ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

TRANSFORMING FEEDBACK CONTROL SYSTEMS ON WHITEBOARD INTO MATLAB VIA A DEEP LEARNING BASED INTELLIGENT SYSTEM

M.Sc. THESIS

Dorukhan ERDEM

Department of Control and Automation Engineering

Control and Automation Engineering Programme

JULY 2020

ISTANBUL TECHNICAL UNIVERSITY ★ GRADUATE SCHOOL OF SCIENCE ENGINEERING AND TECHNOLOGY

TRANSFORMING FEEDBACK CONTROL SYSTEMS ON WHITEBOARD INTO MATLAB VIA A DEEP LEARNING BASED INTELLIGENT SYSTEM

M.Sc. THESIS

Dorukhan ERDEM
(504181110)

Department of Control and Automation Engineering

Control and Automation Engineering Programme

Thesis Advisor: Assoc. Prof. Tufan KUMBASAR

JULY 2020

ISTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

DERİN ÖĞRENME TABANLI AKILLI BİR SİSTEM İLE BEYAZ TAHTADAKİ GERİBESLEMELİ KONTROL SİSTEMLERİNİN MATLAB ORTAMINA AKTARILMASI

YÜKSEK LİSANS TEZİ

Dorukhan ERDEM
(504181110)

Kontrol ve Otomasyon Mühendisliği Anabilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Doç. Dr. Tufan KUMBASAR

TEMMUZ 2020

Dorukhan Erdem, a M.Sc. student of İTU Graduate School of Science Engineering and Technology student ID 504181110, successfully defended the thesis entitled "Transforming Feedback Control Systems on Whiteboard into Matlab via A Deep Learning Based Intelligent System", which he prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Tufan KUMBASAR** İstanbul Technical University	.....
Jury Members :	**Assist. Prof. İlker ÜSTOĞLU** Istanbul Technical University	.....
	Assist. Prof. Gürkan SOYKAN Bahçeşehir University	.....

Date of Submission : 7 July 2020
Date of Defense : 14 July 2020

To my family,

FOREWORD

I would like to thank my thesis advisor Assoc. Prof. Dr Tufan Kumbasar for his guidance and everlasting support. I am grateful to him for making me part of the Artificial Intelligent and Intelligent Systems Laboratory team.

I would also like to thank Scientific and Technological Research Council of Turkey (TUBITAK) for their aid under the research project 118E807.

Finally, i would like to thank Gamze for standing by my side through thick and thin and thank my family for their unending support.

July 2020 Dorukhan ERDEM

TABLE OF CONTENTS

Page

ABBREVIATIONS

FCA : Feedback Control Architecture
TF : Transfer Function
LQR : Linear Quadratic Control
IMC : Internal Model Control
DL : Deep Learning
CNN : Convolutional Neural Networks
HFCA : Handwritten FCA
PR : Pattern Recognition
NN : Neural Network
ML : Machine Learning
AI : Artificial Intelligence
SL : Supervised Learning
GD : Gradient Descent
MSE : Mean Square Error
SGD : Stochastic Gradient Descent
FP : Forward Propagation
BP : Backward Propagation
BN : Batch Normalization
RGB : Red Green Blue
HFCAR : HFCA Recognition
HCR : Handwritten Character Recognition
BB : Bounding Box

SYMBOLS

x_i : Input of ith layer
w_i : Weight of ith layer
b_i : Bias term of ith layer
z_i : Activation of ith layer
y_i : Output of ith layer
g_i : Activation function for ith layer
α : Learning rate
$\hat{y}$: Predicted output of a NN
θ : Vector containing weights and biases of a neural network
λ : Regularization constant
v : Momentum term in GD with momentum
$\hat{x}_i$: Normalized inputs of ith layer
μ_b : Batch mean
σ_b : Batch standard deviation
γ : Scaling constant in BN
β : Shifting constant in BN
H_f : Height of filter in 2D convolution
W_f : Width of filter in 2D
H_x : Height of the input image in convolution
W_x : Width of the input image in convolution
C : Number of channels of a tensor
F : Number of filters in a convolutional layer
H_z : Output height in convolutional layer
W_z : Output width
P : Padding amount in a conv
S : Stride amount in a convolutional or pooling layr
Q : Total number of classes in a classification network
D : Binarization threshold
C : Circularity measure
R : Rectangularity measure
L_{major} : Length of major axis of a region
L_{minor} : Length of minor axis of a region

LIST OF TABLES

Page

LIST OF FIGURES

Page

TRANSFORMING FEEDBACK CONTROL SYSTEMS ON WHITEBOARD INTO MATLAB VIA A DEEP LEARNING BASED INTELLIGENT SYSTEM

SUMMARY

In control theory, some concepts can be better grasped with further visualization. Some core concepts such as the transient and steady-state behavior of a system or the interval at which it is stable can be better expressed using plots in time and frequency domains. The introduction of these visual aids has an important place in control lectures. In the classroom environment, instructors usually draw the corresponding graphs about a system by hand after presenting its control diagram. Hand drawing a plot is a difficult and time-consuming process and it is not possible to correctly scaled graphs this way. In order to correctly analyze a system, a simulation environment like Matlab® should be preferred. In this thesis, we have proposed a deep learning based pipeline that can recognize and transfer a block diagram on the whiteboard to the Matlab® environment. This way, control systems drawn by the lecturers can be properly analyzed with computer-generated plots without redefining them in the Matlab®.

Generally, feedback control architectures are covered in control system lectures. These architectures can be visually represented with block diagrams. In a block diagram, transfer blocks are shown as rectangles and the connections between these blocks are indicated with arrows. The arithmetic operations like summation and subtraction between the signals are also shown inside small circles. A feedback control system may contain transfer functions for the filter or noise dynamics along with the controller and the plant. The number and the location of these transfer functions, their connection schemes, the number of forward or backward paths separates an architecture from another. In order to transfer a handwritten block diagram to Matlab® these informations must be extracted first using computer vision techniques. Then, the contents of each transfer function must be found in order to represent them in Matlab®. An architecture with known connections and transfer functions can be easily expressed in the Matlab® environment. If desired, a simulation model can be created Simulink™ or analyzed with Control System Designer App™.

The control architecture from a given picture can be found with different approaches. A possible one is to handle the recognition task as a classification problem. In classification problems, inputs are assigned to one of the pre-defined classes by calculating a similarity ratio. In the context of feedback control architecture recognition, frequently used architectures can be selected as the class set, and the corresponding class can be found by examining the input image. Classification problems usually require a feature extraction method. These methods transfer the inputs to a feature space and the classification is done between the points in that space. The accuracy of the classification task depends on the selected feature extraction method and finding a proper method is the main challenge in classification problems. An ideal feature extraction method should not be affected by the uncertainties in the input image. Lighting condition is among the most common uncertainties in image processing problems. The intensity of the light can cause dark areas to appear brighter

and can cause information loss. Also, the quality of the lecturer's handwriting quality can cause further uncertainties in our handwritten architecture recognition problem. Long sessions may deteriorate the lecturer's handwriting quality or different handwritings among various lecturers can harm the generalizability of the classification model. Various feature extraction algorithms exist in the computer vision literature, but a majority of them are not robust to transformations or noises in the input images. Therefore, structures that automatically extract features such as neural networks are preferred.

In this thesis, recognition of a handwritten feedback control architecture is handled as a classification task where the classes are the 6 feedback control architectures defined in the Control System Designer App™. In order to accomplish the classification task, we have trained a deep Convolutional Neural Network (CNN). For the training, we have collected images of handwritten feedback architectures with the help of various lecturers. Although we have gathered as many samples as we can, constructing a dataset is a time-consuming process. To avoid problems that may arise from the lack of data, we have adopted the transfer learning approach. With the transfer learning approach, a deep network trained on large datasets like ImageNet can be adapted to new classification tasks by replacing its last layers. As the base network for our CNN, we have selected ResNet-50 since it shows high classification accuracies in benchmark tests and doesn't suffer from a common problem called gradient vanishing.

The next step in our deep learning based pipeline is to construct the transfer functions. To create the mathematical expressions of the transfer functions, characters in the input images must first be recognized. Besides, we need to be able to determine which characters belong to which transfer function. If the location and the dimensions of each transfer function block in the input image are known, the corresponding characters can be searched inside these regions. Considering the transfer functions are represented as rectangles in block diagrams, we continued our pipeline with a rectangle detection step in this context. First, the input image is binarized with a thresholding algorithm. Possible noises caused by illuminations are then filtered from the binary image. After the filtering operation, an edge detection algorithm is applied and the outermost edge is removed from the binary image. With a filling method that fills closed contours, we ended up with regions as candidate rectangles. Using the connected component analysis and inspecting the rectangularity of each region, actual rectangles and their locations are found. At this point, the architecture and the transfer functions it contains are already known. Starting from the upper-left one, rectangles are then matched with the transfer functions by comparing their locations.

Following the rectangle detection step, regions belonging to the transfer functions are cropped from the filtered binary image. Then the characters are detected in these cropped regions using the connected component analysis. In order to label the detected characters, we have trained another CNN using a pre-trained ResNet-50 as a base network. There are 20 possible labels for a character consisting of the arithmetic operators, parentheses, digits, and the 's' character. We have also constructed a second dataset for the character classification task.

After detecting the transfer function blocks and recognizing the characters they contain, symbolic expressions for each transfer function can be constructed. A transfer function can be defined in Matlab® by finding the coefficients of its numerator and denominator. The coefficients can easily found from the symbolic expressions with the functions provided by Matlab®. For this purpose, the characters in each block are

first sorted by their horizontal positions. Using the vertical position of the character that represents the fraction sign, the remaining ones are then separated as numerator and denominator. By concatenating the labels of the ordered characters, equation strings are generated. From the equation strings, symbolic expressions are constructed and the required coefficients are found. Thus, all the information necessary to transfer the feedback architecture drawn on the board to the Matlab® environment was obtained.

At the end of the deep learning based pipeline, the recognized architecture can easily be transferred to the Control System Designer App™. The application requires the type of the feedback control architecture and the expressions of the transfer functions to create a new analysis, which is already present at this point. Alternatively, a Simulink™ model can be generated by defining the connections and transfer functions. To test the real-time capabilities of our pipeline, we finally created a test application that uses our pipeline. It processes images gathered from a live camera and creates a Simulink™ model if it detects a suitable feedback control architecture.

DERİN ÖĞRENME TABANLI AKILLI BİR SİSTEM İLE BEYAZ TAHTADAKİ GERİBESLEMELİ KONTROL SİSTEMLERİNİN MATLAB ORTAMINA AKTARILMASI

ÖZET

Kontrol teorisinde bazı konular, daha iyi kavranabilmeleri için görsel metotlara ihtiyaç duyar. Bir sistemin geçici ve sürekli haldeki davranışı veya hangi aralıklarda kararlı olduğu gibi temel bilgiler görsel zaman veya frekans tanım bölgelerinde oluşturulan grafiklerle gösterildiğinde daha iyi ifade edilebilmektedir. Ayrıca köklerin yer eğrisi veya Bode grafikleri gibi zaman ve frekans bölgesindeki görsel yöntemler ile sistemlerin tasarımı da yapılabilmektedir. Kontrol derslerinde de bu görsel araçların tanıtımı önemli bir yer tutmaktadır. Sınıf ortamında işlenen derslerde eğitmenler blok şeması verilen bir sistem ile ilgili grafikleri tahtaya yaklaşık bir şekilde çizmektedirler. Bu hem zaman alan zor bir süreçtir hem de ölçeksiz çizilen grafikler yüzünden kavramların anlaşılmasını güçleştirmektedir. Ele alınan sistem hakkında doğru bir şekilde yorum yapabilmek için Matlab® gibi bir benzetim ortamı kullanılmalıdır. Bu tez kapsamında, tahtaya çizilmiş kontrol şemalarını tanıyan ve Matlab® ortamına aktarabilen derin öğrenme tabanlı bir yöntem önerilmiştir. Önerilen bu yöntem derin öğrenme ve görüntü işleme yöntemlerini kullanmakta ve kontrol şemalarını gerçek zamanda bilgisayar ortamına aktarabilmektir. Böylece ders esnasında eğitmen tarafından tahtaya çizilen kontrol mimarilerinin sistem yanıtı, kontrolör işareti gibi grafikler kolaylıkla ve doğru bir şekilde elde edilebilmektedir. Üstelik tahtadaki mimarinin Matlab® ortamında bir kez daha oluşturulmasına gerek kalmamaktadır. Önerilen yöntem gerçek zamanlı çalıştığı için el ile çizilmiş şemadaki herhangi bir değişiklik de benzetim ortamında görülebilmektedir. Bu şekilde sistemin kutup ve sıfırları güncellenerek sistem cevabı üzerindeki etkileri kolaylıkla görülebilmekte veya farklı kontrolör parametrelerinin ayarı yapılabilmektedir.

Kontrol sistemi derslerinde genellikle geri beslemeli kontrol mimarileri ele alınmaktadır. Bu mimariler blok şemalarıyla görsel olarak da ifade edilebilmektedir. Bir kontrol şemasında transfer fonksiyonları dörtgenler ile temsil edilmekte ve oklar aracılığıyla birbirilerine bağlanmaktadır. Ayrıca sinyaller arasındaki toplama ve çıkarma gibi aritmetik işlemler de daireler içerisinde gösterilmektedir. Geri beslemeli bir kontrol yapısında kontrol edilmek istenen sistemin ve kontrolörlerin yanında sensor dinamiklerini ve gürültü filtrelerini temsil eden transfer fonksiyonları da bulunabilmektedir. Belirtilen transfer fonksiyonlarının konumları, aralarındaki bağlantılar, ileri ve geri yolların sayıları gibi etkenler herhangi bir mimariyi bir diğerinden ayırmaktadır. Tahtaya çizilmiş bir blok şemasını Matlab® ortamına aktarılabilmesi için öncelikle şemanın bilgisayar tarafından algılanması gerekmektedir. Bir kamera ile görüntüsü kaydedilen bir blok şemasından, görüntü işleme yöntemleri kullanılarak gerekli bilgiler elde edilebilir ve bu şema bilgisayar ortamında yeniden oluşturulabilir. Bunu başarabilmek için öncelikle şemada belirtilen mimarinin yapısı oluşturulmalıdır. Mimarinin kaç tane transfer bloğu içerdiği,

aralarındaki bağlantı yapıları gibi bilgilerin bilinmesi daha sonraki adımlarını da kolaylaştıracaktır. Bir sonraki aşamada ise her bir transfer bloğunun içeriği elde edilmeli ve transfer fonksiyonları benzetim ortamında oluşturulmalıdır. Bağlantıları ve transfer fonksiyonları bilinen bir mimari kolaylıkla Matlab® ortamında ifade edilebilmektedir. İstenirse Simulink™ ile benzetim modeli oluşturulabilmekte veya kontrol sistem tasarım uygulaması (Control System Designer App™) ile analizi yapılabilmektedir.

Verilen bir resimden kontrol mimarisi farklı yaklaşımlarla bulunabilir. Bunlardan bir tanesi, problemi bir sınıflandırma görevi olarak ele almaktır. Sınıflandırma problemlerinde girdiler benzerlik oranlarına göre önceden belirlenmiş sınıflardan bir tanesine atanırlar. Kontrol mimarisi tespiti bağlamında da sıklıkla kullanılan mimariler önceden belirlenebilir ve blok şemasının resmi incelenerek hangisine en çok benzediği bulunabilir. Sınıflandırma problemleri genellikle bir özellik çıkarma yöntemine ihtiyaç duyar. Özellik çıkarma yöntemleri girdileri bir özellik uzayına aktarır ve sınıflandırma işlemi bu uzaydaki noktalar arasında yapılır. Sınıflandırma işleminin başarısı, kullanılan özellik çıkarma yöntemine dayanmaktadır ve uygun bir özellik çıkarma yöntemi bulmak sınıflandırma problemlerinin başlıca zorluğudur. İdeal bir özellik çıkarma yönteminin resimdeki belirsizliklerden etkilenmemesi gerekir. Görüntü işleme problemlerinde ortam ışığı en çok karşılaşılan belirsizlikler arasındadır. Işığın şiddeti yer yer parlamalarla koyu bölgelerin aydınlık gözükmesine ve ilgilenen objenin kaybolmasına neden olabilir. Ele alınan mimari tanıma probleminde ise ışıktan kaynaklanan gürültülerin yanı sıra eğitmenin yazı kalitesi de belirsizlik yaratabilmektedir. Eğitmenin yazısı bir süre sonra bozulabilmekte veya eğitmenler arasında karakter tipleri farklılık gösterebilmektedir. Görüntü işleme literatüründe farklı özellik çıkarma algoritmaları bulunsa da bu yöntemler gürültülere karşı dayanıksız kalmakta ya da telif hakkıyla korunmaktadırlar. Bu yüzden nöral ağlar gibi otomatik özellik çıkaran yapılar tercih edilmektedir.

Bu tez kapsamında mimari tespiti bir sınıflandırma problemi olarak ele alınmış ve kontrol derslerinde sıklıkla kullanılan 6 farklı mimari belirlenmiştir. Ayrıca belirtilmelidir ki kontrol sistem tasarım uygulaması da yine bu 6 farklı mimariye ait yapılarla çalışabilmektedir. Sınıflandırma görevinde kullanılmak üzere derin bir Evreşimli Sinir Ağı (CNN) eğitilmiştir. Görüntü işleme problemlerinde CNN yapıları Eğitim için farklı eğitmenlerden toplanan ve belirlenen 6 mimarinin sınıf ortamındaki beyaz tahta çizimlerini içeren bir veri seti oluşturulmuştur. Her ne kadar toplanan veri sayısı yüksek tutulmaya çalışılsa da geniş bir veri seti oluşturmak zaman almaktadır. Veri azlığından kaynaklanabilecek sorunları aşabilmek için öğrenme aktarımı yaklaşımı tercih edilmiştir. Öğrenme aktarımı ile ImageNet gibi büyük veri setlerinde eğitilmiş bir ağ, son katmanları değiştirilerek yeni veri setlerine uyarlanabilmektedir. Sınıflandırma görevinde kullanılan ağ ise önceden eğitilmiş bir ResNet-50 ağına dayanmaktadır.

Blok şeması resmindeki mimari belirlendikten sonraki aşama transfer fonksiyonlarının elde edilmesidir. Bir transfer fonksiyonunun matematiksel ifadesini elde edebilmek için öncelikle resimdeki karakterlerin tanınması ve konumlarının bulunması gerekmektedir. Ayrıca hangi karakterlerin hangi transfer fonksiyonuna ait oldukları da belirlenmelidir. Transfer fonksiyonu bloklarının konumu ve büyüklükleri önceden belirlenebilirse ilgili karakterler sadece bu bloklar içerisinde aranabilir. Kontrol sistemlerinin blok şemalarında transfer fonksiyonlarının dörtgenler içerisinde ifade edildiği göz önünde bulundurularak mimari tespitinden sonra resimdeki dörtgenler aranmıştır. Dörtgenlerin tespiti için öncelikle girdi resminin ikili hali elde edilmiştir.

Işıktan kaynaklanan belirsizlikler etkisini en çok ikili resmin elde edilmesinde göstermekte ve istenmeyen gürültüler üretmektedir. Bu gürültüler, klasik görüntü işleme yöntemleri kullanılarak mümkün olabildiğince giderilmeye çalışılmış ve sadece şemaya ait piksellerin pozitif olduğu ikili resmin oluşturulması amaçlanmıştır. Takip eden adımlarda kenarlar tespit edilmiş, en geniş çevre silinmiş ve kalan kapalı çevreler doldurulmuştur. Böylece birbirinden ayrı ve dörtgen olma olasılığı yüksek pozitif bölgeler elde edilmiştir. Bağlı bileşen analizi ile kalan bölgelerin dörtgenlikleri belirlenmiş ve transfer fonksiyonu blokları tespit edilmiştir. Bağlı bileşen analizinde bir bölgenin konumu ve boyutları bulunabildiği için resimdeki transfer fonksiyonlarının da konum ve boyutları belirlenebilmiştir. Ayrıca, bir kontrol mimarisinde sistemin, kontrolörün veya filtrelerin birbirilerine göre konumları bilinmektedir. Bu sayede dörtgen tespiti ile elde edilen bölgelerin mimarideki hangi transfer bloğuna karşılık geldiği en sol üstteki dörtgenden başlayarak bulunabilmektedir.

Dörtgen tespitinin ardından belirlenen her bir transfer fonksiyonu bloğunun bulunduğu bölge, gürültülerden arındırılmış ikili resimden kırpılmıştır. Kırpılan bu bölgelerdeki karakterler yine bağlı bileşen analizi kullanılarak tespit edilmiştir. Bu aşamada yalnızca karakterlere ait bileşenler bilinmekte, hangi karakter oldukları bilinmemektedir. Her bir blok içerisindeki karakterler bulunduktan sonra yine bir CNN kullanılarak bu karakterler etiketlenmiştir. Aritmetik operasyon sembolleri, parantezler, rakamlar ve ‘s’ karakterini içeren 20 sınıflık bir sınıflandırma problemini çözen bu ağ da ResNet-50 ile öğrenme aktarımı yöntemini kullanmaktadır. Eğitim için oluşturulan veri seti, mimari tanıma problemi için oluşturulmuş çizimlerden ayıklanmış karakter resimleri kullanılmıştır. Sınıflar arası eşit bir dağılım olması için eksik kalan sınıflar yine farklı eğitmenlerden toplanmış örneklerle tamamlanmıştır.

Transfer fonksiyonu blokları ve içerdikleri karakterler tespit edildikten sonra transfer fonksiyonlarının Matlab® ortamında sembolik ifadeleri elde edilmiştir. Matlab® ortamında bir transfer fonksiyonunu oluşturabilmek için pay ve payda polinomlarının katsayılarının bulunması gerekmektedir. Sembolik ifadesi elde edilen bir polinomun katsayıları yine Matlab® aracılığı ile kolaylıkla bulunabilir. Bu amaçla karakterler öncelikle yatay konumlarına göre sıralanmıştır. Ardından, pay ve payda polinomlarını ayıran kesir çizgisinin düşeydeki konumu kullanılarak karakterler pay ve payda olacak şekilde iki kümede toplanmıştır. Karakterler yatayda sıralı oldukları için pay ve paydayı temsil eden karakter katarları, karakterlerin etiketleri peşi sıra eklenerek oluşturulmuştur. Karakter katarları yaratılırken üslü sayılar da komşu iki karakter arasındaki düşey konum farkına bakılarak tespit edilmiştir. Karakter katarından sembolik polinom oluşturabilen bir fonksiyon tanımlanmış ve mimarideki transfer fonksiyonları bu sembolik ifadelerin katsayıları aracılığı ile oluşturulmuştur. Böylece tahtaya çizilmiş geri beslemeli mimariyi Matlab® ortamına aktarmak için gerekli bütün bilgiler elde edilmiştir.

Geri beslemeli bir kontrol mimarisini, kontrol sistem tasarım uygulamasına aktarmak oldukça kolaydır. Tek yapılması, gereken uygulama bünyesinde tanımlanmış 6 farklı mimariden birini seçmek ve her bir transfer fonksiyonunun ifadesini belirlemektir. Çizilen bir blok şemasını Simulink™ ortamına aktarabilmek için ise öncelikle 6 farklı model oluşturulmuştur. Bu modellerde gerekli transfer fonksiyonları birim olarak tanımlanmış ve kontrol sistem tasarım uygulamasında belirtildiği gibi isimlendirilmiştir. Tanınan kontrol blok şeması Simulink™ ortamına aktarılırken için ilgili model dosyası açılır ve isim eşleştirme ile transfer fonksiyonlarının gerçek değerleri modele aktarılır. Önerilen derin öğrenme tabanlı yöntemin gerçek zaman

performansını ölçmek için ise son olarak bir test uygulaması oluşturulmuştur. Bir sınıftaki beyaz tahtaya sabitlenmiş bir kameradan canlı alınan görüntüler gerçek zamanlı bir şekilde işlenerek çizilen geri beslemeli kontrol mimarilerinin Simulink™ ortamına aktarılması amaçlanmıştır. Ayrıca herhangi bir transfer bloğunda yapılan değişiklikler de takip edilerek modelin güncellenmesi sağlanmıştır.

1. INTRODUCTION

Teaching control theory is difficult as there are many theoretical concepts to be addressed. The main difficulty that students face is visualizing and understanding the relationship between the time and frequency domain parameters of a control system [1-3]. Therefore, the visualization of control systems is crucial to demonstrate the role of mathematics in control system design [1]. As this problem is not new, various approaches have been proposed to provide innovative techniques to enhance the students' motivation and improve their comprehension of control theory. For instance, interactive software tools are presented for teaching control systems in [4-8]. Besides, remote & virtual control laboratories are developed to provide students with a hands-on experience of control systems [9-12].

In most of the control system design courses, the main focus is usually on Feedback Control Architectures (FCAs) that are composed of controllers and Transfer Functions (TFs) structured within single/multi loop configurations [13]. In Figure 1.1, the most commonly handled FCAs are shown and their descriptions are provided in Table 1.1. The teaching approach to control system design is usually performed in a threefold approach. Firstly, the lecturer defines one of the FCA (shown in Figure 1.1) and then analyses it in the time and/or frequency domain. Finally, the lecturer provides the students with the theoretical background on controller design approaches such as graphical (Bode and Root-Locus plots) or automatic (LQR and IMC tuning) tuning methods [13]. To design and analyze FCAs, Matlab® provides an excellent environment; especially the Simulink™ and the Control System Toolbox™ [14]. The FCAs that are shown in Figure 1.1 can be easily analyzed and designed via the user interface of the Control System Designer™ as they built-in structures.

Teaching control system design is usually performed in an old-fashion style with a whiteboard. The lecturer basically defines one of the FCAs on the whiteboard as shown in Figure 2. Although whiteboards are easy to use, we believe that this approach has its disadvantages. First of all, control systems can not be accurately visualized with hand-drawn plots. It may even be impossible for high-order control systems. Scaled

and accurate plots are vital to show how a parameter affects the overall system dynamics.

Table 1.1 : Descriptions of FCA's defined in [14].

FCA	Descriptions
FCA-1	• Compensator (C(s)) and plant (G(s)) in forward path • Sensor dynamics (H(s)) in feedback path • Prefilter F(s)
FCA-2	• Single feedback loop • Plant (G(s)) in forward path • Compensator (C(s)) and sensor dynamics (H(s)) in feedback path • Prefilter F(s)
FCA-3	• Compensator (C(s)) and plant (G(s)) in forward path • Sensor dynamics (H(s)) in feedback path • Feedforward prefilter F(s) for input disturbance attenuation
FCA-4	• Outer loop with compensator (C1(s)) in forward path • Inner loop with compensator (C2(s)) in feedback path • Plant (G(s)) in forward path • Sensor dynamics (H(s)) in feedback path
FCA-5	• Compensator (C(s)) in forward path • Plant G1(s) and plant predictive model G2(s) • Prefilter F(s)
FCA-6	• Plant models (G1(s) and G2(s)), compensators (C1(s) and C2(s)) in the forward path • Sensor dynamics (H1(s) and H2(s)) in the feedback path of both loops • Prefilter F(s)

It is also hard to represent the reference tracking and disturbance rejection performances of a system with sketched plots. Additionally, it takes time to draw some of the plots and some may even require computers to be generated. Due to these problems, control lectures can be improved if a handwritten FCA (HFCA) can be transferred into a simulation environment like Matlab® that can generate accurate visual representations instead of rough sketches.

In the handwritten diagram detection literature, some of the studies deal with ink-input devices and use the locational information of pen strokes in their recognition method [15-19]. In ink-input devices, pen strokes are saved as collections of close points on a two-dimensional space. Mentioned studies handle the mentioned recognition task by assigning the strokes to distinct shapes. For example, a stroke can represent an arrow while another defines a rectangle. Since the positions of the points that define a shape are known, locations of the shapes can also be found, from which the complete chart

can be constructed. Although dealing with ink-input devices has its difficulties, the information provided by a digital drawing tablet is noise-free and every point belongs to the drawing with a high probability. For cases where the only input is a digital image, this initial information is not known. The only available information in a digital image is the finite number of evenly distributed two-dimensional points called pixels. A pixel contains one or more integer intensity values that can be interpreted as the brightness values for that particular pixel. For example, in a digital image where the intensity value is between 0 and 255, 0 represents complete darkness while 255 represents full illumination. To be able to detect the diagram in a digital image, a methodology that separates pixels belonging to the foreground from the background is required for further operations. In the context of handwritten diagram detection, pixels belonging to the diagram are considered as foreground while the rest are considered background. Extracting such knowledge falls into the area of digital image processing, which mainly deals with image manipulation problems.

Separating foreground pixels from the background can easily be accomplished for the images where there is a high contrast between both groups. Unfortunately, environmental lighting has disruptive effects in most of the cases. Light reflections on surfaces can saturate pixels that should have low-intensity values. Also, the quality and the resolution of the digital image can cause issues where elements in a scene are not represented accurately. These problems can be considered as uncertainties in an image processing task. Similar versions of the aforementioned uncertainties have been widely encountered in the handwritten flowchart and character recognition [20, 21]. The common point of these studies is that they use image processing algorithms to segment the desired parts from an image. Widely adopted approaches start by removing the noise in an image using filtering methods [16]. Then a binarization technique is applied to roughly separate foreground from the background [17]. Binarization techniques essentially apply a threshold to the pixels and generate a binary image where a pixel can either be 1 or 0. For further improving the segmentation, edge detection techniques that can detect sharp intensity changes in an image can be applied [18]. At the end of the segmentation steps, we generally want to end up with separate foreground pixel groups that may represent a character or any other shape in the image. In this context, connected component analysis can be used to find groups of pixels that form continuous regions [19].

The main purpose of separating foreground pixels using image processing techniques in a detection task is to eliminate all the other pixels that don't belong to the object of interest. These preliminary steps are called pre-processing steps and they provide the necessary information for further recognition operations. Being able to tell what a set of inputs represents can be considered as pattern recognition (PR) task. In recent years, imitating human behavior in machines gained more importance as automation became an essential part of today's society. As a result, a great amount of work has been made to develop computational methods that can recognize patterns in the last century [22]. For example, researchers successfully employed techniques that can recognize a written character [23] or a recorded speech [24] in the past.

The general form of a PR task is given in 1.1 where the d-dimensional vector x represents the input, scalar y represents the output, and θ is the set of parameters of function f.

$$\begin{aligned} y &= f(x;\ \theta) \\ x &\in \mathbb{R}^d \\ y &\in \mathbb{R} \end{aligned} \tag{1.1}$$

In PR tasks, inputs are considered as the features of the pattern to be recognized while the outputs are considered as the categorical data belonging to that pattern. A pattern recognizer maps the features in feature space to their corresponding labels in the output space. In handwritten diagram detection, the output can represent circles, rectangles, arrows, or other commonly used shapes. The main challenge is to construct the feature space. The area in terms of pixels, a value that represents the circularity or similar metrics for a pixel region found by connected component analysis can be considered as features. Examples of PR with such region metrics can be found in the literature [25, 26]. Using region properties as features may cause problems if input space shifts, rotates, or scales. There are feature extraction methods robust to these transformations such as histograms of oriented gradients (HOG) and scale invariant feature transform (SIFT) and studies that use these as feature extractors are present in literature [27,28].

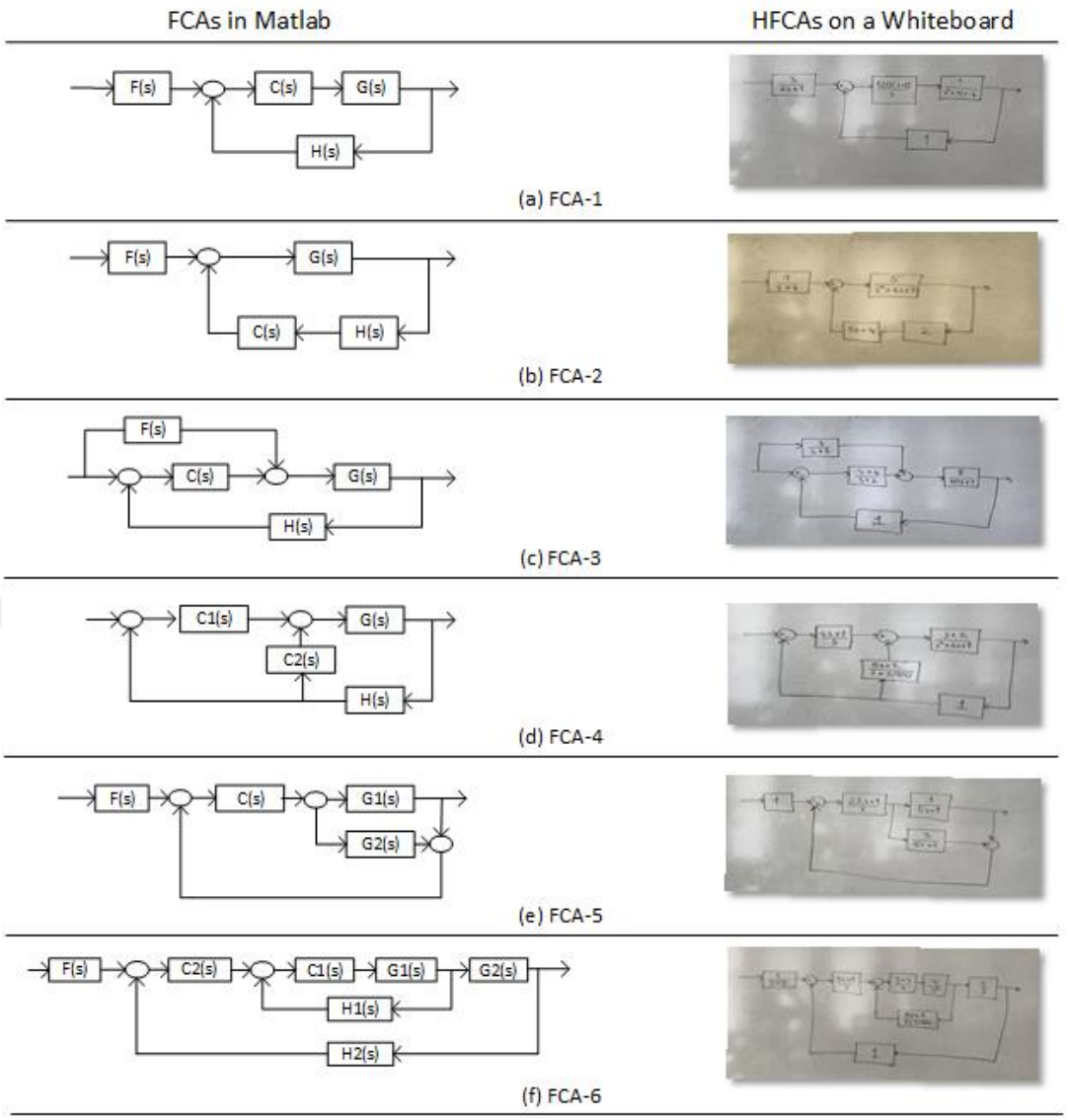


Figure 1.1 : Illustration of the FCAs

Choice of the feature extraction may depend on the task. A simple method that depends on regional properties in a binary image may perform poorly. On the other hand, a complex method such as SIFT can be too slow for real-time applications. Unlike these algorithms, neural networks can automatically extract features and they are recognized as an important tool in PR [29].

Generally compared to actual neurons in human brains, NNs consist of layers where each layer scales its input with a set of parameters called weights. They can represent any non-linear function by establishing a mapping between an input space and an output space. Just like a human brain adjusts the firing rates of certain neurons to represent new information, a NN also updates its weights to learn new mapping functions. The number of layers a network contains defines its capabilities.

Constructing deep NNs by stacking more layers can help to create more accurate models at the cost of computational resources. The field of Deep Learning (DL) interests in developing deep NNs and it tries to overcome problems that may arise because of the increasing layer depths. Regardless of their depths, NNs have been widely used in PR tasks [30-32]. One of the main advantages of NNs is that they can serve as an automatic feature extractor where the final outputs are the features of a given input. Further task-specific layers can then be added to the network to produce the desired results. The automatic feature extraction can be thought of as a mapping function that translates the inputs to the feature space. To correctly approximate this desired mapping function, the weights of the NN must be updated to give the best fit. Since the main purpose of using NNs in PR is to automatically extract features, an explicit form of this mapping function is unknown. Fortunately, if the correct outputs for a sufficient amount of inputs are known, weights can be updated by comparing the predictions with the ground truths and the network can learn from examples. The extracted features can then be used for classification with additional NN layers. In fact, there are early examples in the literature where NN is used solely in the classification part [33, 34]. In cases where the training samples lie on low dimensional spaces, they can be directly fed into a neural network without requiring complex feature extraction algorithms [35, 36].

The main problem in integrating NNs with PR for computer vision is that even small images lie on a high dimensional space when each pixel is considered as a dimension. Assigning weights to each pixel will not only result in very complex networks it will also transform the input into a one-dimensional vector. Flattening an image can harm the overall performance of the network as it will lose the positional information on one axis, depending on how the flattening is applied. Convolutional Neural Networks (CNNs) can overcome these problems and achieve satisfactory performances in various recognition tasks such as character recognition [37, 38], document recognition [39], face recognition [40], image recognition [41]. The major difference of CNNs from a regular NN is that they contain convolutional layers that use two-dimensional convolution operation to calculate their outputs. In a two-dimensional convolution, a set of weights arranged as a two-dimensional square matrix slide over the input image along both vertical and horizontal axis. By linearly combining the currently covered pixels with the weight matrix, the corresponding pixel of the output image can be

calculated. Since the same set of weights are used during this operation, CNNs need less learnable parameters compared to regular NNs. Convolutional layers also remove the necessity of flattening which saves the locational information of input pixels. It should also be noted that as the input image goes through several layers in a CNN, their dimensions change and they become less interpretable. Thus, the outputs in a CNN are considered as feature maps instead of actual images.

In recent years, various CNN architectures that achieve high classification accuracies on the ImageNet dataset [42] are developed by researchers. AlexNet can be considered the first major architecture since it showed the capabilities of CNNs in computer vision tasks [43]. Since AlexNet, architectures became deeper and more complex with different connection schemes. For example, the winner of ILSVRC 2015 ResNet contains up to 152 layers [44], while AlexNet contains 5 convolutional layers and 3 fully connected layers [43]. ResNet also introduces a new "skip connections" concept which helps to solve a common problem in DL called "vanishing gradients". Instead of designing a new architecture for a classification task, one can copy the architecture these networks that have already been proven successful. The main issue in this approach is that these deep networks are trained on large datasets and they require more time compared to simpler architectures. To avoid these problems, a pre-trained network can be used by replacing its non-feature extraction layers. This approach is called transfer learning and it enables researchers to make use of high performing architectures in any PR task without long training times or needing a large dataset like ImageNet [45]. In recent years, transfer learning approach have been widely used in medical image classification problems since it is challenging to construct large datasets in the medical field [46-48].

In this thesis, we proposed a DL based pipeline that is capable to recognize Handwritten FCAs (HFCAs) on the whiteboard and to transform them into Matlab® for visualization and analysis of FCAs. The main design challenge of the proposed DL based pipeline is to find a set of instructions to recognize the HFCAs. Because of the aforementioned uncertainties, it is challenging to employ classical image processing methods in this context. To deal with these difficulties, we integrated DL methods into our pipeline and trained two separate deep CNNs that can recognize HFCAs and handwritten characters on a whiteboard. We preferred the transfer learning approach to construct our deep CNNs and used a pre-trained ResNet-50 as our base model. For

the training of the task-specific layers, we constructed datasets containing images of HFCAs and handwritten characters on a whiteboard collected from five control system lecturers.

The proposed DL based pipeline starts by taking the image of an HFCA (shown in the right of Figure 1.1) as input and recognizing the corresponding FCA class (shown in the left of Figure 1.1) via the first deep CNN structure. Then, the recognized HFCA is further processed to extract the TF blocks by using classical image processing methods. In the next step, the characters of each extracted TFs are recognized and labeled by the second deep CNN structure. After generating symbolic expressions, continuous-time TF representations are generated that is compatible with Matlab®. The visualization and analysis of the HFCA are then straightforwardly performed via the Control System Design™ Toolbox and Simulink of Matlab® in real-time.

In the following chapter of this thesis, we give brief information about the history of DL and present important concepts that are already used in this work. We also explain the layers of a CNN with more details and further clarify the advantages of transfer learning along with the ResNet architecture. In the third chapter, we describe each step of the pipeline and represent the classification accuracies of our deep CNNs.

2. DEEP LEARNING

For long, humanity has imagined machines that can act on their own and only recently this dream is starting to be realized [49] with the rise of computers. Although the current technology hasn't reached the point where an artificial agent can replace a human, machine learning (ML) or more generally artificial intelligence (AI) became an essential part of today's society. Especially neural networks (NN), which is a sub-field of ML, are used frequently in engineering problems since they provide an unexplicit way to establish behavior in machines. Although NNs were known for a long time [50], they gained popularity in recent years with the advances in computational powers. Using the parallelism provided by powerful graphical processing units (GPUs), researches were able to construct NN architectures with high amount of layers. Today, these deep networks are investigated under the field of DL. In this chapter, we present a brief history of DL and NNs and give detailed information about the popular architecture ResNet-50.

2.1 Perceptron Models

NNs resemble networks in the human brain in the sense that the combination of simpler structures that has adjustable weights constructs more complex ones that could accomplish much harder tasks. In NNs, this simplest structure is called a perceptron. The task of this smallest building block is to map a set of inputs x to an output y by the means of linear combinations using a weight vector w and a bias b. A visual representation of such a perceptron is also given in Figure 2.1.

$$\begin{gathered} f(x, w, b) = w^T x + b = y \\ x, w \in \mathbb{R}^d \\ b, y \in \mathbb{R} \end{gathered} \tag{2.2}$$

The first computational model of a neuron proposed by McCulloh and Pitts [51] was also based on the same principle. They showed that their model can be used in binary

classification tasks, given in (2.2), by choosing a threshold value T_p. Of course, the accuracy of this method depends on w and T_p. In 1958, Rosenblatt proposed another perceptron model [52] where these parameters could be learned from a set of labeled examples. The proposed algorithm was to update w and T_p whenever the predicted output was different from its target value. Today, this kind of learning where a teacher exists is called supervised learning (SL) and it is an important concept in NNs.

$$y = \begin{cases} 1 \ \ if \ \ f(x,w,b) \ \geq T_p \\ 0 \ \ if \ \ f(x,w,b) \ < T_p \end{cases} \tag{2.2}$$

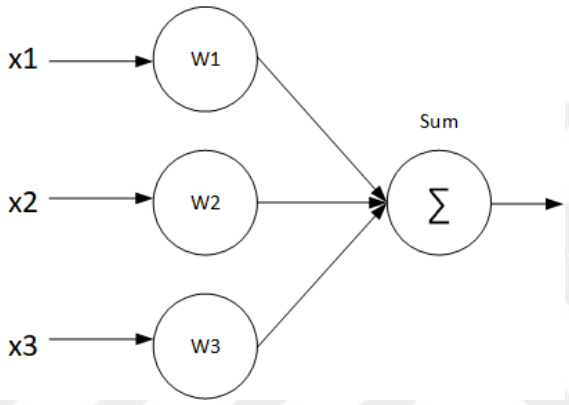


Figure 2.1 : A visual representation of perceptron

Although Rosenblatt's perceptron was an important step in NN's history, it was unable to separate non linearly separable classes or imitate the XOR functionality since only a single linear function was used to calculate the output. Also, thresholding was preventing to see how weights affected the produced outputs when their values changed. In 1960, Widrow and Hoff proposed a new perceptron model called ADALINE, which stands for adaptive linear neurons [53]. The main difference between ADALINE and its predecessor is that in ADALINE, errors are calculated before thresholding operation. Calculating errors from continuous values instead of quantized ones enabled a faster convergence. Widrow and Hoff also used their perceptron model to predict real valued numbers. In other words, their model was able to find the coefficients of a linear function given in (2.3). Approximating a function with given inputs and outputs is called a regression problem in ML and ADALINE was the first example of a linear regressor.

$$y = b + w_1 x_1 + \cdots w_d x_d \ , \ \ x \in \mathbb{R}^d \tag{2.3}$$

The perceptron model can also be improved by adding a non-linear activation function right after calculating $f(x, w, b)$. A perceptron with activation function is given in Figure 2.2. One of the most commonly used activation functions in NNs is the sigmoid activation function which is given in (2.4) where z is the activation of the perceptron.

$$g(z) = \frac{1}{1 + e^{-z}} \tag{2.4}$$

Since inputs to this activation function are the output of $f(x, w, b)$, the above equation can be written as in (2.5).

$$g\big(f(x, w, b)\big) = \frac{1}{1 + e^{-(w^T x + b)}} \tag{2.5}$$

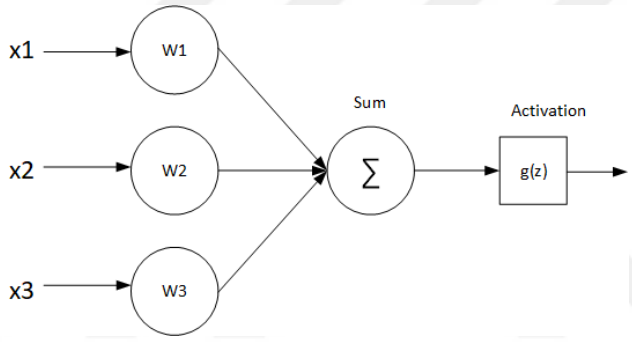


Figure 2.2 : Perceptron with activation function

The sigmoid function is plotted in Figure 2.3. From the figure, it can be seen that the output values vary in the (0,1) interval and they cross the 0.5 point when inputs are zero. Using this knowledge, a classification using (2.6) can be accomplished.

$$y = \begin{cases} 1 \;\; if \;\; g\big(f(x, w, b)\big) \geq 0.5 \\ 0 \;\; if \;\; g(\, f(x, w, b)) \;\; < 0.5 \end{cases} \tag{2.6}$$

With the sigmoid function, the search for a T value isn't needed. Also, the output values can be treated as the class probabilities. Using these probability values, the amount of error made by the perceptron can be calculated. In literature, classification with perceptrons that use sigmoid function is called logistic regression.

If a function that expresses the prediction error can be constructed, finding the best set of parameters would then become an optimization problem. Widrow and Hoff used a similar approach to update their perceptron's weights using a gradient-based numerical optimization method.

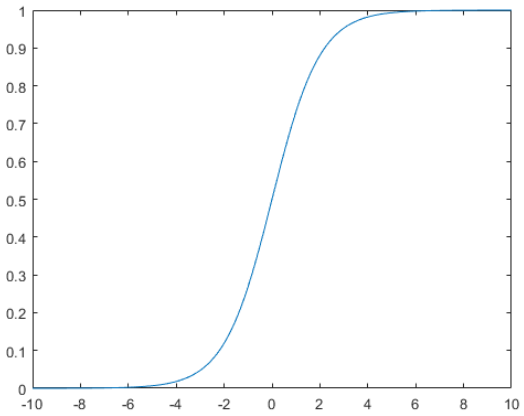


Figure 2.3 : Sigmoid activation function

2.2 Gradient-Based Learning

From calculus, it is known that the derivative of a function $y = f(x)$ gives information about how small changes in the input affect the output. Positive derivative means an increase and negative derivative means a decrease in the output. When a function is tried to be minimized, or in other words, a local minimum point is searched, derivative information can be very useful. By moving in the opposite direction of the derivative with small steps starting from a starting point, a minimum point for $f(x)$ can be found around a neighborhood. Since the movements are done in steps, this becomes an iterative method where the current point is updated at each iteration. Update rule is given in (2.6) where $x[k]$ is the current point, $x[k+1]$ is the next point and α is a constant that defines the step size called the learning rate.

$$x[k+1] = x[k] - \alpha \frac{df}{dx} \tag{2.7}$$

When the function that desired to be minimized has multiple inputs, update rule becomes (2.7), where ∇f is the gradient vector of $f(x)$.

$$x[k+1] = x[k] - \alpha \nabla f, \quad x \in \mathbb{R}^d \tag{2.8}$$

The iterative process of searching a local minimum point can be terminated when ∇f becomes a zero vector by the definition of an extremum point, but this approach may fail if $x[k+1]$ never rests at the solution, since α may never be small enough. As a stopping criteria, the process can be terminated when the value of $f(x)$ increases

instead of decreasing. This iterative approach of minimization is called Gradient Descent (GD) and is one of the key ideas of NN's. A visual representation of Gradient Descent is given in Figure 2.4. It is also common practice to run the algorithm for a pre-defined number of epochs (iterations) instead of establishing a stopping condition.

To train a perceptron using GD, a suitable $f(x)$ must be found. This $f(x)$ is called the loss function in ML and it gives a measurement of the prediction error made by the model. If the learnable parameters of the perceptron are given as arguments to the loss function, GD can find a minimum point where the error is minimized, using the derivatives of these parameters.

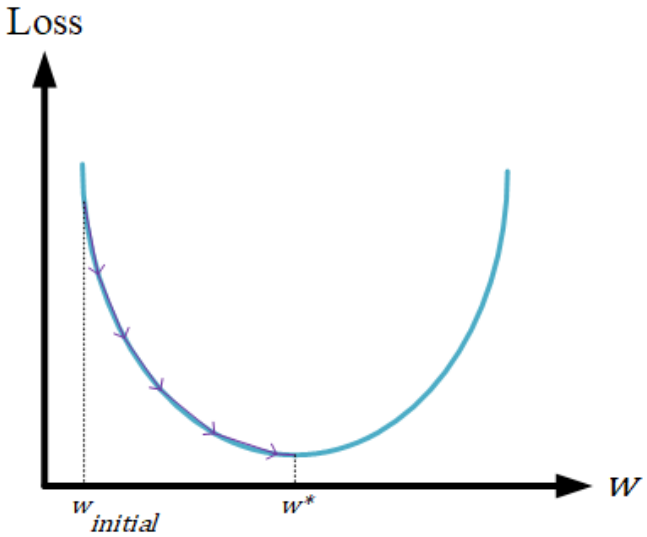


Figure 2.4 : Visual representation of GD

A downside of the GD method is that the calculated minimum point may not be a global one. Only if a convex function is chosen as a loss function a global solution for the minimization problem can be found, since convex functions have a single extremum point.

For a labeled training set of N samples, function to be minimized $J(w, b)$ is expressed in (2.8) where x^i is the ith sample, $\hat{y}^i$ is the label of the ith sample and y^i is the predicted label for the ith sample.

$$J(w,b) = \frac{1}{N}\sum_{i}^{N} Loss\left(y^i, \hat{y}^i, w, b\right) \tag{2.8}$$

As it can be seen, $J(w,b)$ only depends on the learnable parameters of the perceptron and it takes the average of the loss values calculated for all the samples in the training set. With $J(w,b)$ defined, update rule can be written as the following.

$$w[k+1] = w[k] - \alpha \frac{\partial J(w,b)}{\partial w}$$
$$b[k+1] = b[k] - \alpha \frac{\partial J(w,b)}{\partial b} \tag{2.9}$$

The partial differentials with respect to layer parameters are given in (2.10).

$$\frac{\partial J(w,b)}{\partial w} = \frac{1}{N}\sum_{i}^{N} \frac{\partial Loss\left(y^i, \hat{y}^i, w, b\right)}{\partial w}$$
$$\frac{\partial J(w,b)}{\partial b} = \frac{1}{N}\sum_{i}^{N} \frac{\partial Loss\left(y^i, \hat{y}^i, w, b\right)}{\partial b} \tag{2.10}$$

The selection of the proper loss function depends on the task at hand as different tasks require different loss functions. One of the simplest loss function that could be used to train a perceptron is the Mean Square Error (MSE) function. Using the MSE loss, $J(w,b)$ can be expressed as follows.

$$J(w,b) = \frac{1}{N}\sum_{i}^{N}\left(y^i - \hat{y}^i\right)^2 \tag{2.11}$$

The MSE loss function is generally used for regression tasks. The problem arises for the logistic regression. Using MSE loss along with sigmoid function results in a non-convex $J(w,b)$. For regression problems where the activation function can be linear MSE will prove useful, but for classification tasks, another loss function must be selected. In literature, a cross-entropy loss, presented in (2.12) is generally used for classification tasks [54].

$$Loss_{Cross} = -(\hat{y}^i \log\left(y^i\right) + (1-\hat{y}^i)\log(1-y^i)) \tag{2.12}$$

2.3 Stochastic Gradient Descent

Most of the NN models utilize a modified version of the GD algorithm called Stochastic Gradient Descent (SGD) [54]. In GD, parameters are updated after calculating loss values for all the samples in the training set. Ideally, this approach is preferred for better generalization since the gradients calculated with every available sample better represents the statistics of the input space. The downside is, it may not be possible to calculate the gradient for large datasets because of the computational costs. To handle datasets that are too large for GD, a smaller part of the dataset called mini-batch can be sampled from the complete set. Gradient calculated from this mini-batch may be different from the actual gradient since a small set can not completely represent the whole set. For fast convergence, the mini-batch gradient should be similar to the actual gradient as much as possible. This can only be achieved by uniformly selecting examples from the dataset. For a mini-batch with m elements, the mini-batch gradient is given in (2.13) where θ is the learnable parameters.

$$g = \frac{1}{m}\nabla_\theta \sum_{i}^{m} \frac{\partial Loss\left(y^i, \hat{y}^i, \theta\right)}{\partial \theta} \tag{2.13}$$

Using the mini-batch gradient, parameters can be updated as in (2.14).

$$\theta[k+1] = \theta[k] - \alpha g \tag{2.14}$$

As it can be seen, gradient calculation and update rule in SGD are nearly the same as GD. If the randomly selected mini-batch is a good representative of the complete dataset, there won't be any significant performance hit. If the mini-batch size is selected as too small, the actual gradient can't be correctly estimated. On the other hand, large mini-batch sizes require large amounts of memories and can cause long training times. Since the choice of mini-batch size is a trade-off between computational costs and good convergence, it is considered as a hyperparameter of SGD.

2.4 Multilayer Perceptrons

Using single perceptrons for classification or regression tasks may give poor results as the complexity of the tackled problems increase. To better approximate the desired mapping function, perceptrons can be used in a collection. Multilayer perceptrons are

structures where perceptrons are stacked in a layered fashion. The terms multilayer perceptron and NN are used interchangeably in ML. A similar relation also exists between the term perceptron and the term neuron.

In NNs, a layer can include any number of neurons while the whole structure usually contains at least three layers. An example of NN is given in Figure 2.5. The first layer of a NN is called the input layer while the last layer is called the output layer. All the remaining layers are named as hidden layers. NNs process the fed data by propagating it through its layers. For example, the ith layer uses the outputs of each neuron in (i-1)th layer to provide input for the (i+1)th layer. Calculations in a layer are similar to the operations presented for the perceptron model, only difference is that the weights are stored in matrices in NNs instead of vectors. Size of the weight matrix for a particular layer depends on the neuron count of both that layer and the layer before it. If ith layer has m neurons and (i-1)th layer has n neurons, the weight matrix of ith layer, which can be defined as w^i, will have n rows and m columns and its bias vector b^i will be a vector of m elements. If the same activation function g_i is used throughout all the units in layer i, its output can be calculated as in (2.15). This process of output generation in NNs is called forward propagation (FP).

$$\begin{aligned} y_i &= g_i(w_i.x + b_i) = g_i(z_i) \\ x_i &\in \mathbb{R}^{n\times 1} \\ w_i &\in \mathbb{R}^{m\times n} \\ b_i &\in \mathbb{R}^{m\times 1} \end{aligned} \tag{2.15}$$

As it can be seen in (2.15), there is a matrix multiplication between the layer's weight matrix w_i and input vector x. A row in the weight matrix holds the individual weights going through a specific neuron in the layer. For example, rth row in w_i represents the weights of the rth neuron in ith layer. This implies that the term $w_i.x$ creates all the possible connections between (i-1)th and Ith layer. Layers constructed this way also called fully connected layers in the literature. A network consisting of solely fully connected layers can also be called a fully connected network. Figure 2.5 is also an example of a fully connected network (FCN). Researches show that when an adequate number of neurons used, any linear function can be represented with a NN [55].

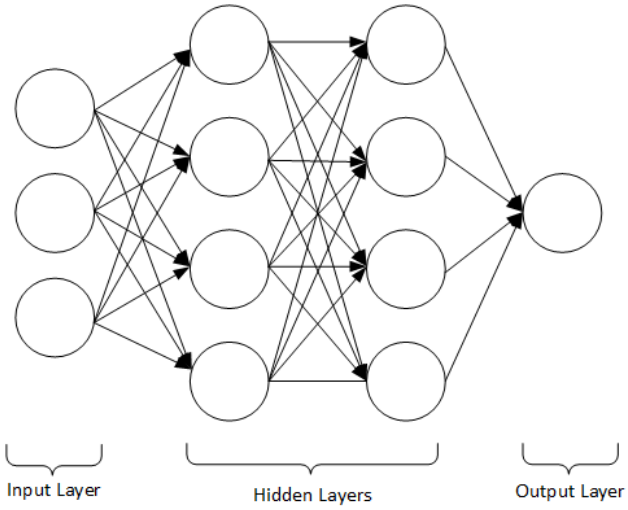


Figure 2.5 : An example NN

A NN can generate multi-dimensional outputs since the final layer can have more than one neuron. This way, regression problems in multi-dimensional spaces or multi-class classification problems can be handled. For regression tasks, neurons at the final layer represent dimensions of the predicted output. On the other hand, a classification task consisting of Q classes aims to assign probability values for a given input for each of the classes. The probability values assigned to the classes are a measure of how likely they belong to those classes. Thus, the outputs of classification networks should be a vector belonging to $\mathbb{R}^Q$. The predicted label will then be the index of the maximum element of the output vector.

In logistic regression, the sigmoid function was able to classify inputs since it was a binary classification problem. Multi-class tasks however require another activation layer at the final layer. In literature, the softmax activation function defined in (2.16) is used at the final layer for multi-class classification problems [56]. Softmax can be thought of as the extension of sigmoid to the multi-class domain. It takes the values in the network's final layer and produces the class probabilities. Similar to the sigmoid function, softmax results in a non-convex loss function if MSE is used. Thus, the cross-entropy loss function is preferred for multi-class classification tasks with NNs.

$$softmax(y)_i = \frac{e^{y_i}}{\sum_{k=1}^{Q} e^{y_k}} \; for \; i = 1 \dots Q \tag{2.16}$$

2.5 Back-propagation

With perceptrons, overall cost function was presented as $J(w, b)$. In NN's, this can be re-written as $J(\theta)$, where θ is the collection vector of all weights and biases in the network. While minimizing $J(\theta)$, an analytical expression for its gradient can be found, but it would be computationally expensive to evaluate. This issue can be solved by an algorithm called back-propagation [54].

Back-propagation (BP) is a method that is used to compute the gradients of all the parameters used in a function. The algorithm treats the operations in a NNs as a computational graph and utilizes the chain rule of calculus to find the gradients. All the variables that are required in the gradient calculation are represented as nodes and operations are shown in computational graphs. Graph representation of the multiplication operation is given in Figure 2.6.

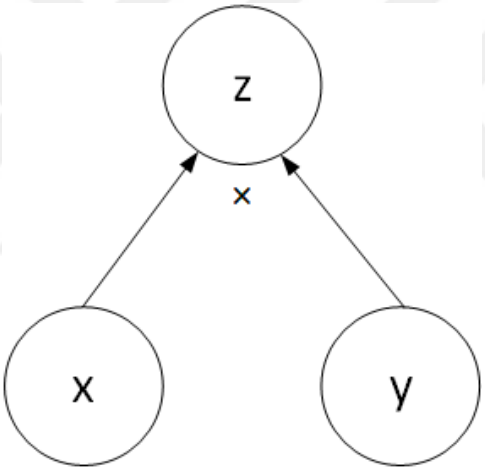


Figure 2.6 : Graph representation of multiplication operation

The calculation of a node's gradient starts by traversing the graph backward from the output node. Along the way, gradients of all the traversed nodes must be calculated as well. To avoid repeating the same computations, the algorithm first handles nodes that are closer to the output and saves the calculated gradients. For demonstration, the computational graph of a three layered NN is given in Figure 2.7. Parameters that need update are w_2, b_2, w_3 and b_2 where the second layer is the hidden layer and the third layer is the output layer. Gradients of the third layer are given in (2.16) and in (2.17) while the gradients of the second layer in (2.18) and in (2.19).

$$\frac{\partial J}{\partial w_3} = \frac{\partial J}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial w_3} \tag{2.16}$$

$$\frac{\partial J}{\partial b_3} = \frac{\partial J}{\partial y_3}\frac{\partial y_3}{\partial z_3}\frac{\partial z_3}{\partial b_3} \tag{2.17}$$

$$\frac{\partial J}{\partial w_2} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial w_2} \tag{2.18}$$

$$\frac{\partial J}{\partial b_2} = \frac{\partial J}{\partial z_3}\frac{\partial z_3}{\partial y_2}\frac{\partial y_2}{\partial z_2}\frac{\partial z_2}{\partial b_2} \tag{2.19}$$

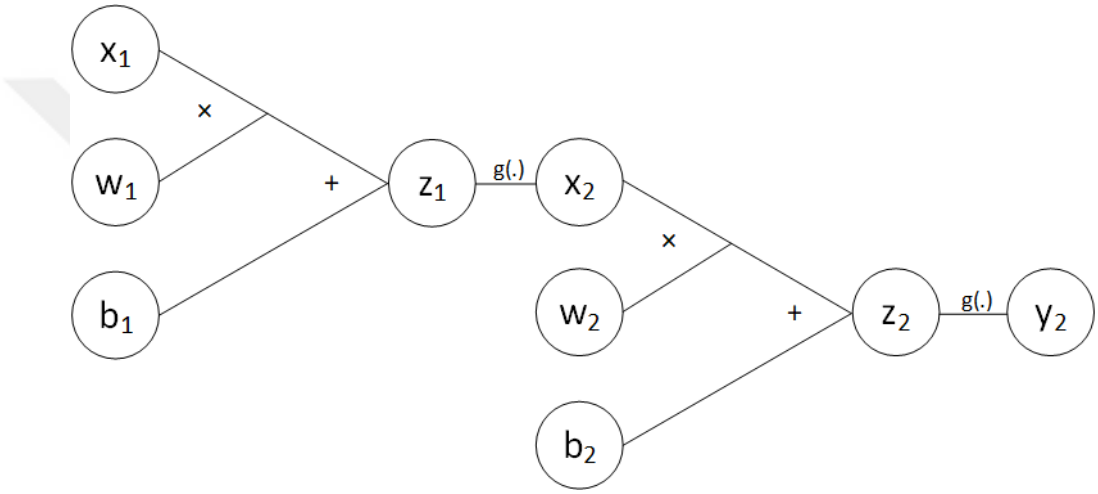


Figure 2.7 : Computational graph of a three-layered network

2.6 Training, Validating and Testing Datasets

Training NNs uses labeled samples to learn the parameters since loss functions require ground truth values. In ML, the collection of samples that are used to update the learnable parameters is called a training dataset. After updating the parameters of the network, some error measurement can be computed by applying a FP to the training data to calculate the training error. The training error can also be calculated from the mini-batches if SGD is used. If the optimizer is able to find a minimum point, the training error should be small for the trained network. For a regression task the only available error metric is the loss values. For a classification task, number of correctly classified samples is more important than loss values. In ML the classification accuracy is measured with a top-k error rate. Prediction for a sample is considered as correct if the class probability assigned to it is in the top-k. For example, a prediction

can only be considered as true if the highest probability assigned by the model is the same as ground truth in a top-1 error rate measurment.

Performance of a model on a training dataset is important but in ML problems the aim is to perform well on unseed data. A model that shows low error rates on unseen data is said to be well-generalized. The generalization capacity of a model is tested on a test set that consists of samples that haven't been used during the training. The test samples should be in the same domain as the training samples. If a model is trained to classify dogs and cats, test set should contain images of dogs and cats that aren't present in the training set. Sometimes a model performs well on training dataset but gives large errors for the test set. This phenomenon is called the overfitting and can be caused by small sized training dataset [57].

In ML, an additional dataset called the validation set can be used during the training. A ML model can contain various hyper-parameters such as the learning rate, number of training epochs, mini-batch size and many more. There isn't an established methodology for the selection of these hyper-parameters and it is generally handled with try and error. To be able to tell how the currently selected hyper-parameters change the performance, the model can be tested with the validation set at certain iterations. The feedback from the validation set can then be used to tune the hyper-parameters. Although samples from validation set aren't used to update the learnable parameters, they are considered as a part of the training set. The final performance of the model can only be measured on the dataset.

Generally, the training samples hold a great portion of the complete dataset. An example partition for a dataset is given in Figure 2.8 where training samples consist the 70% of all the data.

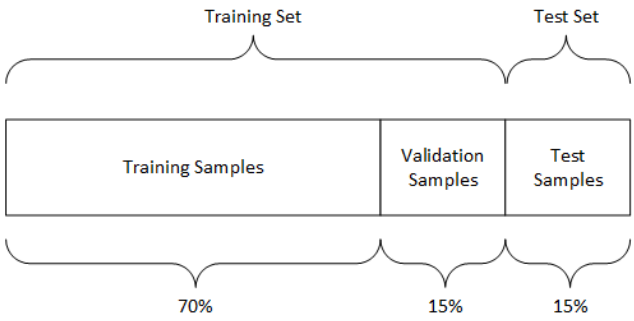


Figure 2.8 : Example partitioning of a dataset.

2.7 Methods to Improve Performance

2.7.1 Regularization

A common problem in ML is that a model can perform poorly on test data while achieving high accuracies or small loss values during the training [54]. A model that behaves in such a way is said to be overfitted. This problem can be solved with regularization methods which adds a parameter norm penalty function $\Omega(w)$ to the loss function. A loss function with regularization is given in (2.20).

$$\tilde{J}(\theta) = J(\theta) + \lambda\Omega(w) \tag{2.20}$$

In the above equation, λ is a non-negative hyperparameter that adjusts the regularization effect where higher values mean more regularization. It should also be noted that regularization affects only w. It has been observed that regularizing biases cause underfitting in NNs [54].

Although $\Omega(w)$ can be defined in many ways, these definitions must affect the weight values to avoid overfitting. A simple type of $\Omega(w)$ is the L^2 parameter norm penalty which is defined in (2.21).

$$\Omega(w) = \frac{1}{2}\|w\|_2^2 \tag{2.21}$$

In L^2 regularization, weights tend to decay since they appear on the loss function and the optimizer would try to minimize them.

2.7.2 Gradient descent with momentum

A downside of the GD is that sometimes gradients may oscillate as it approaches to an optimum point. An example case is given in Figure 2.9. From the figure, it can be seen that the updates should be larger on the horizontal axis for faster convergence.

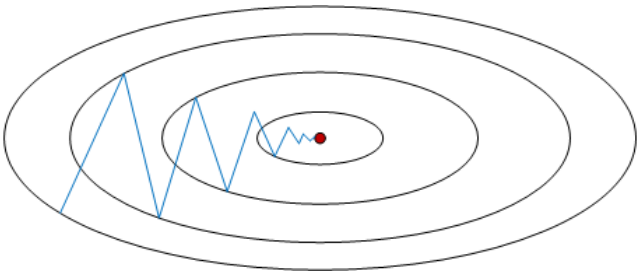


Figure 2.9 : An example GD with oscillating gradients

The average of the past gradients can be useful during the optimization. Since the moving average of the past gradients will show the direction towards the optimal point, the update rule can be re-written to include this average. In (2.22) update rule for the GD with momentum is given where m is a scalar in the interval [0,1].

$$\begin{aligned} v[k+1] &= mv[k] - \alpha \nabla_\theta J(\theta) \\ \theta[k+1] &= \theta[k] + v[k+1] \end{aligned} \tag{2.22}$$

GD with momentum speeds up the training process by accumulating the past gradients as v. If these past gradients oscillate along an axis, their displacement along that axis will cancel each other out since the current gradient is subtracted in (2.22). The scalar β is usually called the momentum and it adjusts how much the past gradients will affect the current direction.

2.7.3 Batch normalization

A problem that is encountered while training a NN is that distribution of the input data can vary across the samples which makes it harder to learn an appropriate mapping. Such a shift in the input distribution is called the covariate shift [58] and can be solved by feeding the normalized. However, for a deep NN covariate shift can happen even in the hidden layers [59] as the parameters are updated during the training. Batch normalization (BN) aims to eliminate the internal covariate shift by fixing the means and variances of the layer's inputs [59]. Regular normalization can be realized by subtracting the batch mean from each sample and dividing it with the batch standard deviation. BN modifies this as shown in (2.23).

$$\begin{aligned} \hat{x}_i &= \frac{x_i - \mu_b}{\sqrt{\sigma_b^2 + \epsilon}} \\ y_i &= \gamma \hat{x}_i + \beta \end{aligned} \tag{2.23}$$

In the above equation, μ_b and σ_b^2 are the mean and variance of the batch while ϵ is a very small constant to avoid division by zero. BN introduces two new learnable parameters in the form of γ and β. These parameters scale and shift the normalized layer inputs $\hat{x}_i$ to produce the layer output y_i. It is also observed that the parameters also act as regularization term and they improve the generalization of the model [59].

2.8 Convolutional Neural Networks

CNNs are special NNs that are commonly used in computer vision tasks like object classification and detection. After their first proposal in 1995 by LeCun [60] they gained popularity with AlexNet [44]. Unlike fully connected NN's, CNN structures use convolution operation in forward-propagation instead of matrix multiplication [54]. This way, the data can be processed in a grid topology, meaning that CNNs can deal with three-dimensional matrices called tensors. The third dimension in these matrices is called channels. Images can be given as an example to tensors since colored images contain three channels for red, green, and blue (RGB) colors.

The reason behind using CNNs in computer vision tasks is that they are much more efficient in working with images compared to a regular NN. Very first drawback of FCNs is that they can only accept two-dimensional inputs. In order to be able to feed images to an FCN, they must be vectorized. Vectorization can be done by juxtaposing the consecutive rows or the consecutive columns to each other for each channel. Then the individual channel can be stacked to create a one-dimensional vector. Transforming a three-dimensional data to a vector will cause an information loss on at least two dimensions. CNNs on the other hand, do not require vectorization and saves the locational information of pixels on all the axis.

The number of learnable parameters affects directly the network's complexity and the time required to train it. A very complex network may not even be trainable if it requires large amounts of computational resources. That is why it is not efficient to use FCNs in computer vision tasks. Even images with small resolutions contain many pixels and will result in large weight matrices in fully connected layers. For example, a single channeled image with a resolution of 32x32 contains 1024 pixels. If the neuron count of the next layer is n, the weight matrix of the next will contain $1024 \times n$ learnable parameters. CNNs do not suffer from this problem as they use convolution operation instead of matrix multiplication.

A typical CNN consists of dimension reduction and activation layers alongside its convolutional layers. Outputs of these layers that accept n-channeled tensors as inputs can be interpreted as feature maps. In classical image processing algorithms, various methods are used just for feature extraction tasks [61]. Since CNN's are essentially neural networks that can efficiently process images, they can automatically extract

features without needing any other algorithm. In fact, past researches show that the convolutional layers in a CNN act as a feature extractor and can be used in various tasks [62]. An example CNN architecture called LeNet is given in Figure 2.9.

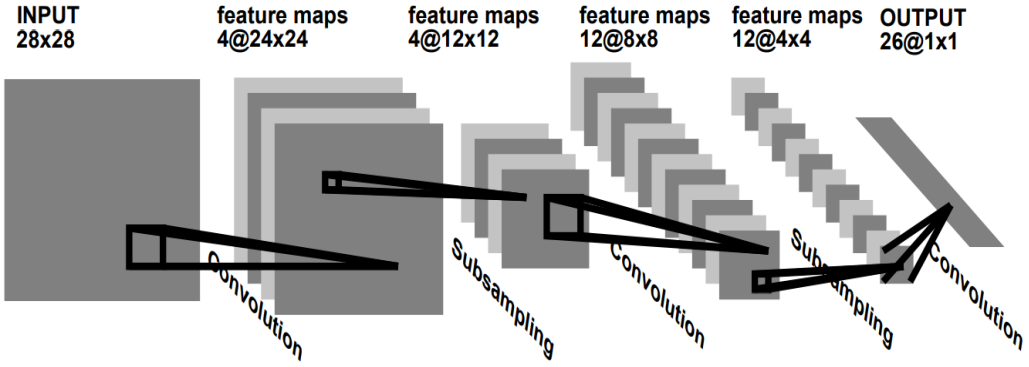


Figure 2.10 : LeNet Architecture [60]

In Figure 2.10, the final layer is visualized slightly differently from the rest. This last layer is a fully-connected layer and it serves the purpose of assigning class probabilities. While the remaining convolutional layers serve the purpose of feature extraction, other task-specific layers are required to use the extracted features for a desired task. That is why CNNs usually contains a task-specific layer at their ends.

2.8.1 Convolutional Layer

Convolutional layers can be considered as the backbones of CNN structures. In convolutional layers, outputs are calculated by convolving a special tensor called kernel over the input tensors. Kernel tensor holds the learnable parameters of the layer and it can be compared to the weight matrix of the fully connected layers. Convolution operation in one dimension for continuous signals is given in (2.24).

$$z(t) = (x * k)(t) = \int x(a)k(t-a)da \tag{2.24}$$

The same operation can be expressed as in (2.25) for two-dimensional signals where H_f is the height (row count) and W_f is the width (column count) of the filter.

$$z[i,j] = (x * f)[i,j] = \sum_{m}^{H_f} \sum_{n}^{W_f} x[i-m, j-n] f[m,n] \tag{2.25}$$

Convolution operation on tensors is conducted with a sliding window approach. Equation (2.22) is repeated for each element of input matrix x where i and j represent the row and column indices of the currently iterated element. At each iteration, i is incremented by 1 and when j^{th} is completely traversed the algorithm returns to the first column and increment j by one.

A problem arises during the convolution operation at the edges of the convolved image where $x[i-m, j-n]$ is not defined. This undefined element case occurs when either the index values are negative or greater than the width or height of the x. To avoid this, the following constraints can be applied where H_x is the height of input image and W_x is the width of the input image at the cost of having a downsampled output image.

$$
\begin{aligned}
i &\geq \begin{cases} \frac{H_f}{2} & if\ H_f\ is\ even \\ \frac{H-1}{2} & if\ H_f\ is\ odd \end{cases} \\
i &\leq \begin{cases} H_x - \frac{H_f}{2} & if\ H_f\ is\ even \\ H_x - \frac{H_f-1}{2} & if\ H_f\ is\ odd \end{cases} \\
j &\geq \begin{cases} \frac{W_f}{2} & if\ W_f\ is\ even \\ \frac{W_f-1}{2} & if\ is\ odd \end{cases} \\
j &\leq \begin{cases} W_x - \frac{W_f}{2} & if\ W_f\ is\ even \\ W_x - \frac{W_f-1}{2} & if\ W_f\ is\ odd \end{cases}
\end{aligned}
\tag{2.26}
$$

A convolution operation with the above constraints is shown in Figure 2.11. Alternatively, classical image processing methods solve this problem by applying a padding to the input image [63]. Padding can be in the form of 0 valued elements added around the input image (zero padding) or by repeating the values at the edges. Same principle can also be applied to the convolutional layers. A convolution operation with zero padding is shown in Figure 2.12.

Convolution operation on two dimensions can be used with matrices, but for CNNs where even a grayscale image is expressed as a tensor with a single channel, the above

expression must be updated as follows, where C is the number of channels for both of the tensors.

$$z[i,j] = (x * f)[i,j,k] = \sum_{k}^{C} \sum_{m}^{H_f} \sum_{n}^{W_f} x[i-m, j-n, k] f[m,n,k] \tag{2.27}$$

Although the input tensor and kernel may have multiple channels the result will be a two-dimensional matrix in convolutional layers. It should also be noted that channel sizes of input tensor and the filter must be the same. Since convolution on tensors creates single channeled tensors, or simply two-dimensional matrices, a convolutional layer can have multiple filters. In fact, the number of filter F is a hyperparameter that should be decided by the designer. Other hyperparameters for a convolutional layer are H_f and W_f, but not C. Number of channels of a filter and the filter count are not the same thing. A convolutional layer can have F amount of C channeled filters, with a total of $H_f \times W_f \times C \times F$ learnable parameters. The value of C on i[th] layer depends on the value of F on the (i-1)[th] layer.

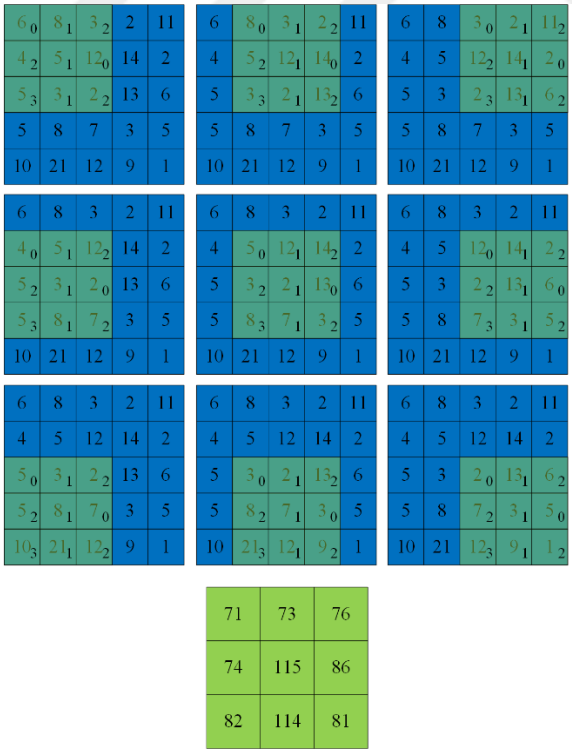


Figure 2.11 : Convolution operation with constraints

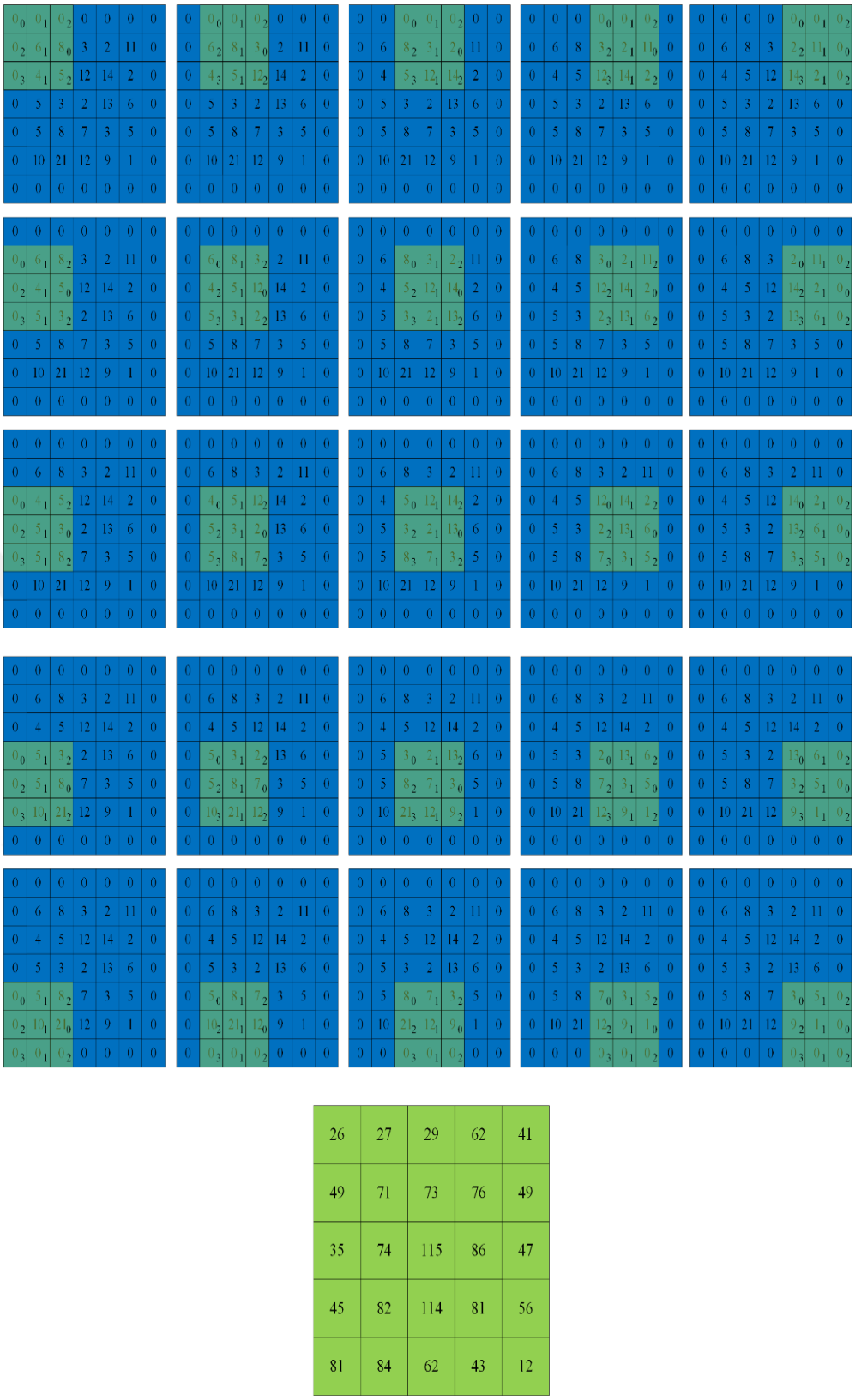


Figure 2.12 : Convolution with zero padding

Another commonly used technique in the convolutional layer is to skip a few elements in the tensor while conducting the convolution operation. Instead of incrementing i and j by one as explained earlier, they can be incremented by a stride value S which is also a hyperparameter. This technique is called strided convolution and can help to diminish the dimensions of the output tensor.

All the hyperparameters will affect the dimensions of the output tensor. Its channel number will the same as F. Output height H_z and width W_z values are given in (2.28) and (2.29) respectively.

$$H_z = \frac{(H_x - H_f + 2P)}{S} + 1 \tag{2.28}$$

$$W_z = \frac{(W_x - W_f + 2P)}{S} + 1 \tag{2.29}$$

2.8.2 Pooling Layer

Although H_z and W_z values can be adjusted in convolutional layers, pooling layers provides a way for dimension reduction without introducing new learnable parameters. Pooling layers traverses the input tensor just like the convolutional layers and selects a region of size H_f by W_f around $x[i,j]$. The corresponding element of the output tensor $z[i,j]$ then calculated by applying a desired operation to this region. Striding can also be implemented in pooling layers. Output tensor's H_z and W_z values in a pooling layer are given in (2.30) and (2.31) respectively.

$$H_z = \frac{(H_x - H_f)}{S} + 1 \tag{2.30}$$

$$W_z = \frac{(W_x - W_f)}{S} + 1 \tag{2.31}$$

Most common operations on pooling layers are averaging [64] and max operation [65], but any other operation can also be selected as long as they return a single element for a pooled region. A visual representation of max pooling with $S = 2$ is given in Figure 2.13 and a visual representation of average pooing with $S = 2$ is given in Figure 2.14.

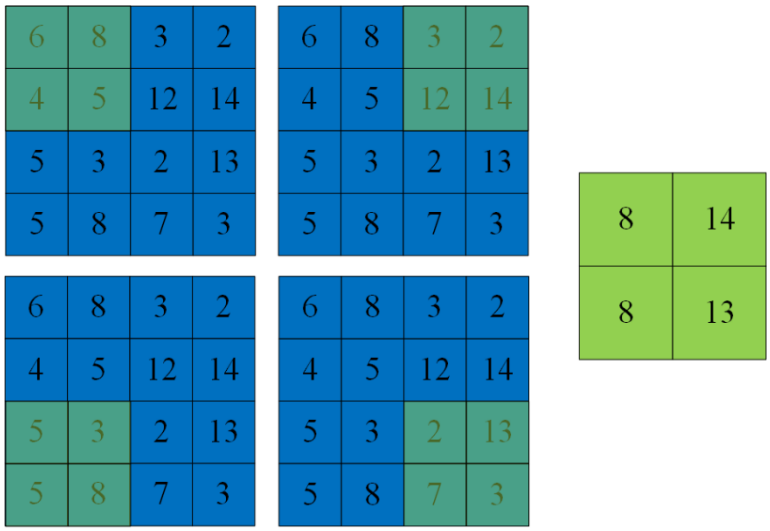


Figure 2.13 : A visual representation of max pooling

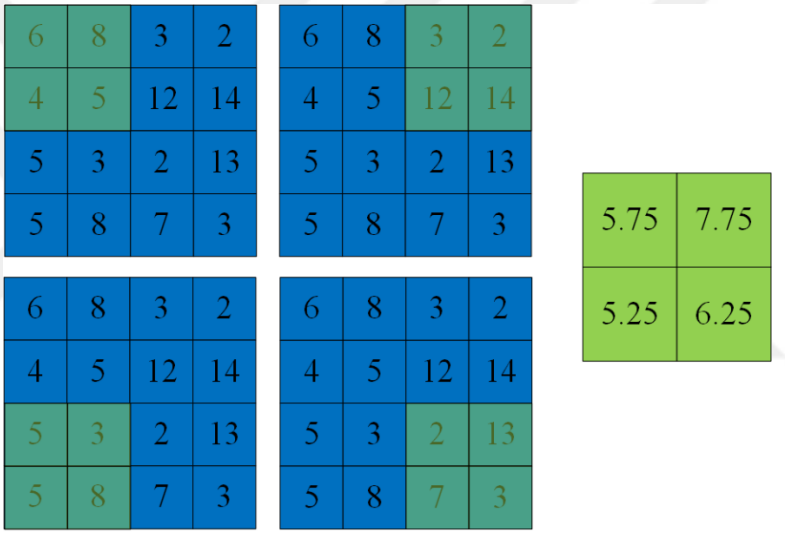


Figure 2.14 : A visual representation of average pooling

Along with reducing dimensions, pooling layers can make the network more robust to small translations in the input image [54]. It does so by summarizing the pixels in a neighborhood and extracting the more important responses.

2.8.3 Two Dimensional Batch Normalization

It is common practice to apply BN in CNN for the benefits explained earlier. BN is generally applied after the convolutional and before the activation layers. BN in CNNs differs from BN in fully connected networks in the sense that the batch statistics are calculated for each channel of the input to be normalized. This requires separate of γ and β for each channel. Other than that, the calculations remain the same.

2.8.4 Activation Layer

The convolution operation is essentially a function that takes the linear combination of its inputs and because of that, a CNN may not be able to perform well without non-linear activations. Activation layers in CNNs generate their outputs by applying a non-linear function to their inputs. The sigmoid function used in earlier perceptron models can also be used in the activation layers of CNNs. Also, a slightly different version of the sigmoid function called the hyperbolic function is another option. Unlike the sigmoid function, the output range of the hyperbolic function is between -1 and 1. In other words, it can also output negative values. The definition of hyperbolic function is given in (2.32) and its plot is given in Figure 2.15.

$$g(z) = \frac{2}{1 + e^{-2z}} - 1 \tag{2.32}$$

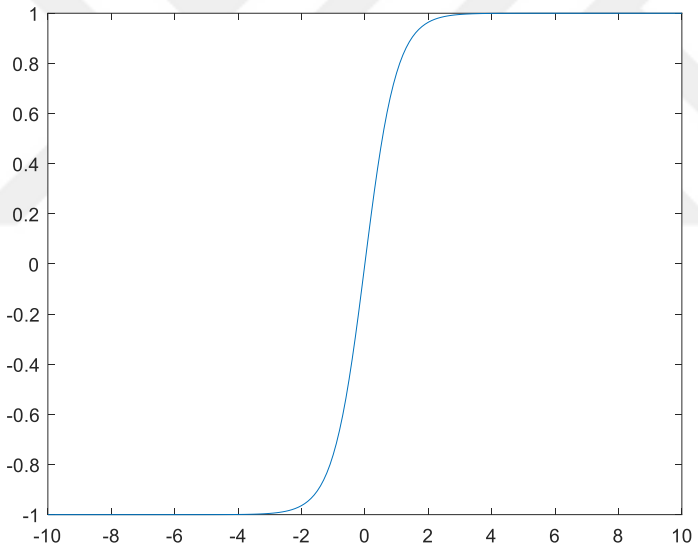


Figure 2.15 : Hyperbolic activation function

Both sigmoid and hyperbolic activation functions suffer from a similar problem. They are good assigning probability values to the inputs but the gradients are non-zero for a small portion of the input space. Values that are far away from the origin will always produce zero gradients and the network's weights won't be updated. This can be seen from the first-order derivative plots of the sigmoid and hyperbolic function given in Figure 2.16.

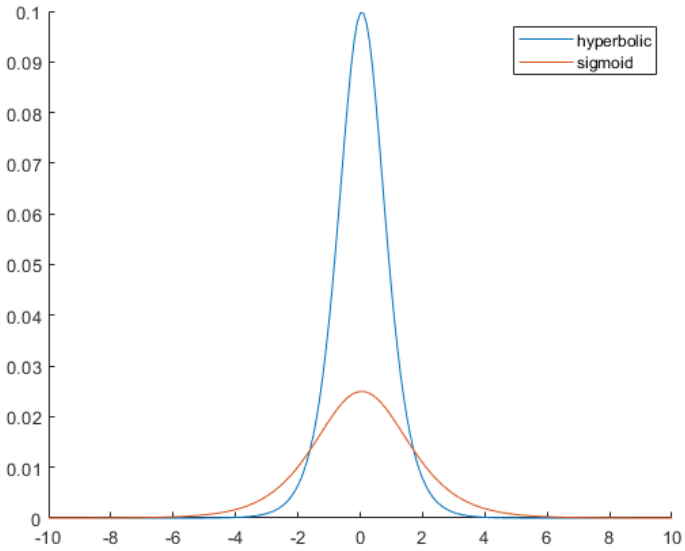


Figure 2.16 : First-order derivatives of hyperbolic and sigmoid functions

Getting zero gradients for changing inputs is called the vanishing gradient problem and it can hinder the learning process of the network. This problem can be solved by normalizing the inputs coming into the activation layer or using a normalized initialization for the layer weights [66], [67]. With a mean and variance normalization, data can be distributed along with the origin and zero gradients can be avoided. Another possible solution of course is to use another activation function. Unlike the mentioned activation functions, rectified linear unit (ReLU) [68] doesn't cause vanishing gradients. This activation function only sets the negative values zero. The sharp behavioral change at point zero that could be seen in Figure 2.17 adds the non-linear characteristic to the network. Another advantage of ReLU is that its gradients are easier to calculate compared to sigmoid and hyperbolic activation functions.

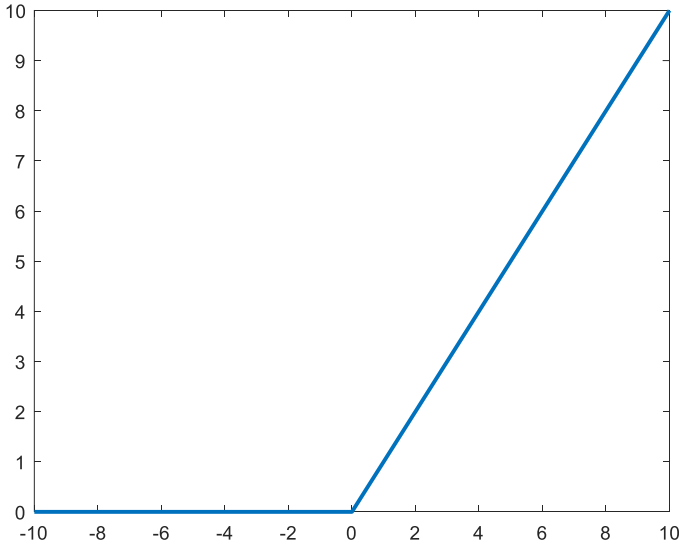


Figure 2.17 : Plot of ReLU activation function

2.8.5 Task-Specific Layer

The convolutional, pooling, and activation layers form the feature extraction part of a CNN. A well-established feature extraction method has good generalizability and can be used in various tasks. The purpose of the task-specific layers is to output the desired results using the extracted features. In computer vision, the handled task can be categorized as object detection and object classification.

Object detection is the task of finding the location of an object in an image. It is an often addressed problem in computer vision and there exist different approaches in object detection [69]. In the object detection literature, the location of an object is generally defined by the smallest rectangle that contains the object called the bounding box (BB) [70]. The BB is defined with four parameters; the horizontal position of its top-left corner, the vertical position of its top-left corner, its width, and its height. One of the possible approaches is to handle the task as a regression problem. With a FC layer at the end, the network can be trained with MSE loss to predict the four values that define the BB [71].

In case the CNN is desired to be used for a classification task, a set of FC layers that output class probabilities can be added to convolutional layers. Then the network can be trained with cross-entropy loss by applying softmax operation to the output layer.

One of the limitations of the task-specific layers is that they are generally in the form of fully connected layers as explained. To be able to add a fully connected layer after

a convolutional layer, tensors must be transformed into vectors. Unfortunately, vectorization operation forces CNNs to accept images with a certain size. As an example, AlexNet can only be trained on images with a resolution of 224x224 because of its fully connected layers.

2.9 Deep Neural Networks

In NN literature, various types of architectures exist that excel in certain areas. Previously presented FCNs and CNNs are two examples of major architectures. Although each architecture has distinctive differences, they all consist of layers with learnable parameters. As more layers stacked together, the network becomes more complex and requires more computational power. That is why the average number of layers increases with the advances in hardware. Today, networks can be categorized as shallow and deep networks regardless of their architecture. Shallow networks contain few layers while their counterpart has more. There is no sharp separation between shallow and deep networks, but a model with more layers is said to be deeper compared to a few layered network. Recent studies show that deeper networks perform better since they contain more learnable parameters [72].

There are many modern deep CNN architectures that achieve high classification accuracies on benchmark sets like GoogLeNet [73], VGGNet [74] and ResNet [44]. Being deep networks, they contain many layers with a high amount of learnable parameters. AlexNet is considered as the architecture that proved the capabilities CNNs and it contains 5 convolutional layers and 3 fully-connected layers with 60 million learnable parameters. VGGNet was proposed two years after AlexNet and it has 19 layers with 138 million learnable parameters. For visual comparasion, architecture of AlexNet is given in Figure 2.18 and the architecture of VGGNet is given in Figure 2.19.

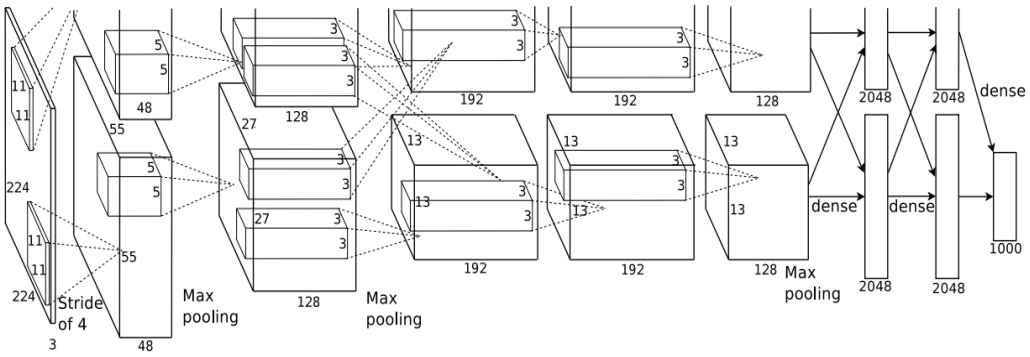


Figure 2.18 : AlexNet architecture [43]

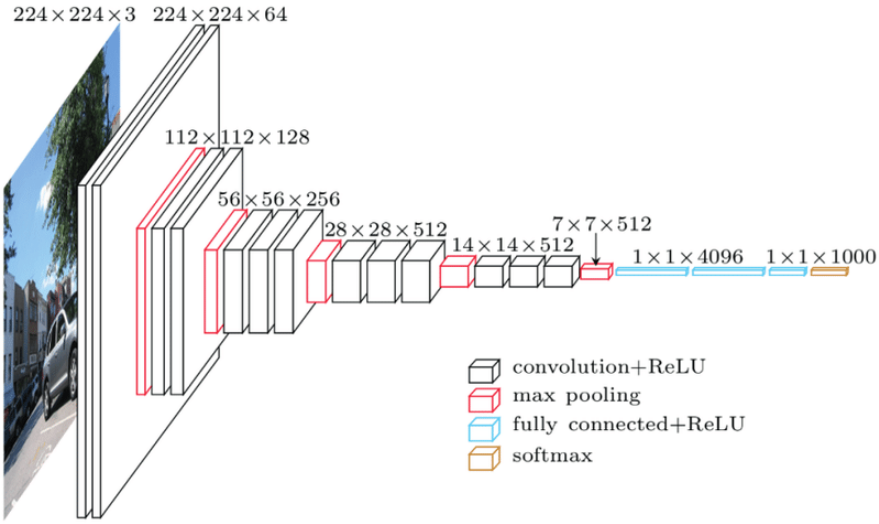


Figure 2.19 : VGGNet architecture [75]

VGGNet showed that the depth of a network was an important factor in the accuracy rates. In Table 2.1, we have given the depth, number of learnable parameters and top 5 error rate for some of the famous CNN architectures. The steady increase in the network depths can be seen from the table.

Table 2.1 : Depth and error rates of major CNN architectures

	Year	Depth	Parameter Count	Error Rate (Top-5)
LeNet	1998	5	0.06 M	MNIST: 0.95%
AlexNet	2012	8	60 M	ImageNet: 16.4%
VGG	2014	19	138 M	ImageNet: 7.3%
GoogLeNet	2015	22	4 M	ImageNet: 6.7%
ResNet-152	2016	50	25.6 M	ImageNet: 3.6%

As the networks become deeper, researchers realized that the classification accuracies saturated after a certain depth [76], [77]. Vanishing gradients were always an issue and deep networks suffered more compared to shallow networks. The issue in deep networks wasn't the usage of sigmoid or hyperbolic activations functions. In fact, even an early architecture such as VGGNet was using the ReLU activation function. The problem was that the gradients were underflowing during BP [78]. As chain rule requires, gradients in shallower layers are calculated by multiplying the gradients in the deeper layers together. If the gradients are smaller than 1, they will diminish in value during BP until the point that the gradients in shallower layers are too small to be represented in the memory. To solve this issue, gradients needed to be transferred into shallower layers without underflowing. Developed by Microsoft Research Team, Residual Networks (or namely the ResNets) solve the underflowing gradients issue by introducing a new concept called residual learning.

As mentioned before, a NN tries to find a mapping function between the input space and the output space and there are even optimal mappings among stacked layers. Let us call $H(x)$ an optimal mapping to be fit by a stack of hidden layers. For the sake of argument, assume that we know the target outputs $\hat{y}$ of this stacked. An optimal fitting would minimize the prediction error given by $F(x) = H(x) - \hat{y}$. In statistics, a residual is the distance vector between the target and prediction, so $F(x)$ represents the residual for the stack. In the original paper of ResNet, it is argued that if a complex function $H(x)$ can be approximated by multiple layers its residual $F(x)$ can also be approximated. So it is safe to let layers update their weights to imitate $F(x)$ instead, but it should also be noted that the desired behavior is still defined by $H(x)$. The difference between residual learning and non-residual learning is that layers approximate $F(x)$ and $H(x)$ is achieved by summing the target values to the residuals in residual learning. Reformulation of $H(x)$ is given in (2.29).

$$H(x) = F(x) + \hat{y} \tag{2.29}$$

In real applications, it is not possible to apply (2.29) since target values are unknown, but residual learning can be used for identity mappings where $\hat{y} = x$. This way (2.29) becomes the following.

$$H(x) = F(x) + x \tag{2.30}$$

The above equation can easily be implemented in CNNs with skip connections. ResNet uses stacks of skipped layers called residual blocks to learn identity mappings. A residual block with two-layer skipping is given in Figure 2.20.

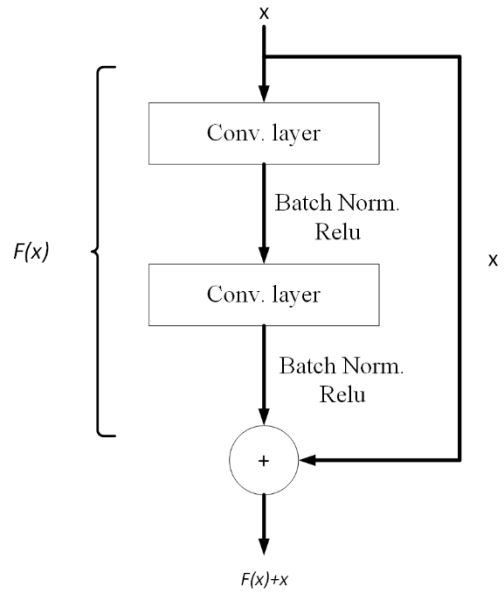


Figure 2.20 : Residual block with two-layer skipping

A residual network can be built by stacking residual blocks stacking one after another. A plain network without any skip connections is given in Figure 2.21. This plain network contains 34 layers and is the base architecture of the 34 layered version of the ResNet. The ResNet-34 has the same layers, but it also includes skip connections as shown in Figure 2.22.

Figure 2.21 : Base plain network [44]

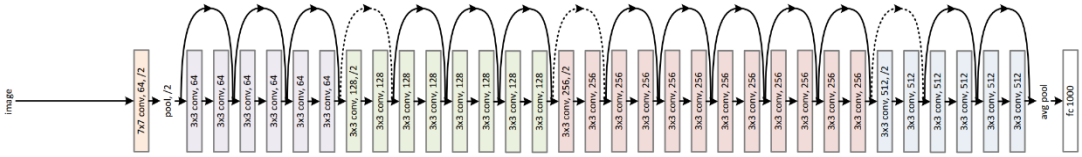


Figure 2.22 : ResNet-34 architecture [44]

The importance of identity mappings is that they can transfer the gradients to shallower layers without diminishing them. For example, if there is a skip connection from the end of i^{th} layer to the end of $(i+2)^{th}$ layer, gradients in layer (i+2) can be directly

transferred to i^{th} layer during the BP. To mathematically represent, the gradient of a residual block's input is given in (2.31).

$$\frac{\partial J}{\partial x} = \frac{\partial J}{\partial H}\frac{\partial H}{\partial x} = \frac{\partial J}{\partial H}\left(\frac{\partial F}{\partial x} + 1\right) = \frac{\partial J}{\partial H}\frac{\partial F}{\partial x} + \frac{\partial J}{\partial H} \tag{2.31}$$

Even with residual blocks, gradients can still vanish if the network is deep enough. Skip connections are just providing a way to add more layers to an existing architecture until underflowing happens. If more layers are skipped at each residual blocks, network depth can be further increased. In fact, the residual block given in Figure 2.19 belongs to the ResNet-34. There are also 50 layered (ResNet-50), 101 layered (ResNet-101) and 152 layered (ResNet-152) versions of ResNet and they use the residual block given in Figure 2.23.

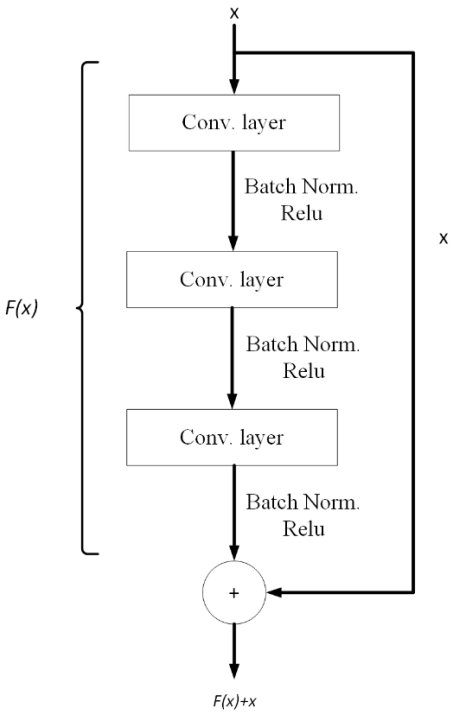


Figure 2.23 : Residual block with three-layer skipping

2.10 Transfer Learning

In computer vision, a feature extraction algorithm can be integrated into various classification or detection tasks where different object sets are handled. A good feature extraction algorithm should be task agnostic while mapping the input space to a feature space. We have already mentioned that a CNN consists of non-task specific and task specific layers. The non-task specific layers serve the purpose of feature extraction and

they are in the form of convolutional and pooling layers. Task-specific layers on the other hand usually consist of FC layers and solves a classification or detection task. The task specific layers are further emphasized in Figure 2.24.

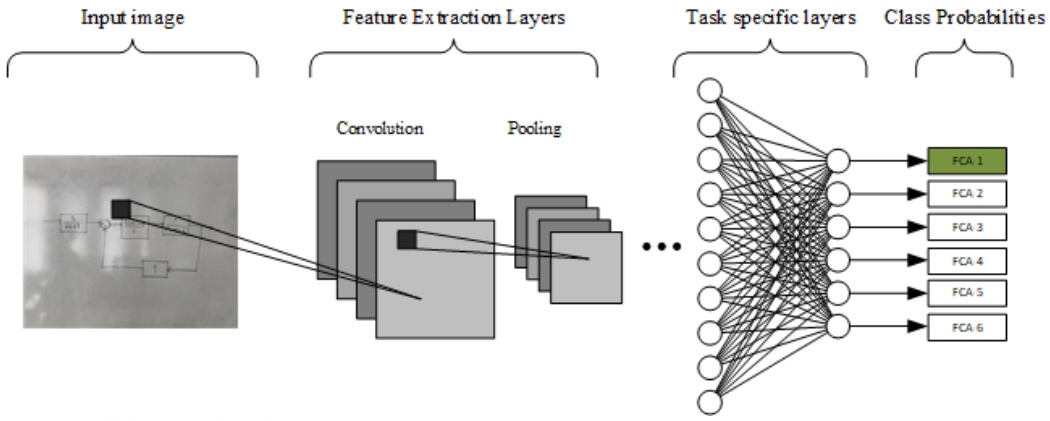


Figure 2.24 : Feature extraction and task specific layers of a CNN

Having a seperation between the feature extraction and task specific layers provides a way to adapt an existing network to a different task just by replacing its feature extraction layers. By only replacing the task specific layers, the existing knowledge of feature extraction can be transferred into any computer vision task. The newly added layer would still require further training, but this approach is more efficient compared to training a new model from scratch, since the task-specific layers form a small portion of the complete architecture. Adapting an existing network for new tasks is called transfer learning and can be used to incorporate existing deep CNNs.

The feature extraction layers of a CNN is given below where θ_{fe} represents the learnable parameters of these layers and z_{fe} is the extracted features.

$$z_{fe} = g_{\theta_{fe}}(x) \tag{2.32}$$

The complete CNN model can be expressed as in (2.33) where θ_{ts} is the learnable parameters of the task-specific layers.

$$y = f_{\theta_{ts}}(g_{\theta_{fe}}(x)) \tag{2.33}$$

The transfer learning approach is to replace $f_{\theta_{ts}}$ with a new one as shown below where θ'_{ts} is the learnable parameters of new task-specific layers.

$$y = f'_{\theta'_{ts}}(g_{\theta_{fe}}(x)) \tag{2.34}$$

Re-training the modified network to learn θ'_{ts} is called fine-tuning. Since the feature extraction layers are already optimized, gradient calculation is not needed for θ_{fe} during the BP stage of the fine-tuning. Layers or weights that aren't being updated during the BP stage are said to be frozen and it is common practice to freeze θ_{fe} in fine-tuning. This way a training iteration (FP and BP together) completes faster compared to training from scratch. Even if θ_{fe} is not, the network will converge much faster with fewer epoch counts.

Using a pre-trained network not only saves training time, it also enables high accuracy performance for a network without needing a large amount of data [79]. In Table 2.2, we have given the number of samples for some of the popular image datasets. From the table it can be seen that these datasets contain very large amounts of samples and it is not always possible to construct such large datasets. For example in medical imaging problems number of collected samples tend to be smaller compared to other tasks. The transfer learning approaches is therefore essential for cases where datasets are not sufficiently large.

Table 2.2 : Number of samples for some of the image datasets

	Number Of Samples	Size in Memory
MNIST	70 000	50 MB
Fashion MNIST	70 000	30 MB
CIFAR-10	60 000	170 MB
COCO	$330x10^3$	25 GB
ImageNet	$1.5x10^6$	150 GB

3. DEEP LEARNING BASED PIPELINE

In control lectures, lecturers frequently use plots in time and frequency domains to better explain core concepts. Although the dynamics of a system, its stability margins can be expressed mathematically it is also convenient to visualize these metrics. Today, control lectures are usually conducted in classroom environments where lecturers heavily use drawing boards. The corresponding plots for the handled systems are drawn to these drawing boards. While drawing boards have their advantages, plotting a system response or similar graphs may prove challenging and time-consuming. It is also not possible to correctly scale these visuals and only their approximations can be presented to students. In some cases, the plots can't even be drawn if the handled system is not simple enough. A better approach is to define and analyze systems in a simulation environment such as Matlab®, but then the lecturer would be deprived of the advantages of using a whiteboard. Instead, the block diagrams drawn on the whiteboard can be recognized with a computer and be transferred to a simulation program for further analysis.

In this thesis, we proposed a DL based pipeline that can transfer an HFCA to the Matlab® environment. An overview of the DL based pipeline is given in Figure 3.1 that is summarized with the following steps:

Step-1: Recognizing the structure of the HFCA with DL.

Step-2: Detecting TF blocks using image processing.

Step-3: Segmenting and recognizing the characters with DL.

Step-4: Constructing symbolic expressions from the recognized characters to construct continuous-time TFs in Matlab®.

Step-5: Generating the recognized continuous-time HFCA in Matlab®.

In **Step-1** and **Step-3** of the proposed DL based pipeline, the following two PR problems are defined.

- HFCA Recognition (HFCAR)

- Handwritten Character Recognition (HCR)

The main challenge arises from the quality of the lecturer's handwriting and lighting conditions, especially in HFCAR and HCR problems. To handle such uncertainties, we used the transfer learning approach of ResNet-50 described to construct deep CNNs.

To train the deep CNNs, an image dataset collected from lecturers of Control System Design courses in a small-sized classroom environment in the presence of different lighting conditions has been constructed. The implementation of deep CNNs has been done with the Deep Learning Toolbox™ to have easy integration with Control System Toolbox™ and Simulink™ of Matlab®. Both of the networks are trained with SGDM as the optimizer for the sake of simplicity as it has only a single hyperparameter, which is α. We also used the cross-entropy loss function with defined in (2.12) L2 regularization defined in (3.1) where λ is the regularization term, y_q is the predicted output, $\hat{y}_q$ is the target label and Q is the total number of classes.

$$L(y_q, \hat{y}_q) = -\frac{1}{Q}\sum_{q=1}^{Q}\left[y_q \log \hat{y}_q + (1 - y_q)\log(1 - \hat{y}_q)\right] + \lambda\frac{w^T w}{2} \qquad (3.1)$$

In this chapter, a detailed description of the steps of DL based pipeline as shown in Figure 3.1 is given. For illustrative purposes, the steps of pipeline are illustrated on an example HFCA which is shown in Figure 3.2.

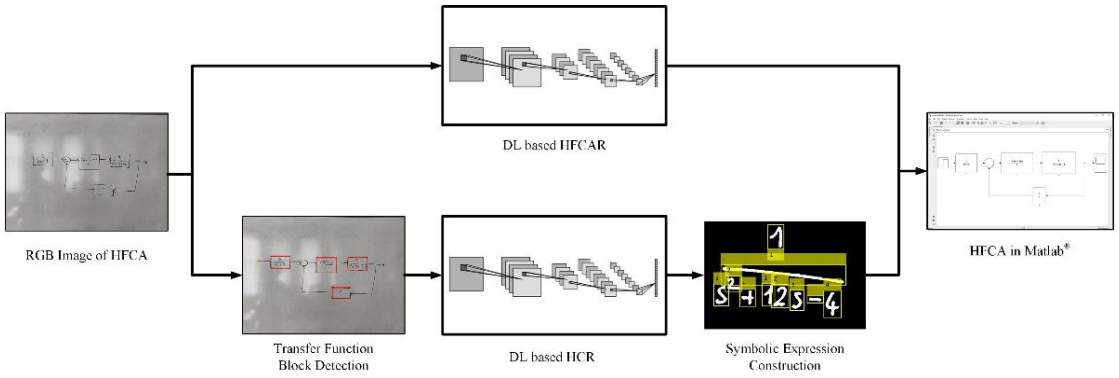


Figure 3.1 : Overview of the proposed DL pipeline

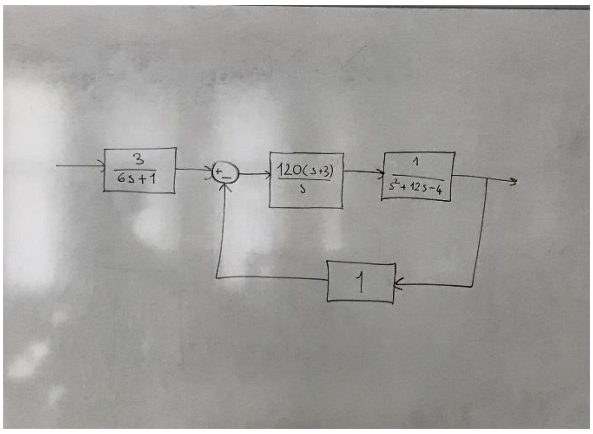


Figure 3.2 : An example HFCA

3.1 Handwritten Feedback Control Architecture Recognition

The first task to be accomplished in the proposed DL based pipeline is to solve the HFCAR problem to identify one of the architectures shown in Figure 1.1 (i.e. Q=6 classes). In this context, we constructed a dataset with 306 RGB HFCA images with a resolution of 4032x3024 which were captured from an actual whiteboard. The dataset is then labeled manually with the classes and is then split as 216 images (36 per class) for training, 36 images (6 per class) for validation, and 54 images (9 per class) for testing. Examples from each class are given in Fig. 3.3.

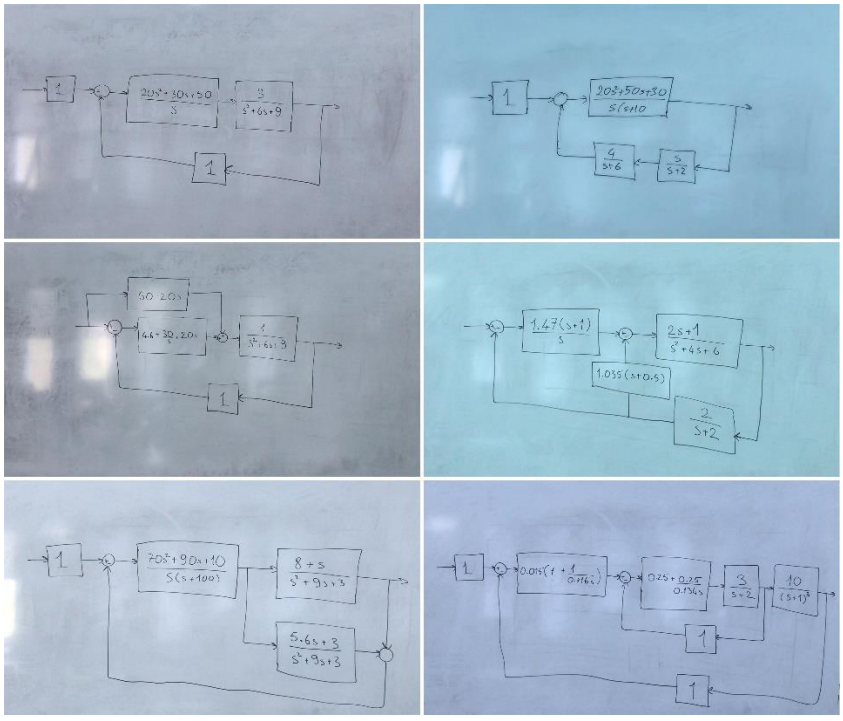


Figure 3.3 : Example HFCA images

The multi-label HFCAR problem is solved with a ResNet-50 based CNN as described in chapter 2.3. In learning of the deep CNN, all images are resized to a resolution of 224x224 without any further pre-processing. Furthermore, we employed the online data augmentation method to create artificially modified versions of images to increase the size of training dataset by slightly rotating and scaling each image at each training epoch. The DL hyperparameters are set as: 100 epochs, minibatch size of 1, and a learning rate of 10^{-3} with a drop rate factor of 0.1 at every 10 epochs.

The best and mean training, validation and testing accuracies over 5 experiments are tabulated in Table 3.1. The mean training and validation accuracy values are given in Figure 3.3 (only the first 3000 iterations are given). It can be concluded that the performance of the deep CNN is statisfactory as it resulted with mean testing accuracy value of 89.25%.

Table 3.1 : Performance of deep CNN for HFCAR

	Best Accuracy	Mean Accuracy
Training	100%	100%
Validation	97.2%	94.4%
Testing	94.4%	89.25%

3.2 Transfer Function Block Detection

After recognizing the class of the HFCA, TF blocks in the image that are enclosed with rectangular shapes must be extracted. Taking into account the fact that TF blocks are represented as rectangles in FCAs, an algorithm that can detect rectangles can be used for this task. Detection of a rectangle simply means to find the position and the dimensions of the smallest box that contains it. As it can be guessed, TF blocks on an HFCA may not be perfect rectangles unlike their bounding boxes (BB).

Finding the BBs of TF blocks is accomplished by an algorithm that contains image processing techniques like binarization, edge detection, and connected component analysis. The steps of this algorithm are explained in detail in the following sections.

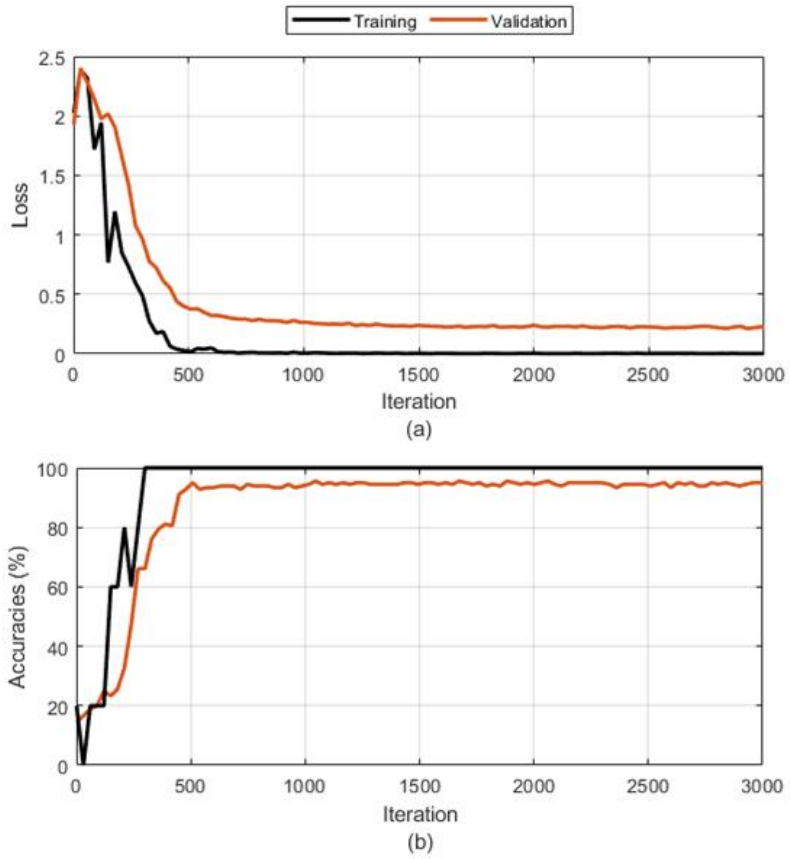


Figure 3.4 : HFCAR mean (a) loss values (b) accuracy values

3.2.1 Binarization of HFCA

Transforming a colored image into an array of logical elements is required for applying morphological operations [52]. A binary image can be obtained by thresholding the input image like the following where x_{ij} is the element of the input image at position (i,j), b_{ij} is the corresponding element in the resulting binary image and T is the threshold value.

$$b_{ij} = \begin{cases} 1 \; if \; x_{ij} \geq T \\ 0 \; if \; x_{ij} < T \end{cases} \tag{3.2}$$

The binarization technique with a single threshold value given in (3.2) is called global thresholding. For images where shadows and nonuniform illumination exist, a global thresholding method will not be able to give satisfying results.

To deal with faded fonts and light reflections like in Figure 3.2, a convolution operation with zero padding is employed with the following *n* by *n* kernel k_{ij}:

$$k_{ij} = \begin{cases} 1/n & i = \dfrac{n-1}{2}, j = \dfrac{n-1}{2} \\ -1 & else \end{cases} \tag{3.3}$$

If a pixel's intensity value is darker when compared to its *n* by *n* neighborhood, the result of the convolution operation has a positive value at that pixel's location. We defined a positive threshold value *D* to determine the binary version of the input image. Mathematical expression for the used thresholding method is given in (3.4) where $c = x * k$.

$$b_{ij} = \begin{cases} 1 \; if \; c_{ij} \geq D \\ 0 \; if \; c_{ij} < D \end{cases} \tag{3.4}$$

By trial and error, we found that a 15x15 kernel with $D = 15$ is a suitable value for our purposes. The binary version of Figure 3.2 is given in Figure 3.5.

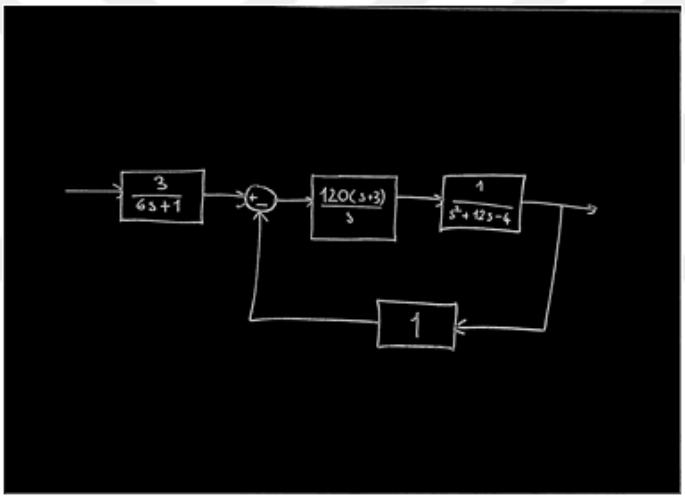


Figure 3.5 : Binarized image

3.2.2 Character and Noise Removal

In the next step of the rectangle detection, the characters inside the rectangles are removed. In FCA diagrams, all the lines except the characters and symbols inside TF and operation blocks are connected. When the characters and all the noise are removed, it is expected to end up with a binary image where only the mentioned lines remain.

It can safely be assumed that the biggest connected component (i.e. connected component with highest pixel count) in Fig 3.5 is the diagram itself. With that assumption, undesired pixels can be removed by finding the biggest connected component. Using 8 connectivity, the resulting segmentation of the HFCA image is shown in Figure 3.6. Note that it is very likely that the lecturer might not connect all

the blocks perfectly. Thus, a dilation operation was also applied to close such disconnections.

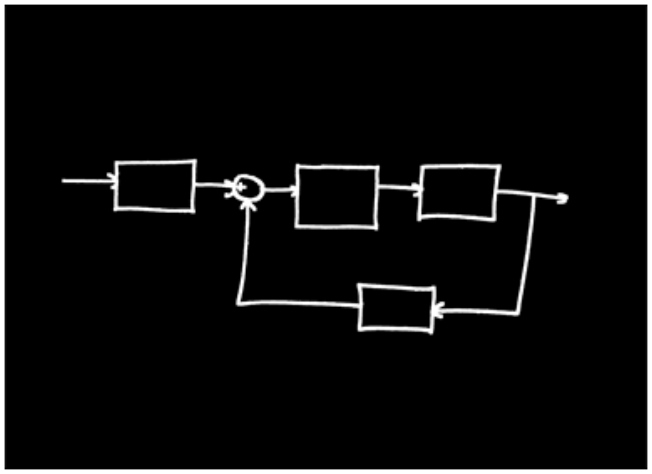


Figure 3.6 : Character and noise filtered binary image

3.2.3 Filling Transfer Function Blocks

In rectangle detection, the aim is to have filled blobs where rectangles reside. A filling can't be applied directly to Figure 3.6 or it will result in a single large blob. Instead, the Canny edge detection algorithm is deployed to remove the outermost edge loop that wraps the whole diagram first. Then, a morphological fill algorithm is employed for the remaining image. The result of the filling operation is given in Figure 3.7.

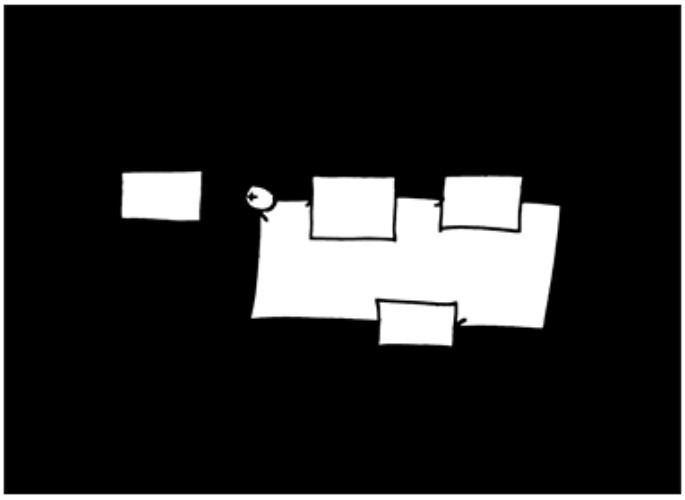


Figure 3.7 : Filled binary image

3.2.4 Rectangle Extraction

As it can be seen from Figure 3.7, not all the remaining white regions belong to a TF block. By performing a shape analysis, desired rectangles can be extracted. For the

shape analysis, the circularity and rectangularity of each connected component is calculated. The circularity measure (C) is defined in (3.5) where A is the area and P is the perimeter of the region in terms of pixels.

$$C = \frac{4\pi A}{P^2} \tag{3.5}$$

The rectangularity measure (R) is given in (3.6) where L_{major} is the length of the major axis and L_{minor} is the length of the minor axis of the region.

$$R = \frac{A}{L_{major} L_{minor}} \tag{3.6}$$

Regions that satisfy the conditions given in (3.7) are selected as rectangles.

$$\begin{gathered} C \leq 0.9 \\ 0.7 \leq R \leq 1.5 \end{gathered} \tag{3.7}$$

Once the rectangle regions are segmented, their BB informations are saved for further use. Segmented regions for Figure 3.2 are shown in Figure 3.8.

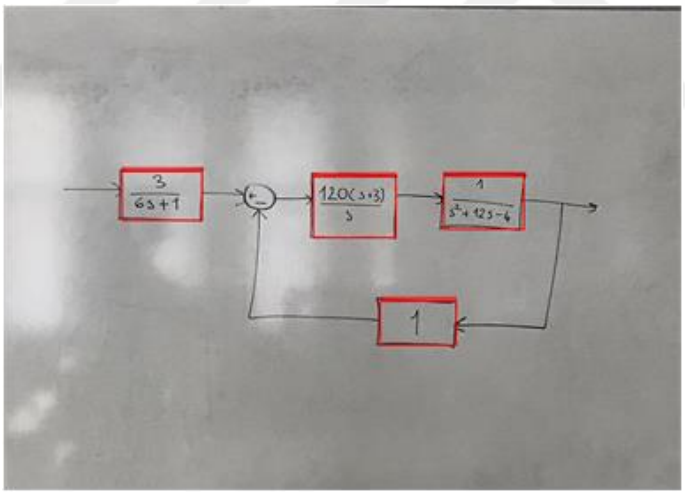


Figure 3.8 : Detected rectangles

3.2.5 Handwritten Character Recognition

In the next step of the pipeline, characters inside the TF blocks are recognized using a deep CNN. To be able to classify characters with DL, they must be located first. Instead of searching for characters over the whole image, small patches are cropped from the input image for each TF block using their BB information. An example cropped image patch is shown in Figure 3.9. A character segmentation algorithm can

then be employed on individual patches to be able to tell which character belongs to which TF block.

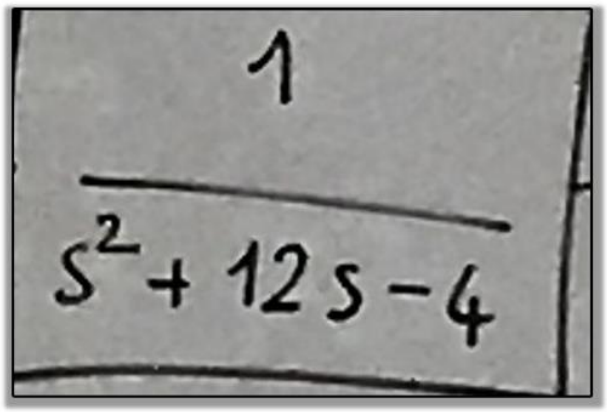


Figure 3.9 : Extracted TF block image

The adopted character segmentation method starts with the binarization of the extracted image patch. Although a binarized version of the whole image already generated in the rectangle detection, we have seen that the previously mentioned binarization technique doesn't provide satisfactory results for small image patches. Global binarization technique as described in (3.2) is adopted instead with a T of 125 (half instensity value in RGB images). Binary version of Figure 3.9 is given in Figure 3.10. After the binarization, connected components are found in the patches. It is assumed that the characters do not touch each other and consists of single blobs and because of these assumptions all the found components are treated as characters. A morphological close operation is employed before to deal with possible discontinuities. Once the characters are segmented and their BBs are found, they are cropped and resized into 224x224 resolution to be fed into a deep CNN for labeling.

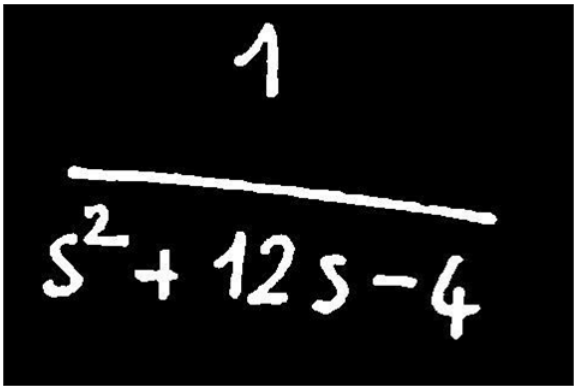


Figure 3.10 : Segmented TF block image

The characters to be recognized are digits (0-9), arithmetic operators (‘+’, ‘-‘, ‘x’, ‘*’), round and square bracket pairs, and the ‘s’ and ‘.’ characters, which makes in total $Q = 20$ classes. In the dataset construction for HCR, we used the whole constructed HFCA dataset containing 306 samples by first extracting TF blocks then segmenting characters via the aforementioned approach. It is worth to underline that we have observed that lighting conditions had a much bigger impact on small images and therefore we decided to extract the characters from the binary version of the HFCA images.

Each extracted character is labeled manually by the authors. An example of the used character images is shown in Figure 3.11. Moreover, we would like to point out that the dataset is enriched with extra handwritten character images to end up with evenly distributed samples for each class. The resulting HCR dataset has 3920 images in total which is split as 155 images per class for training, 32 images per class for validation and 9 images per class for testing.

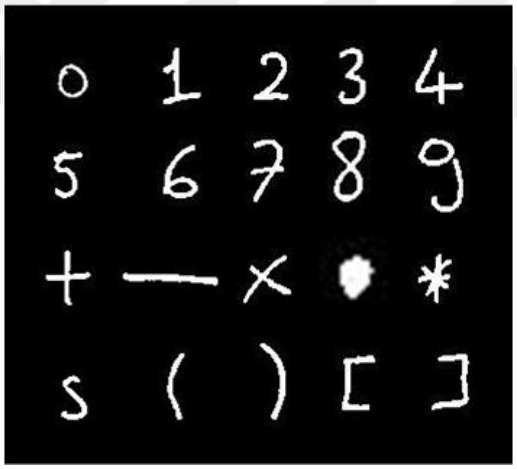


Figure 3.11 : Example character images

The HCR problem is solved with a deep CNN that is trained as described previously with hyperparameter settings of 50 epochs, minibatch size of 4, and a learning rate of 10^{-3} with a drop rate factor of 0.1 at every 10 epochs. We also employed online data augmentation during the learning.

The best and mean training, validation, and testing accuracies over 5 experiments are given in Table 3.2. The mean training and validation accuracy values are illustrated in Figure 3.12 (only the first 500 iterations are given). It can be seen that the learning performance of the ResNet-50 based deep CNN is satisfactory since it resulted in a

mean accuracy of more than 96%. As it can be seen from Figure 3.13, the trained deep CNN is capable to successfully label the characters of the segmented TF image.

Table 3.2 : Performance of the deep CNN for HCR

	Best Accuracy	Mean Accuracy
Training	100%	100%
Validation	98.39%	97.2%
Testing	96.17%	96.08%

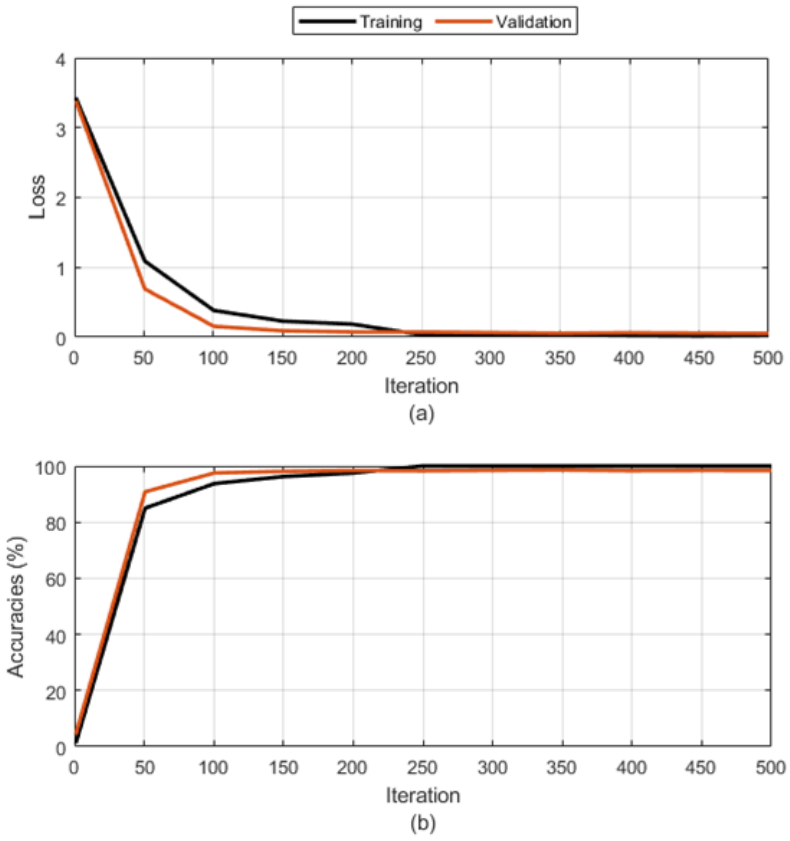


Figure 3.12 : HCR mean (a) loss values (b) accuracy values

3.3 Symbolic Expression Construction

A single input-single output TF is expressed as a ratio of two polynomials, namely the numerator and denominator. In Matlab®, a TF can be defined using the coefficients of these polynomials. For that reason, symbolic expressions from the unordered but labeled characters for each TF are constructed. With Matlab®'s Symbolic Math Toolbox™, required coefficients can be found from the constructed expressions.

First step in symbolic expression construction is to build the equation string using the recognized characters as illustrated in Figure 3.13. This process starts with finding the fraction signs that seperates the numerator from the denominator. Although not very likely, we consider the cases where multiple fractional terms exist in TFs. It is not safe to consider all the characters with "-" labels as the same label is used also for the minus sign. To distinguish the numerator and denominator polynomials of the recognized TF, the characters labeled with the class "-" are initially examined. To differentiate whether this label represents the subtraction or fraction operator, we simply checked if there is another labeled character above and below of its position. If this condition results in a non-empty set, then we concluded that the labeled character is a fraction symbol that separates the numerator and denominator of TF as shown in Figure 3.14. Then, the remaining characters are allocated as elements of the sets defining the numerator and denominator part of TF with respect to their position. In this step, we have also handled exponent characters in the polynomials. A character is labeled as an exponent if it is positioned (slightly) above of its preceding character. As shown in Figure 3.14, the first character labeled as "2" is an exponent, because of its relative vertical position to the first occurring "s" character. Thus, we add a caret character (^) between the base and the exponent characters.

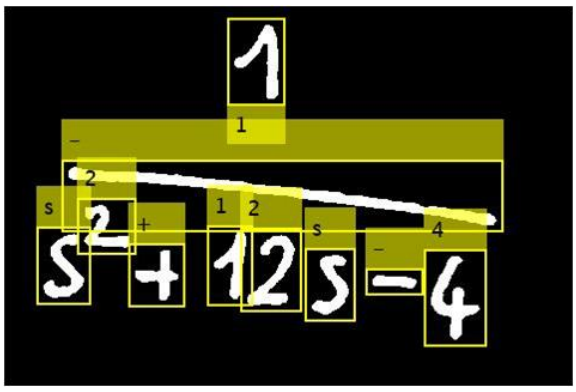
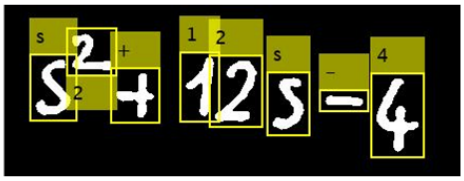


Figure 3.13 : Labeled TF block image

['1']

['/']



['s', '^', '2', '+', '1', '2', 's', '-', '4']

Figure 3.14 : Labeled numerator and denominator

Once all the characters are allocated in order, the character sets of the numerators and denominators are turned into strings and then merged with a division symbol to obtain a string expression of the extracted TF image. This string expression is then transformed into a symbolic expression to obtain TF representation in which the "s" character becomes the only symbolic variable. Finally, the coefficients of the symbolic expressions of the numerator and denominator are extracted to define TFs in Matlab®. A screenshot of transfer function construction in Matlab® is given in Fig 3.15.

```
>> [num, den] = numden(symbolic);
G = tf(double(coeffs(num, 'All')),...
double(coeffs(den, 'All')))

G =

        1
  --------------
  s^2 + 12 s - 4

Continuous-time transfer function.
```

Figure 3.15 : TF in Matlab®

3.4 Feedback Control Architecture Generation In Matlab®

Once the deep CNN trained to solve HFCAR problem recognizes the FCA class (all classes are shown in Figure 1.1), the extracted TF representations of the image have to be matched with their appropriate slots in the FCA. In this context, using the center coordinates of the extracted TF blocks that are calculated via their BB information, we assign the extracted TFs that have similar vertical positions in the image to the same path, match and name them with the corresponding TFs (such as G(s), C(s), H(s),…) defined in the FCAs through their horizontal positions in the image. Note that, we defined a path as a horizontal route a signal can follow in FCAs (i.e. feedforward or feedback path).

In order to provide a clear understanding, let us explain the matching and naming of the TFs on the FCA-1 structure for illustrative purposes. As can be observed from Figure 1.1a, the FCA-1 has two paths including a feedforward path with 3 TFs (F(s), C(s) and (G(s)) at top and a feedback path with a single TF (H(s)). If a HFCA is recognized as FCA-1 and all the TFs are extracted in the image frame, then we end up with 3 TFs aligned and a single TF near the upper and lower half of the image, respectively. In the FCA-1, since we know that the prefilter F(s) is the first TF in the feedforward path, we name and match the extracted TF with smaller horizontal coordinate at the upper path as the TF F(s), while the next one to its right as the compensator TF C(s) and the rightmost one as the plant TF G(s). The remaining extracted TF is directly matched and named as the TF that defines the sensor dynamics H(s) since the feedback path of FCA-1 contains a single TF. In a similar manner, the rest 6 FCAs are matched with the extracted TFs in the proposed DL based pipeline. Now, the FCA constructed from HFCA can be directly processed and analyzed in Matlab® since all the TFs of FCA are defined in the workspace of Matlab®.

In order to analyze the FCAs via the Control System Designer App™ of Matlab®, we define a Matlab object in which the recognized FCA class is defined with the extracted and matched TFs and then import it to the graphical user interface of the application. As shown in Figure 3.16, the user can now not only visualize the control system in the time and frequency domains but also tune compensator C(s) if desired.

It is also worth mentioning that the DL based pipeline automatically generates also a Simulink™ diagram as shown in Figure 3.16d which can be directly used for

simulation purposes. To accomplish such a goal, we created template Simulink files in advance for all the FCAs in which all TFs are named as defined in Table I. In the template Simulink files, the simulation time and solver options are defined with the default settings of Simulink™. Once the HFCA is recognized, the corresponding Simulink file is automatically opened and the matched TFs are loaded into the file.

3.5 Real-Time Performance of the Pipeline

To test the real-time performance of the pipeline, a series of experiments are conducted in a small-sized classroom containing a whiteboard and a projection board. For the experiments, we created a simple program in Matlab® that detects the HFCA currently drawn on the whiteboard in a continuous loop. At each time-step, a snapshot is taken from a camera that is fixed to the whiteboard. The image is then fed to the pipeline and if a valid FCA is detected the program sends it to the Simulink™ environment.

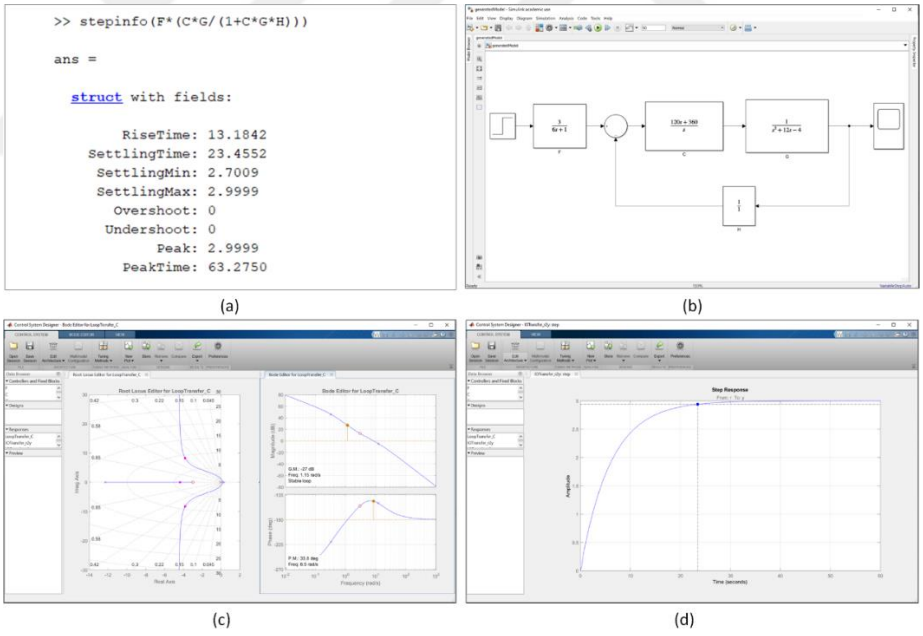


Figure 3.16 : (a) Matlab command window (b) Step Response (c) Root-Locus and Bode plots (d) Simulink diagram

The program does not terminate after recognizing the HFCA and completing the pipeline. It repeats the first step of the pipeline (HFCAR) at the start of each iteration. If a different HFCA is successfully recognized it completes the remaining steps (block detection, HCR, symbolic expression construction) and opens a new Simulink™ model. If the recognized HFCA is the same as before, it checks the TF blocks for any

possible changes. In case of modification inside a TF block, HCR and symbolic expression construction steps are repeated for that specific block. This way, we were able to avoid repeating the block detection step at each iteration. Flowchart of the real-time program is given in Figure 3.17.

We also projected the Simulink™ window with a projection device to a projection board as in Figure 3.18a. We observed that the pipeline takes roughly a second to generate a result and concluded that the pipeline can handle inter-quality uncertainty as it is capable to recognize FCAs from different users in real-time. It can also handle intra-quality uncertainties as it can recognize FCAs from in the presence of various lighting conditions in real-time. Snapshots from the real-time experiment are given in Figure 3.1.

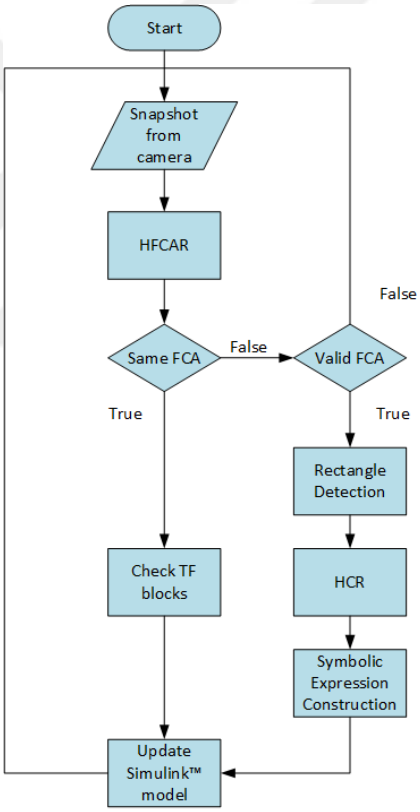
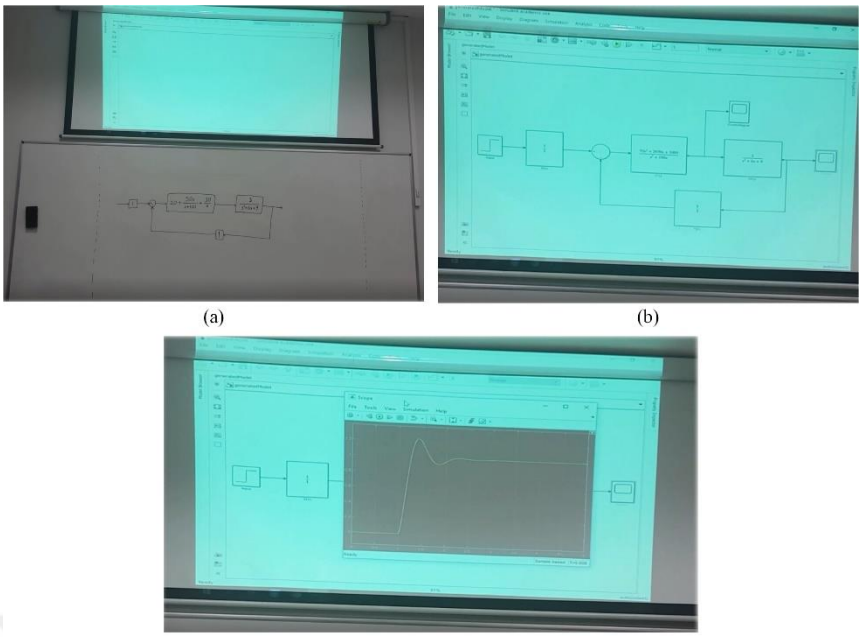


Figure 3.17 : Flowchart of the real-time application



(a) (b)

(c)

Figure 3.18 : (a) Experiment environment (b) Projected Simulink™ window (c) Simulation result

4. CONCLUSION

In this thesis, we established a DL based pipeline that can recognize an HFCA along with the TFs it contains in real-time. We also presented the DL concepts used in this work. We believe that being able to transform an HFCA to a simulation environment such as Matlab® provides the opportunity to the lecturers/researchers to easily visualize and analyze HFCAs during a lecture. The proposed pipeline has been accomplished by integrating frameworks of deep learning and using various PR and image processing techniques. We provided all the details and necessary information about each step of the proposed DL based pipeline. For steps that include classification with DL, we integrated deep CNNs to solve the corresponding PR problems (HFCAR and HCR). Instead of designing a new architecture, we used the transfer learning approach. We selected ResNet-50 as the base model since we believe that it is capable of handling the intra-quality and inter-quality uncertainties that mainly occur due to handwriting quality of the lecturers and lighting conditions. We also created separate datasets for both of the recognition problems for the fine-tuning with the help of different lecturers.

We tested the capabilities of our pipeline by conducting experiments in a small-sized classroom with a camera fixed on a whiteboard. During the experiments, we tried to transfer the currently drawn HFCA on the whiteboard to the Matlab® in real-time. The experiments showed that the DL based pipeline is a powerful tool to visualize and analyze HFCAs as it can recognize the FCA with high accuracy in a short amount of time. It is also worth underlining that the developed DL based pipeline is capable to detect changes in each TF block and it can update the current representation. This way lecturers can show how poles and zeros affect a system, adjust the controller parameters, or employ similar alterations to further analyze an FCA.

We think that the DL based pipeline has the potential to ease the difficulty in teaching control systems as real-time visualizations of control systems and simulations are generated as the lecturer is sketching FCAs during the lectures.

REFERENCES

[1] **Prendergast, D. P., & Eydgahi, A. M.** (1993). EDCON: An educational control systems analysis and design program. *IEEE Trans. Educ.*, 42-44.

[2] **Kheir, N. A., Astrom, K. J., Austander, D., Cheok, K. C., Franklin, G. F., Masten, M., & Rabins, M.** (1996). Control systems engineering education. *Automatica*, 147-166.

[3] **Johansson, M., Gafvert, M., & Astrom, K.** (1998). Interactive tools for education in automatic control. *Control Systems IEEE*, 33-40.

[4] **Kroumov, V., Shibayama, K., & Inoue, A.** (2003). Interactive learning tools for enhancing the education in control systems. *Frontiers in Education*, 23-28.

[5] **Rodriguez, A. A., DeHerrera, M. F., & Metzger, R. P.** (1996). An interactive MATLAB-based tool for teaching classical systems and controls. *Technology-Based Re-Engineering Engineering Education Proceedings of Frontiers in Education FIE'96 26th Annual Conference*, (s. 624-627). Salt Lake City.

[6] **Chow, J. H., & Cheung, K. W.** (1992). A toolbox for power system dynamics and control engineering education and research. *IEEE Trans. Power Syst., 7(4)*, 1559-1564.

[7] **Dormido, S., Dormido-Canto, S., Dormido, R., Sanchez, J., & Duro, N.** (2005). The role of interactivity in control learning",. *Int. J. Eng. Educ., 21(6)*, 1122-1133.

[8] **Vargas, H., Moreno, J. S., Jara, C., Candelas, F., Torres, F., & Dormido, S.** (2011). A network of automatic control web-based laboratories. *IEEE Trans. Learn. Technol., 4(3)*, 197-208.

[9] **Maiti, A., Zutin, D. G., Wuttke, H., Henke, K., Maxwell, A. D., & Kist, A. A.** (2018). A framework for analyzing and evaluating architectures and control strategies in distributed remote laboratories. *IEEE Trans. Learn. Technol., 11(4)*, 441-455.

[10] **Bencomo, S. D.** (2004). Control learning: Present and future. *Annu. Rev. Control, 28(1)*.

[11] **Sanchez, J., Dormido, S., Pastor, R., & Morilla, F.** (2004). A Java/MATLAB-based environment for remote control system laboratories: Illustrated with an inverted pendulum. *IEEE Trans. Educ., 47(3)*, 321-329.

[12] **Leva, A., & Donida, F.** (2008). "Multifunctional remote laboratory for education in automatic control: The CrAutoLab experience. *IEEE Trans. Ind. Electron, 55(6)*, 2376-2385.

[13] **Dorf, R. C., & Bishop, R. H.** (2011). *Modern Control Systems.* Pearson.

[14] **The MathWorks Inc.** (2019). *MATLAB® Control System Toolbox™ Users Guide.* https://www.mathworks.com/help/pdf_doc/control/control_ug.pdf adresinden alındı

[15] **Kara, L., & Stahovich, T.** (2005). An Image-Based Trainable Symbol Recognizer for Hand-Drawn Sketches. *Computers & Graphics, 29(4)*, 501-517.

[16] **Bresler, M., Phan, T. V., Prusa, D., Nakagawa, M., & Hlavác, V.** (2014). Recognition System for On-Line Sketched Diagrams. *14th International Conference on Frontiers in Handwriting Recognition*, (s. 563-568). Heraklion.

[17] **Chen, Q., Shi, D., Feng, G., Zhao, X., & Luo, B.** (2015). On-line handwritten flowchart recognition based on logical structure and graph grammar. *5th International Conference on Information Science and Technology*, (s. 424-429). Changsha.

[18] **Hammond, T., & Paulson, B.** (2011). Recognizing sketched multistroke primitives. *ACM Trans. Interact. Intell. Syst.*,.

[19] **Kara, L., & Stahovich, T.** (2004). Hierarchical parsing and recognition of hand-sketched. *Proc. 17th ACM Symp. User Interface Software and Technology*, (s. 13-22). Santa Fe.

[20] **Davis, L. S.** (1975). A survey of edge detection techniques. *Computer Graphics and Image Processing, 4(3)*, 248-270.

[21] **Teow, L., & Loe, K.** (2002). Robust vision-Based features and classification schemes for off-Line handwritten digit recognition. *Pattern Recognition, 35(11)*, 2355-2364.

[22] **Holmström, L., & Koistinen, P.** (2010). Pattern recognition. *Wiley Interdisciplinary Reviews: Computational Statistics, 2(4)*, 404-413.

[23] **Mantas, J.** (1986). An overview of character recognition methodologies. *Pattern Recognition, 19(6)*, 425-430.

[24] **Atal, B., & Rabiner, L.** (1976). A pattern recognition approach to voiced-unvoiced-silence classification with applications to speech recognition. *IEEE Transactions on Acoustics, Speech, and Signal Processing, 24(3)*, 201-212.

[25] **Azad, R., & Shayegh, H. R.** (2013). New method for optimization of license plate recognition system with use of edge detection and connected component. *ICCKE 2013*, 21-25.

[26] **Cheng, M.-M., Zhang, Z., Lin, W.-Y., & Torr, P.** (2014). BING: Binarized normed gradients for objectness estimation at 300fps. *IEEE Conference on Computer Vision and Pattern Recognition* (s. 3286-3293). Columbus: IEEE.

[27] **Uijlings, J., & van de Sande, K. G.** (2013). Selective Search for Object Recognition. *Int J Comput Vis 104*, 154-171.

[28] **Zhou, H., Yuan, Y., & Shi, C.** (2009). Object tracking using SIFT features and mean shift. *Computer Vision and Image Understanding, 113(3)*, 345-352.

[29] **Bezdek, J. C.** (1992). On the relationship between neural networks, pattern recognition and intelligence. *International Journal of Approximate Reasoning, 6(2)*, 85-107.

[30] **Wang, J., Chen, Y., Hao, S., Peng, X., & Hu, L.** (2019). Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letter, 119*, 3-11.

[31] **Lu, Q., Liu, C., Jiang, Z., Men, A., & Yang, B.** (2017). G-CNN: object detection via grid convolutional network. *IEEE Access, 5*, 24023-24031.

[32] **Beke, A., & Kumbasar, T.** (2019). Learning with type-2 fuzzy activation functions to improve the performance of deep neural networks. *Engineering Applications of Artificial Intelligence, 85*, 372–384.

[33] **Tsujii, O., Freedman, M. T., & Mun, S. K.** (1999). Classification of microcalcifications in digital mammograms using trend-oriented radial basis function neural network. *Pattern Recognition*, 891-903.

[34] **Reddick, W. E., Glass, J. O., Cook, E. N., Elkin, T. D., & Deaton, R. J.** (1997). Automated segmentation and classification of multispectral magnetic resonance images of brain using artificial neural networks. *IEEE Transactions on Medical Imaging, 16(6)*, 911-918.

[35] **Auld, T., Moore, A. W., & Gull, S. F.** (2007). Bayesian Neural Networks for Internet Traffic Classification. *IEEE Transactions on Neural Networks, 18(1)*, 223-239.

[36] **Sigillito, V. G., Wing, S. P., Hutton, L. V., & Baker, K. B.** (1989). Classification of radar returns from the ionosphere using neural networks. *Johns Hopkins APL Technical Digest, 10(3)*, 262-266.

[37] **Simard, D., Steinkraus, P., & Platt, J.** (2003). Best practices for convolutional neural networks. *Proc. 7th Int'l Conf. Document Analysis and Recognition.* Edinburgh.

[38] **Zhang, X.-Y., Bengio, Y., & Liu, C.-L.** (2017). Online and offline handwritten chinese character recognition. *Pattern Recognit, 61*, 348-360.
[39] **LeCun, Y., Bottou, L., Bengioi, Y., & Haffner, P.** (1998). Gradient-based learning applied to document recognition. *Proc. IEEE, 86(11)*, 2278-2324.
[40] **Ling, H., Wu, J., Wu, L., Huang, J., Chen, J., & Li, P.** (2019). Self residual attention network for deep face recognition. *IEEE Access, 7*, 55159-55168.
[41] **Wang, Y., Lei, B., Elazab, A., Tan, E.-L., Wang, W., Huang, F., Wang, T.** (2020). Breast cancer image classification via multi-network features and dual-network orthogonal low-rank learning. *IEEE Access, 8*, 27779-27792.
[42] **Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L.** (2009). ImageNet: A large-scale hierarchical image database. *IEEE Conference on Computer Vision and Pattern Recognition*.
[43] **Krizhevsky, A., Sutskever, I., & Hinton, G.** (2012). ImageNet classification with deep convolutional neural networks. *Proc. Neural Information and Processing Systems*.
[44] **He, K., Zhang, X., Ren, S., & Sun, J.** (2016). Deep residual learning for image recognition. *IEEE Conf. Comput. Vis. and Pattern Recognition*, (s. 770-778). Las Vegas.
[45] **Ng, H.-W., Nguyen, V. D., Vonikakis, V., & Winkler, S.** (2015). Deep Learning for Emotion Recognition on Small Datasets using Transfer Learning. *ACM International Conference on Multimodal Interaction*, (s. 443-449). Seattle.
[46] **Phan, H. T., Kumar, A., Kim, J., & Feng, D.** (2016). Transfer learning of a convolutional neural network for HEp-2 cell image classification. *IEEE 13th International Symposium on Biomedical Imaging (ISBI)*, (s. 1208-1211). Prague.
[47] **Nguyen, L. D., Lin, D., Lin, Z., & Cao, J.** (2018). Deep CNNs for microscopic image classification by exploiting transfer learning and feature concatenation. *IEEE International Symposium on Circuits and Systems (ISCAS)*, (s. 1-5). Florence.
[48] **Li, X., Pang, T., Xiong, B., Liu, W., Liang, P., & Wang, T.** (2017). Convolutional neural networks based transfer learning for diabetic retinopathy fundus image classification. *10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI),*, (pp. 1-11). Shanghai.
[49] **Buchanan, B.** (2006). A (very) brief history of artificial intelligence. *AI Mag., 26*, 53-60.
[50] **Widrow, B., & Lehr, M. A.** (1990). 30 years of adaptive neural networks: perceptron, Madaline and backpropagation. *Proceedings of the IEEE, 79(9)*, 1415-1442.
[51] **W. S. McCulloch, W. P.** (1943). A logical calculus of the ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 115-133.
[52] **Rosenblatt, F.** (1957). *The Perceptron — a perceiving and recognizing automaton.* Cornell Aeronautical Laboratory.
[53] **Widrow, B.** (1960). *Adaptive "Adaline" neuron using chemical "memistors".* Stanford: Stanford Electron. Labs.
[54] **Goodfellow, I., Bengio, Y., & Courville, A.** (2016). *Deep Learning.* Cambridge: The MIT Press.
[55] **Patterson, J., & Gibson, A.** (2017). *Deep Learning: A Practitioner's Approach.* Sebastopol: O'Reilly Media.
[56] **Liu, W., Wen, Y., Yu, Z., & Yang, M.** (2016). Large-Margin Softmax Loss for Convolutional Neural Networks. *International Conference on Machine Learning.* New York.
[57] **Hawkins, D. M.** (2004). The Problem of Overfitting. *Chemical Information and Computer Sciences, 44*, 1-12.

[58] **Shimodaira, H.** (2000). Improving predictive inference under covariate shift by weighting the log-likelihood function. *J. Statistical Planning and Inference, 90*, 227-244.

[59] **Ioffe, S., & Szegedy, C.** (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *Proc. 32nd Int. Conf. Mach. Learn.*, (s. 448–456).

[60] **LeCun, Y., & Bengio, Y.** (1995). Convolutional networks for images speech and time-series. M. A. Arbib içinde, *Handbook of brain theory and neural networks.* Cambridge: MIT Press.

[61] **Kumar, G., & Bhatia, P. K.** (2014). A detailed review of feature extraction in image processing systems. *IEEE Fourth International Conference on Advanced Computing & Communication Technologies.* Rohtak.

[62] **Razavian, A. S., Azizpour, H., Sullivan, J., & Carlsson, S.** (2014). CNN features off-the-shelf: An astounding baseline for recognition. *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. DeepVision Workshop*, (s. 806-813). Columbus.

[63] **Gonzalez, R. C., & Woods, R. E.** (2006). *Digital Image Processing.* Englewood Cliffs: Prentice-Hall.

[64] **Mittal, S.** (2018). A survey of FPGA-based accelerators for convolutional neural networks. *Neural Computer and Applications*, 1-31.

[65] **Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., & Schmidhuber, J.** (2011). Flexible, high performance convolutional neural networks for image classification. *International Joint Conference on Artificial Intelligence*, (s. 1237-1242). Barcelona.

[66] **LeCun, Y. A., Bottou, L., Orr, G. B., & Müller, K.-B.** (1998). Efficient BackProp. G. Montavon, G. B. Orr, & K.-R. Müller içinde, *Neural Networks: Tricks of the Trade* (s. 9-48). Berlin: Springer.

[67] **Glorot, X., & Bengio, Y.** (2010). Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, (s. 249-256). Sardinia.

[68] **Hinton, G., & Nair, V.** (2010). Rectified Linear Units Improve Restricted Boltzmann Machines. *Proceedings of ICML. 27.*, (s. 807-814).

[69] **Liu, L., Quyang, W., Wang, X., Fieguth, P., Chen, J., Liu, X., & Pietikäinen, M.** (2019). Deep Learning for Generic Object Detection: A Survey. *International Journal of Computer Vision, 128*, 261-318.

[70] **Zitnick, C. L., & Dollár, P.** (2014). Edge Boxes: Locating Object Proposals from Edges. *European Conference on Computer Vision* (s. 391-406). Zurich: Springer.

[71]**Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I., & Savarese, S.** (2019). Generalized Intersection Over Union: A Metric and a Loss for Bounding Box Regression. *The IEEE Conference on Computer Vision and Pattern Recognition*, (s. 658-666). Long Beach.

[72] **Mhaskar, H. N., & Poggio, T.** (2016). Deep vs. shallow networks: An approximation theory perspective. *Analysis and Applications, 14(6).*

[73] **Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Rabinovich, A.** (2015). Going deeper with convolutions. *Proc. IEEE Conf. Comput. Vis. Pattern Recognşt.*, 1-9.

[74] **Simonyan, K., & Zisserman, A.** (2015). Very deep convolutional networks for large-scale image recognition. *Proc. Int. Conf. Learn. Representations*.

[75] **Bezdan, T., & Džakula, N. B.** (2019). Convolutional Neural Network Layers and Architectures. *Sinteza 2019 - International Scientific Conference on Information Technology and Data Related Research*, (s. 445-451). Belgrade.

[76] **Srivastava, R. K., Greff, K., & Schmidhuber, J.** (2015). Training very deep networks. *Advances in Neural Information Processing Systems 28 (NIPS 2015).* Montreal.
[77] **He, K., & Sun, J.** (2015). Convolutional Neural Networks at Constrained Time Cost. *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (s. 5353-5360). Boston: IEEE.
[78] **Bengio, Y., Simard, P., & Frasconi, P.** (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks, 5(2),* 157-166.
[79] **Pan, S. J., & Yang, Q.** (2010). A survey on transfer learning. *IEEE Trans. Knowl. Data Eng., 22(10),* 1345-1359.

CURRICULUM VITAE

Name Surname **:** Dorukhan Erdem

Place and Date of Birth **:** Ankara, 01.01.1993

E-Mail **:** dorukhan.erdem@gmail.com

EDUCATION **:**

- **B.Sc.** **:** 2018, Istanbul Technical University, Faculty of Electrical and Electronics Engineering, Control and Automation Engineering

PROFESSIONAL EXPERIENCE:

- 2018-2019 : Software developper at Vircon Group Technologies
- 2020-… : Research asistant at Istanbul Technical University, Control and Automation Engineering Department