPE_MIDDLE_TipVelocity_X
- Hand_Left_Finger_TYPE_MIDDLE_TipVelocity_Y
- Hand_Left_Finger_TYPE_MIDDLE_TipVelocity_Z
- Hand_Left_Finger_TYPE_MIDDLE_TipVelocity_PITCH
- Hand_Left_Finger_TYPE_MIDDLE_TipVelocity_YAW
- Hand_Left_Finger_TYPE_MIDDLE_TipVelocity_ROLL
- Hand_Left_Finger_TYPE_RING_IsExtended
- Hand_Left_Finger_TYPE_RING_Direction_X
- Hand_Left_Finger_TYPE_RING_Direction_Y
- Hand_Left_Finger_TYPE_RING_Direction_Z
- Hand_Left_Finger_TYPE_RING_Direction_PITCH
- Hand_Left_Finger_TYPE_RING_Direction_YAW
- Hand_Left_Finger_TYPE_RING_Direction_ROLL
- Hand_Left_Finger_TYPE_RING_TipVelocity_X
- Hand_Left_Finger_TYPE_RING_TipVelocity_Y
- Hand_Left_Finger_TYPE_RING_TipVelocity_Z
- Hand_Left_Finger_TYPE_RING_TipVelocity_PITCH
- Hand_Left_Finger_TYPE_RING_TipVelocity_YAW
- Hand_Left_Finger_TYPE_RING_TipVelocity_ROLL
- Hand_Left_Finger_TYPE_PINKY_IsExtended
- Hand_Left_Finger_TYPE_PINKY_Direction_X
- Hand_Left_Finger_TYPE_PINKY_Direction_Y
- Hand_Left_Finger_TYPE_PINKY_Direction_Z
- Hand_Left_Finger_TYPE_PINKY_Direction_PITCH
- Hand_Left_Finger_TYPE_PINKY_Direction_YAW
- Hand_Left_Finger_TYPE_PINKY_Direction_ROLL

- Hand_Left_Finger_TYPE_PINKY_TipVelocity_X
- Hand_Left_Finger_TYPE_PINKY_TipVelocity_Y
- Hand_Left_Finger_TYPE_PINKY_TipVelocity_Z
- Hand_Left_Finger_TYPE_PINKY_TipVelocity_PITCH
- Hand_Left_Finger_TYPE_PINKY_TipVelocity_YAW
- Hand_Left_Finger_TYPE_PINKY_TipVelocity_ROLL
- Hand_Right_PalmNormal_X
- Hand_Right_PalmNormal_Y
- Hand_Right_PalmNormal_Z
- Hand_Right_PalmNormal_PITCH
- Hand_Right_PalmNormal_YAW
- Hand_Right_PalmNormal_ROLL
- Hand_Right_PalmVelocityl_X
- Hand_Right_PalmVelocity_Y
- Hand_Right_PalmVelocity_Z
- Hand_Right_PalmVelocity_PITCH
- Hand_Right_PalmVelocity_YAW
- Hand_Right_PalmVelocity_ROLL
- Hand_Right_Direction_X
- Hand_Right_Direction_Y
- Hand_Right_Direction_Z
- Hand_Right_Direction_PITCH
- Hand_Right_Direction_YAW
- Hand_Right_Direction_ROLL
- Hand_Right_Arm_Direction_X
- Hand_Right_Arm_Direction_Y
- Hand_Right_Arm_Direction_Z
- Hand_Right_Arm_Direction_PITCH
- Hand_Right_Arm_Direction_YAW

- Hand_Right_Arm_Direction_ROLL
- Hand_Right_Finger_TYPE_THUMB_IsExtended
- Hand_Right_Finger_TYPE_THUMB_Direction_X
- Hand_Right_Finger_TYPE_THUMB_Direction_Y
- Hand_Right_Finger_TYPE_THUMB_Direction_Z
- Hand_Right_Finger_TYPE_THUMB_Direction_PITCH
- Hand_Right_Finger_TYPE_THUMB_Direction_YAW
- Hand_Right_Finger_TYPE_THUMB_Direction_ROLL
- Hand_Right_Finger_TYPE_THUMB_TipVelocity_X
- Hand_Right_Finger_TYPE_THUMB_TipVelocity_Y
- Hand_Right_Finger_TYPE_THUMB_TipVelocity_Z
- Hand_Right_Finger_TYPE_THUMB_TipVelocity_PITCH
- Hand_Right_Finger_TYPE_THUMB_TipVelocity_YAW
- Hand_Right_Finger_TYPE_THUMB_TipVelocity_ROLL
- Hand_Right_Finger_TYPE_INDEX_IsExtended
- Hand_Right_Finger_TYPE_INDEX_Direction_X
- Hand_Right_Finger_TYPE_INDEX_Direction_Y
- Hand_Right_Finger_TYPE_INDEX_Direction_Z
- Hand_Right_Finger_TYPE_INDEX_Direction_PITCH
- Hand_Right_Finger_TYPE_INDEX_Direction_YAW
- Hand_Right_Finger_TYPE_INDEX_Direction_ROLL
- Hand_Right_Finger_TYPE_INDEX_TipVelocity_X
- Hand_Right_Finger_TYPE_INDEX_TipVelocity_Y
- Hand_Right_Finger_TYPE_INDEX_TipVelocity_Z
- Hand_Right_Finger_TYPE_INDEX_TipVelocity_PITCH
- Hand_Right_Finger_TYPE_INDEX_TipVelocity_YAW
- Hand_Right_Finger_TYPE_INDEX_TipVelocity_ROLL
- Hand_Right_Finger_TYPE_MIDDLE_IsExtended
- Hand_Right_Finger_TYPE_MIDDLE_Direction_X

- Hand_Right_Finger_TYPE_MIDDLE_Direction_Y
- Hand_Right_Finger_TYPE_MIDDLE_Direction_Z
- Hand_Right_Finger_TYPE_MIDDLE_Direction_PITCH
- Hand_Right_Finger_TYPE_MIDDLE_Direction_YAW
- Hand_Right_Finger_TYPE_MIDDLE_Direction_ROLL
- Hand_Right_Finger_TYPE_MIDDLE_TipVelocity_X
- Hand_Right_Finger_TYPE_MIDDLE_TipVelocity_Y
- Hand_Right_Finger_TYPE_MIDDLE_TipVelocity_Z
- Hand_Right_Finger_TYPE_MIDDLE_TipVelocity_PITCH
- Hand_Right_Finger_TYPE_MIDDLE_TipVelocity_YAW
- Hand_Right_Finger_TYPE_MIDDLE_TipVelocity_ROLL
- Hand_Right_Finger_TYPE_RING_IsExtended
- Hand_Right_Finger_TYPE_RING_Direction_X
- Hand_Right_Finger_TYPE_RING_Direction_Y
- Hand_Right_Finger_TYPE_RING_Direction_Z
- Hand_Right_Finger_TYPE_RING_Direction_PITCH
- Hand_Right_Finger_TYPE_RING_Direction_YAW
- Hand_Right_Finger_TYPE_RING_Direction_ROLL
- Hand_Right_Finger_TYPE_RING_TipVelocity_X
- Hand_Right_Finger_TYPE_RING_TipVelocity_Y
- Hand_Right_Finger_TYPE_RING_TipVelocity_Z
- Hand_Right_Finger_TYPE_RING_TipVelocity_PITCH
- Hand_Right_Finger_TYPE_RING_TipVelocity_YAW
- Hand_Right_Finger_TYPE_RING_TipVelocity_ROLL
- Hand_Right_Finger_TYPE_PINKY_IsExtended
- Hand_Right_Finger_TYPE_PINKY_Direction_X
- Hand_Right_Finger_TYPE_PINKY_Direction_Y
- Hand_Right_Finger_TYPE_PINKY_Direction_Z
- Hand_Right_Finger_TYPE_PINKY_Direction_PITCH

- Hand_Right_Finger_TYPE_PINKY_Direction_YAW
- Hand_Right_Finger_TYPE_PINKY_Direction_ROLL
- Hand_Right_Finger_TYPE_PINKY_TipVelocity_X
- Hand_Right_Finger_TYPE_PINKY_TipVelocity_Y
- Hand_Right_Finger_TYPE_PINKY_TipVelocity_Z
- Hand_Right_Finger_TYPE_PINKY_TipVelocity_PITCH
- Hand_Right_Finger_TYPE_PINKY_TipVelocity_YAW
- Hand_Right_Finger_TYPE_PINKY_TipVelocity_ROLL

CURRICULUM VITAE

Name Surname: Burçak Demircioğlu Kam

Place and Date of Birth: 24.11.1991 Bornova

E-Mail: burcakdemircioglu@gmail.com

EDUCATION:

- **B.Sc.:** 2015, Istanbul Technical University, Computer and Informatics Engineering, Computer Engineering

PROFESSIONAL EXPERIENCE AND REWARDS:

- 2019-Present Software Engineer at Microsoft
- 2016-2019 Software Engineer at Hive Bilişim ve Yazılım A.Ş.
- 2016 3rd Place in Enterprising and Innovative Graduation Design Project Competition of İTÜ ARI Teknokent

PUBLICATIONS, PRESENTATIONS AND PATENTS ON THE THESIS:

- B. Demircioğlu Kam, and H. Kose, "A New Data Collection Interface for Dynamic Gesture and Sign Recognition with Leap Motion Sensor," *PUDCAD Game+Design Education International Conference*, Istanbul, 24-26 June,2020.