الـــجـــمـــهـوريـــة الـــجـزائـــريـــة الديـــمقراطـية الـــشعـــبية
PEOPLE'S DEMOCRATIC REPUBLIC OF ALGERIA
وزارة الـــتـــعـــليـــم الـــعـــالـــي و الـــبـــحـــث الـــعـلـــمي
Ministry of Higher Education and Scientific Research
جـــامعة غـــردايـــة
University of Ghardaia
كـلية الـعـلـوم و الـــتكنولـــوجيا
Faculty of Science and Technology
قـــسم الرياضيـــات و الإعـــلام الآلـــي
Department of Mathematics and Computer Science

THESIS

Presented for the degree of **Masters**

In **Computer Science**

Specialty **Intelligent Systems For Knowledge Extraction**

By **Mohamed Abdallah BEDJEGHA**

Theme

Video Classification Using Deep Learning

Jury Members

M. OULED NAOUI Slimane	MCB	Univ. Ghardaia	President
M. ADJILA Abderrahmane	MAA	Univ. Ghardaia	Examiner
M. MAHDJOUB Youcef	MAA	Univ. Ghardaia	Supervisor

College Year 2019/2020


بسم الله الرحمن الرحيم

All praise is due to Allah, by whose honor and majesty, deeds of virtue are accomplished.

Dedicated To

My beloved mother, the woman who raised me to be the man i am today, thank you for all the love and support you provided ever since i was a child. Thank you for being the best mom a person can ever have.

My precious father, the man who i aspire to be like, one day. Thanks for providing my needs and paving the way for me to study and get a diploma. Thank you for all the guidance and encouragement you provided.

I pray to Allah to grant both of you mercy and paradise as you raised me when i was a child. I pray to Allah to grant you good health and a long blessed life.

My lovely sister, who has been there when ever i needed her. Thank you for being a true blessing from god.

And my dear brother, who's the spine i lean on whenever i need backup and who i am lucky to have by my side.

Cordially,
Mohamed Abdallah.

Acknowledgment

First and foremost, I would like start by expressing my sincere gratitude for my supervisor Mr, **Mahdjoub Youcef** who provided continuous advice and encouragement throughout this experience. Thank you very much for your patience and understanding. It was an honor working with you on this thesis, and i look forward for more opportunities to learn from you.

I also would like to express my appreciation to the members of the jury for taking interest in my humble work.

A huge thank you, to all my teachers who taught me throughout the years i spent in college. Your time and efforts are much appreciated.

Thank you all.

Abstract

Video processing is becoming increasingly important for several applications. A great progress has been made in the field using Machine Learning (ML), especially the Deep Learning (DL) which is the subfield that mimics the human brain by modeling the biological neural networks using a tool called Artificial Neural Networks (ANNs) . The Two most important ANNs are Convolutional Neural Networks (CNN) and Recurrent Neural Networks (RNN). Long Short-Term Memory (LSTM) is one of the most commonly used RNN architecture that have achieved great success in processing sequential multimedia data like video. The aims of this work is to find a combination of the most suitable ANNs for building a model that is able to classify videos into their respective categories. In this perspective, we have experimented two different approaches. The first one uses only a CNN, and the second one uses both CNN and LSTM in order to perform a video classification on three splits of training/test data from the UCF-101 dataset. The experiments confirm that using RNN combined with CNN achieved impressive performance compared to using CNN alone, which has difficulties with videos considering their dynamic structure.

Keywords: Machine Learning (ML), Deep Learning (DL), Video classification, Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), Recurrent Neural Network (RNN), Long Short-Term Memory (LSTM), UCF-101.

Résumé

Le traitement des vidéos prend de plus en plus d'importance dans de nombreuse applications. De grand progrès ont été réalisé dans ce domaine grace a l'Apprentissage Automatique (AA), en particulier l'Apprentissage Profond (AP), qui est le sous-domaine qui imite le cerveau humain en modélisant les réseaux de neurones biologiques à l'aide d'un outil appelé Réseaux de Neurones Artificiels (RNA). Les deux RNA les plus importants sont les Réseaux de Neurones Convolutives (RNCs) et les Réseaux de Neurones Récurrents (RNRs). La Mémoire Court et Long Terme (MCLT) est l'une des architectures RNR la plus fréquemment utilisées grâce à son grand succès dans le traitement de donnée séquentielles comme la vidéo. L'objectif de ce travail est de trouver une combinaison de RNA la plus appropriée pour construire un modèle capable de classer les vidéos dans leurs catégories respectives. Dans cette perspective, nous avons expérimenté deux approches différentes. La premiere utilise uniquement un RNC, et la deuxieme utilise à la fois un RNC et une MCLT afin d'effectuer une classification vidéo sur trois distribustions de données entraînement / test provenant de l'ensemble de données UCF-101. Les experimentations confirment que l'utilisation d'un RNR combinée avec un RNC donnait des résultats impressionnants par rapport à l'utilisation de l'RNC seul, qui a des difficultés avec les vidéos compte tenu de leur structure dynamique.

Mots-clés: Apprentissage Automatique (AA), Apprentissage Profond (AP), Classification des vidéos, Réseau de Neurones Artificiel (RNA), Réseau de Neurones Convolutive (RNC), Réseau de Neurones Récurrent (RNR), Long Short-Term Memory (LSTM), UCF-101.

ملخص

أصبحت معالجة الفيديو ذات أهمية متزايدة للعديد من التطبيقات. تم إحراز تقدم كبير في هذا المجال بإستخدام التعلم الآلي, تحديدا التعلم العميق الذي يعرف كمجال فرعي يحاكي الدماغ البشري من خلال نمذجة الشبكات العصبية البيولوجية باستخدام أداة تسمى الشبكات العصبية الاصطناعية. الشبكتين العصبيتين الاصطناعيتين الأهم هما الشبكات العصبية التلافيفية و الشبكات العصبية المتكررة. الذاكرة طويلة-قصيرة المدى الخاصة بالشبكات العصبية المتكررة هي واحدة من الهندسات الأكثر استخدامًا بفضل النجاح الذي حققته في معالجة بيانات الوسائط المتعددة المتسلسلة مثل الفيديو. يهدف هذا العمل الى العثور على تركيبة الشبكات العصبية الملائمة لبناء نموذج يصنف مقاطع الفيديو الى أصنافها المناسبة. من هذا المنظور، جربنا طريقتين مختلفتين. الأولى تستخدم شبكة عصبية تلافيفية معزولة, و الثانية تعتمد على شبكة عصبية تلافيفية إضافة إلى ذاكرة طويلة-قصيرة المدى, بهدف تصنيف مقاطع الفيديو على ثلاثة توزيعات التدريب / الاختبار من مجموعة بيانات UCF-101 .أكدت التجارب أن دمج الشبكة العصبية التكرارية إضافة إلى شبكة عصبية تلافيفية حقق أداءً مذهلاً مقارنةً باستخدام الشبكة العصبية التلافيفية معزولة, و التي واجهت صعوبات مع مقاطع الفيديو بإعتبار بنيتها الديناميكية.

كلمات مفتاحية: تعلم الآلة, التعلم العميق, تصنيف مقاطع الفيديو, الشبكة العصبية الاصطناعية, الشبكة العصبية التلافيفية, الشبكة العصبية التكرارية, لذاكرة طويلة-قصيرة المدى , UCF-101.

.

Contents

List of Tables

List of Figures

List of Abbreviations

3D-SIFT 3D **S**cale **I**nvariant **F**eature **T**ransforms

AA **A**pprentissage **A**utomatique

Adam **Ada**ptive **M**oment Estimation Algorithm

AI **A**rtificial **I**ntelligence

ANN **A**rtificial **N**eural **N**etwork

AP **A**pprentissage **P**rofond

BoF **B**ag **O**f **F**eature

BPTT **B**ack **P**ropagation **T**hrough **T**ime

CE **C**ross **E**ntropy

CNN **C**onvolutional **N**eural **N**etwork

CReLU **C**oncatenated ReLU

CV **C**omputer **V**ision

DBN **D**eep **B**elief **N**etwork

DL **D**eep **L**earning

DSD **D**ense-**S**parse-**D**ense

ELU **E**xponential **L**inear **U**nit

FCN **F**ully **C**onvolutional **N**etworks

FFNN **F**eed-**F**orward **N**eural **N**etworks

FLOP **Fl**oat **Op**erations

GB **G**radient **D**ecent

GPU **G**raphical **P**rocessing **U**nits

GRU **G**ated **R**ecurrent **U**nit

HoG **H**istograms of **o**riented **G**radients

HOG3D **3D H**istograms Of **O**ptical **G**radients

IDT **I**mproved **D**ense **T**rajectories

ILSVRC **L**arge **S**cale **V**isual **R**ecognition **C**ompetition

K-SOM **K**ohonen **S**elf **O**rganizing **M**ap

LR **L**earning **R**ate

LSTM **L**ong **S**hort **T**erm **M**emory

MBH **M**otion **B**oundary **H**istograms

MCLT **M**émoire **C**ourt et **L**ong **T**erme

ML **M**achine **L**earning

MPA **M**ulti-scale **P**yramid **A**ttention

MRI **M**agnetic **R**esonance **I**maging

MS COCO **M**icro**s**oft **C**ommon **O**bjects in **Co**ntext

MSE **M**ean **S**quared **E**rror

MSLE **M**ean **S**quared **L**ogarithmic **E**rror

R-CNN **R**egion based **C**onvolutional **N**ueral **N**etwork

R-FCN **R**egion-Based **F**ully **C**onvolutional **N**etworks

RBF **R**adial **B**asis **F**unction

RBFNN **R**adial **B**asis **F**unction **N**eural **N**etworks

RBM **R**estricted **B**oltzman **M**achine

ReLU **Re**ctified **L**inear **U**nit

RGB **R**ed, **G**reen and **B**lue

RL **R**einforcement **L**earning

RNA **R**éseau de **N**eurones **A**rtificiel

RNC **R**éseau de **N**eurones **C**onvolutive

RNN **R**ecurrent **N**eural **N**etwork

RNR **R**éseau de **N**eurones **R**écurrent

SGD **S**tochastic **G**radient **D**ecent

SIFT **S**cale-**I**nvariant **F**eature **T**ransform

SSD **S**ingle **S**hot multibox **D**etector

TPU **T**ensor **P**rocessing **U**nit

TT-RNN **T**ensor-**T**rain **R**ecurrent **N**eural **N**etwork

TTD **T**ensor-**T**rain **D**ecomposition

VLAD **V**ectors Of **L**ocally **A**ggregated **D**escriptors

YOLO **Y**ou **O**nly **L**ook **O**nce

Introduction

Introduction

In recent years, the advancement in technology promoted the rapid growth of data volume, the variety and size of this data requires efficient processing using new algorithms. Modern Artificial Intelligence (AI), more specifically Machine Learning (ML) is able to utilizes this data along with the available computational power for good, solving very complex problems, and automating tasks which are relevant to us daily.

AI researchers also included the human brain in theirs studies after being inspired by it and that gave birth to a sub field in ML called Deep Learning (DL) which had a huge contribution in the success achieved by ML in many fields. For instance, Computer Vision (CV) which is the field that studies and improves the ability of computers to visualize and understand the content of images and videos and harness the knowledge contained in them.

One of the problems treated by CV is video classification. It is a supervised learning problem where a set of videos is classified as one of a number of predefined classes based on the features learned from a training set.

Achieving efficient video classification opens a lot of doors to several applications, including video surveillance systems, robotics, data organization, etc. That is why it is considered a huge milestone in the road of building intelligent systems that make use of the massive volumes of video data available nowadays. Many researchers worked hard on this problem either by classifying the frames of a video separately disregarding their relation, or by considering that relation and using it as a booster for classification.

In this document, we investigate and study the most convenient DL tools for video classification. Those being Convolutional Neural Networks (CNN), and Recurrent Neural Networks (RNN). We explore two different approaches for building a video classification model on the UCF-101 dataset and compare and discuss their performances.

Our document is structured as follows:

- **In the first chapter** we talk about AI, ML and DL and their relationship. In addition, we go through DL in more details outlining its origins, inspirations, challenges, the resources it utilizes as well as a quick rundown of the mathematics behind it.

- **In the second chapter**, we first talk about Computer Vision (CV) and highlight the contribution of DL to it. Moreover, we mention some of the tasks involving videos as data for training in addition to the video data availability nowadays. Next, we define video classification and explain the different challenges encountered with it. Finally we clarify the different approaches made by the AI community to solve it.

- **In the third and final chapter**, we present the implementation of two different models for video classification including a brief explanation of the software, hardware and the dataset used for the experiment (UCF-101) and finally, a discussion of the results reported by each model.

Chapter 1

Artificial Intelligence, Machine Learning and Deep Learning

1.1 Introduction

Nowadays, AI became among the most interesting fields of computer science, due to its efficiency in solving very complex problems and automating day to day tasks that were once considered hard, rather impossible for a machine to comprehend. A lot of the credit for that goes to ML and DL.
In this chapter, we will present definitions and concepts about ML and its divisions. Moreover, we will explain DL and some of its tools and techniques in a more profound manner.

1.2 Artificial Intelligence

AI is a trending branch in computer science. It is defined by Bellman as:
"The automation of activities that we associate with human thinking, activities such as decision-making, problem solving, learning"[8].
AI's significance is growing in parallel with the data explosion caused by the internet, and the rise of powerful computers. In the meantime, the enormous shift from Expert Systems to ML exploited massive amounts of data by putting the available powerful computers to good use aiming to get a step closer to real AI.

Since Alan Turing proposed the Turing Test[9] concept, which was a huge milestone for AI. The field has evolved remarkably thanks to ML.

1.3 Machine Learning

ML is a sub-field of AI, which uses several techniques to enable computers to learn from data and predict results. The term machine learning was popularized by Arthur Samuel in 1959 as:
"... the field of study that gives computers the ability to learn without being explicitly programmed"[10].
Moreover, a widely quoted definition was proposed by Tom Mitchell who stated that:
"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E"[11].

ML introduced a whole new paradigm of programming for solving problems. Unlike classical programming where we feed the program rules and data expecting answers (e.g., the sum of two integers), ML algorithms receive data and answers as labeled data[1], and they figure out the rules using statistical methods based on the data (Figure 1.1). Additionally, the algorithm is tested to judge its performance using the answers provided earlier.

Along with the rise of Big Data[2]. Humans need a tool to extract and take advantage of

[1]There are cases where the model does not receive answers with the data. Because the goal is to find out how the data is structured so there is no exact answers to be defined. This will be discussed in subsection 1.3.1.

[2]Refer to this article for a good and brief explanation of Big Data: https://medium.com/swlh/big-data-explained-38656c70d15d.

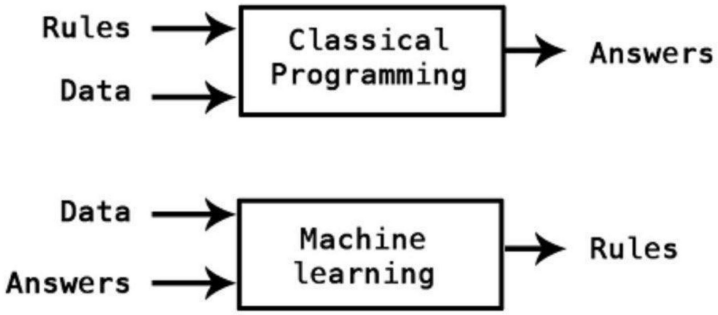


Figure 1.1: The difference between ML and classical programming.

the knowledge within that data, and this is where the importance of ML manifests. In fact, ML is continuously being used to create smart systems that function with minimum human intervention.

1.3.1 Types Of Machine Learning

Supervised Learning In supervised learning, an algorithm outputs a predicted label for an input, based on the rules extracted from labeled training data. Theoretically, it is approximating a target function that maps some inputs to an output label[3]. For example, the target function could map the features of a house to a price, or an Magnetic Resonance Imaging (MRI) scan to one of two classes: malignant tumor/begnin tumor. The algorithm starts by some random function and tries to mimic the behavior of the target function iteratively, by comparing the predictions to the real corresponding outputs and adjusting the function at hand consequently.

Unsupervised Learning Unsupervised learning consists of deducing a function to represent a hidden structure from unlabeled training data. We only have a collection of input examples and the model tries to find any correlations between instances and detect outliers, aiming to cluster the data set instances in groups which behave similarly.

Reinforcement Learning In Reinforcement Learning (RL), the learning system, which is a software-defined agent, interacts with a dynamic environment in which it must attain a certain goal. There is no direct access to the correct output. However, we can get some measure of the quality of an output following a certain input. The program is provided with rewards and punishments as it navigates its problem space. RL algorithms were able to create models with astounding performance such as Deepmind's AlphaGo[4], which is an algorithm that beat the world champion of the Go board game. Other RL models excelled at several other more complex games[12, 13].

Figure 1.2 shows some algorithms included in each type of machine learning.

[3] If the label is a continuous value we call it regression, but if it belongs to a finite number of values (For example 0 or 1) its called classification.

[4] https://deepmind.com/blog/article/alphago-zero-starting-scratch

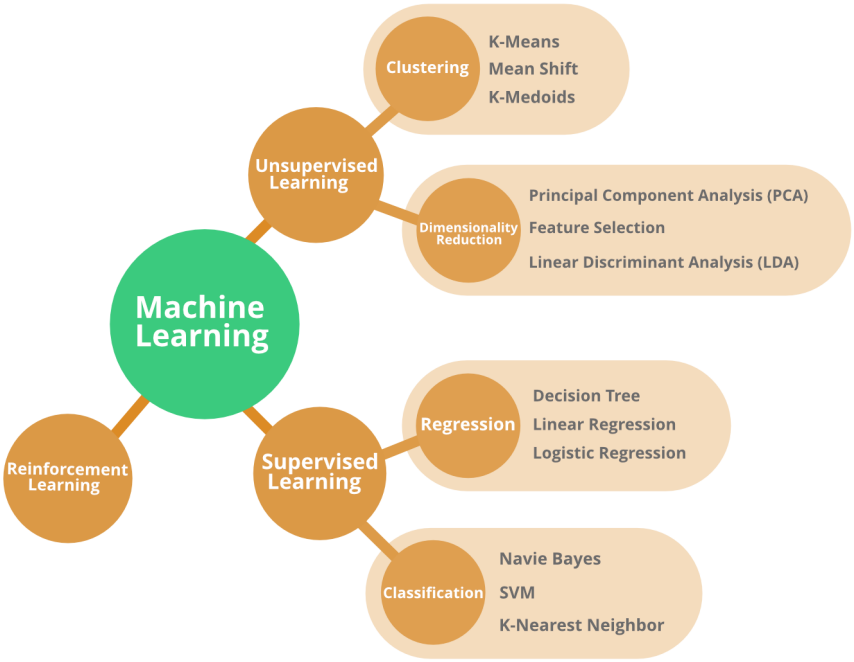


Figure 1.2: Some of the algorithms for each type of ML

1.4 Deep Learning

DL is the sub field of ML that caused a huge improvement in resolving tasks such as speech recognition[14], image recognition[15], topic classification, sentiment analysis, question answering[16] and language translation[17, 18].

DL uses representation-learning methods that exploit multiple levels of data representations. For classification tasks, deeper layers of data representation emphasis aspects of the input that are important for discriminating specific classes, and also suppresses irrelevant aspects that have little effect on the classification. For example, in image classification we start by simple two dimensional arrays. The features learned by the early layers of representation could be simple edges and arrangements of these edges. Deeper layers may gather fragments of familiar objects, and so on. Therefore, the deeper we go, the more features we learn. One of the advantages of DL is that features that define an object or a class are not calculated by humans. Instead, they are learned from data using a general purpose learning procedure.

1.5 Difference between Machine Learning and Deep Learning

Both ML and DL produce meaningful representations starting with the input data. A data representation is a way to interpret or encode the data. For example, color schemes are used to encode images.

On one hand, ML algorithms tend to extract appropriate representations to only complete the task at hand reliably. For example, a ML algorithm might explore only one or two layers of data representation to reach ideal results and that is all. On the other hand, DL is considered a sub field of ML, and it emphasizes on extracting successive layers of meaningful representations. Modern DL can reach tens or even hundreds of successive layers of representations using tools called artificial neural networks which are inspired by the structure of the human brain.

1.6 Biological Neural Network

Biological Neural Networks are the reason why we get to experience the world around us. They are composed of billions of connected biological neurons. Very complex computations can be performed by combining the simple computations made by each neuron.

1.6.1 Biological Neurons

They are cells found in the cerebral cortex. As shown in figure 1.3, dendrites receive electrical signals as weighted inputs. The inputs are used to compute an output signal by the cell body. Once the output signal reaches a specific value, it passes through the axon wire which is connected to other neurons using the synaptic terminal.

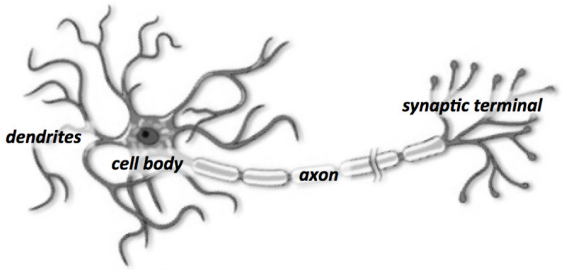


Figure 1.3: The structure of a Biological Neuron

1.6.2 Artificial Neurons

The artificial neuron is the model for the biological neuron. It takes a set of inputs, it multiplies each input by a weight, computes the sum of all these weighted inputs plus a constant value called a Bias. The weighted sum is then fed to a function called an Activation Function, which forwards its output to a another neuron as input. Figure 1.4 illustrates the structure of an artificial neuron.

1.7 Artificial Neural Network

The first ANN was called the Perceptron[19]. It was developed by Rosenblatt Frank in 1957 who was inspired by Warren McCulloch and Walter Pitts[20]. Perceptrons used

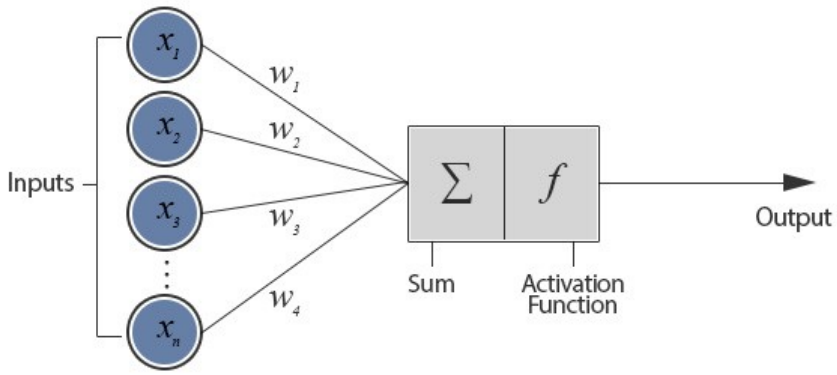


Figure 1.4: The structure of an artificial neuron

an artificial neuron called a linear threshold unit (LTU), in which the activation function returns a value only if the weighted sum of the inputs exceeds a certain threshold. A single perceptron was useful for simple linear binary classification but incapable of learning to separate non linear data (XOR logical function). Later research showed that multi-layered perceptrons were able to implement such functions and even more complex ones. Creating what is now known as Deep Neural Networks.
In the next section, we will present some of the mostly used activation functions in DL.

1.8 Activation Functions

Different neurons require different activation functions in order to perform their task the best. In the next subsections, we will discuss some of the most common activation functions in DL and their characteristics and some of the tasks they are best suited for.

1.8.1 Rectified Linear Unit

The Rectified Linear Unit (ReLU) is defined by the equation 1.1:

$$y = f(z) = \max(0, z) \tag{1.1}$$

Where the function outputs 0 if the input is negative, and it outputs the input its self if it was positive. Nowadays, ReLU is widely used, especially in CNNs. Some of its advantages is eliminating the vanishing gradients problems, not to mention its cheap computational cost and low training time[5]. Figure 1.5 shows the plot for the ReLU function.

[5]There exists some variations for ReLU such as the Exponential Linear Unit (ELU)[21] , Concatenated ReLU (CReLU)[22] , etc.

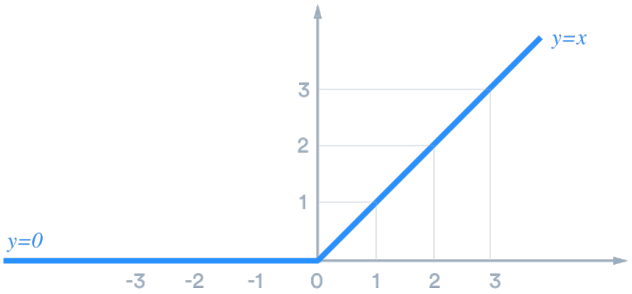


Figure 1.5: The ReLU function plot

1.8.2 Sigmoid

Sigmoid, also known as the Logistic Activation Function. It is often used in models that output a probability, which is a value between 0 and 1. For example a cat/dog classifier, where we have to classify an image as being either a cat (class 1) or a dog (class 0). So as an answer, our model outputs a single value, which is the probability that the image is classified as a cat. Keeping the ability to compute the other probability (of the image being a dog) which is equal to $1 - \hat{y}$, where $\hat{y}$ is the prediction.
The Sigmoid function is defined mathematically by equation 1.2 and figure 1.6 represents its plot:

$$y = f(z) = \frac{1}{1 + e^{-z}} \tag{1.2}$$

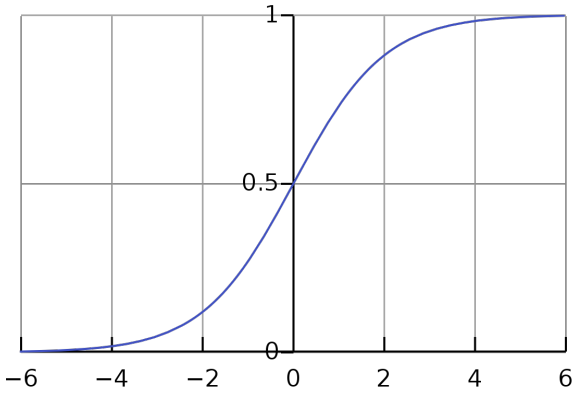


Figure 1.6: The Sigmoid function plot

One of its benefits is that its is non linear, thus it can be used to classify non linear data properly. However, In areas where z is very large or very low, the y value barley reacts to changes of z, this causes the gradient to be very small, leading to the vanishing gradient problem which makes training slower and more complicated. Despite that, Sigmoid is still very popular in classification problems.

1.8.3 Tanh

Tanh is a a nonlinear function similar to Sigmoid. The difference between them is that the Tanh function ranges between -1 and 1, unlike Sigmoid who's values range between 0 and 1. One of the pros tanh has over sigmoid is that its gradients are stronger and the derivatives are steeper and that makes it less prone to cause the vanishing gradients problem. Therefore, it is widely used in RNNs nowadays.
Tanh is defined mathematically by equation 1.3 and figure 1.7 represents its plot:

$$y = f(z) = \frac{2}{1 + e^{-2z}} - 1 \tag{1.3}$$

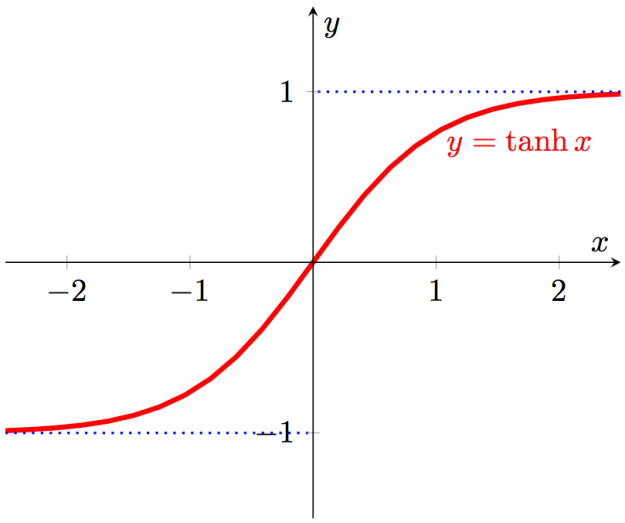


Figure 1.7: The Tanh function plot

1.8.4 Softmax

This function is widely used in the output layer in ANNs that perform multi class classification. It takes a vector of activations and outputs a vector that represents the probability distribution of all possible input classifications. The Softmax function is defined by equation 1.4:

$$y = f(x_i) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{1.4}$$

This function basically takes the input vector x and calculates a sum of the exponentials of its components $\sum_j e^{x_j}$. Then, to compute the first value of the output vector x_1 it just divides the exponential of x_1 by the first sum, and so on(equation 1.4). The final output is a vector of normalized probabilities that all add up to 1. This vector represents a trivial tool to determine the highly favorable class according to the model, as a trained model will output a vector in which the target class has the highest probability.
In the next section we will discuss some of the famous neural network types, starting with Feed-Forward Networks.

1.9 Types of Artificial Neural Networks

1.9.1 Feed-Forward Neural Network

Regardless of its structure, a single neuron cannot classify images or differentiate handwritten digits, and that is why billions of neurons in the brain are stacked in layers. Feed-Forward Neural Networks (FFNN) are constructed by connecting two or more layers of neurons, in a way that every neuron's inputs are the outputs of neurons from the previous layer (Figure 1.8). The first layer in the network is called an input layer, and it contains basic neurons that return whatever input they are fed. The last layer is the output layer, and it outputs the prediction of the network, whereas layers in the middle are hidden layers and their number represents the network's depth. The weights of the inputs for each neuron in a layer form a matrix called the weights matrix.

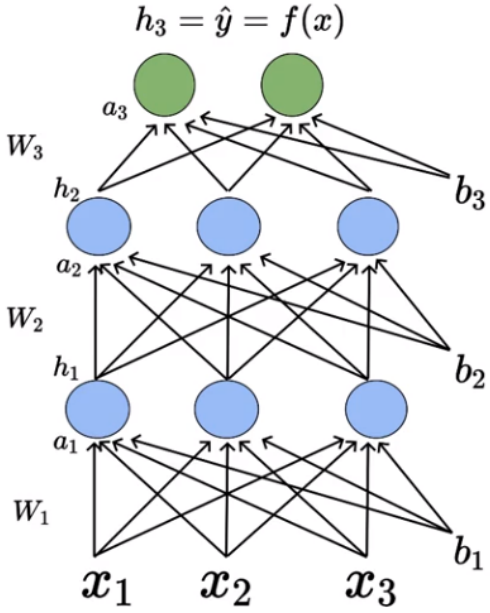


Figure 1.8: A FFNN with four layers: An input layer, an output layer and two hidden layers. Notice how the inputs of each neuron are the outputs of the neurons from the previous layer plus a bias unit.

These networks are called feed-forward because there is no connections between neurons in the same layer. As a result, the information flows one way only.[23, 24].

1.9.2 Radial Basis Function Neural Network

Radial Basis Function Neural Networks (RBFNN) are 3 layered FFNNs. They are limited to having only one hidden layer plus an input and an output layer. The hidden layer contains neurons that use an activation function called a Radial Basis Function (RBF). RBFNNs map non linearly separable data, to feature vectors in another space, causing the projected data to be linearly separable. Only then, feature vectors are fed to the output layer for classification.

The RBF neurons compute the distance between an input vector and a set of stored vectors. For example: In the RBFNN presented in figure 1.9, if the distance between an input vector $\vec{x}$ and $\vec{\mu_1}$ is big, the output of the upper neuron will be close to 0. Reversely, if the distance is small, the output of the neuron will be close to 1. The output neurons are linear and they serve as summation units[25]. Here are some of the most famous RBF functions:

- The multi-quadratic RBF: $h(x) = \sqrt{(d^2 + c^2)}$
- The thin plate spline function: $h(x) = d^2 ln(d)$
- The popular Gaussian RBF: $h(x) = exp(\frac{-d^2}{2})$

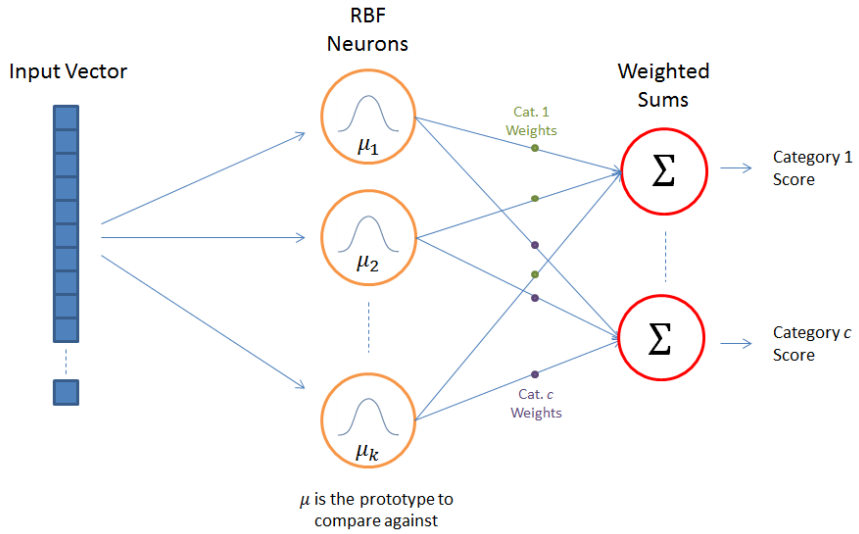


Figure 1.9: A Radial Basis Function Network.

1.9.3 Kohonen Self Organizing Map

The Kohonen Self Organizing Map (K-SOM)[26] is a special type of ANN. It is trained on unlabeled data for clustering purposes, and also used for Dimensionality Reduction. It consists of an input layer connected to a grid of neurons called a Kohonen layer (Figure 1.10). In training, the map starts by initializing the weights connecting the Kohonen layer with the input neurons. A random input vector is selected from the data to trigger a competition between the output nodes. Only the node whose weight vector is most similar to the input vector will be activated and declared as the winner. The similarity is measured with a certain distance (Manhattan, Euclidean, etc.). Next, The weight vectors of the winner and its neighbors gets updated while considering a learning rate and a neighborhood size. Once the weights are updated, a new data instance is chosen and the cycle repeats[27].

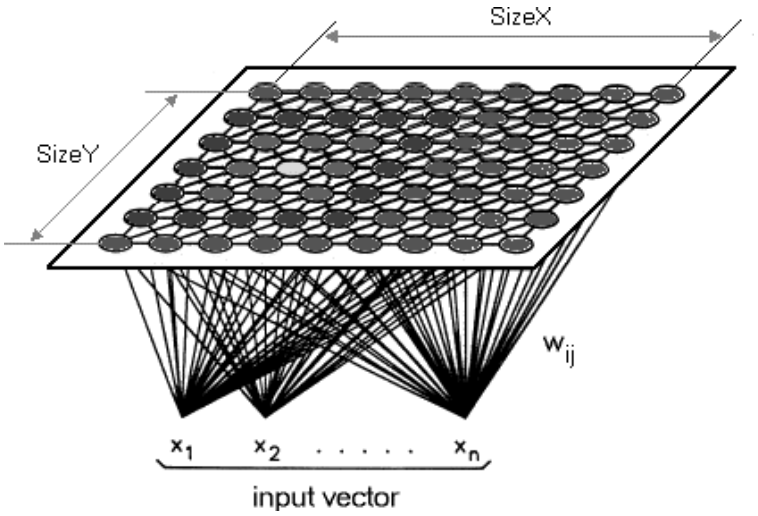


Figure 1.10: A Kohonen Self Organizing Neural Network architecture.

1.9.4 Restricted Boltzman Machine

The Restricted Boltzman Machine (RBM) was originally invented by Geoffrey Hinton[28]. RBMs are ANNs with only two fully connected layers[6] (A visible layer and a hidden layer). RBMs can learn to reconstruct data in an unsupervised fashion, throughout several forward and backward passes between the visible layer and the hidden layer. In the forward pass, the visible layer receives a vector of inputs. Each node in the hidden layer receives a weighted sum of the input values then adds a specific bias to them, then passes the results through a sigmoid activation function. In the backward pass, the same procedure is repeated in the opposite direction, using the same weights and a new bias for every weighted sum.
The weights of an RBM are initialized randomly, which causes the reconstructed data and the original input to be different[7]. During training, this difference is backpropagated with respect to weights iteratively until an error minimum is reached.
Figure 1.11 illustrates the training process of an RBM .

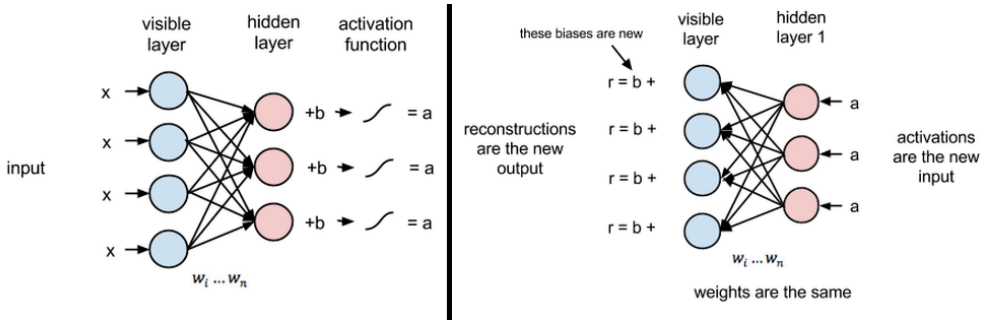


Figure 1.11: An illustration of the forward pass (left) and backward pass (right) in training an RBM.

[6]The two layers of an RBM are fully connected. However the connections between neurons in the same layer are restricted. This restriction makes an RBM a special class of Boltzman machines[29]

[7]This different is referred to as the **reconstruction error**

1.9.5 Deep Belief Network

Deep Belief Networks (DBNs) emerged when Geoffery Hinton proved that RBMs can be stacked and trained in a greedy manner[30]. Accordingly, DBNs can be defined as graphical models with the ability to learn to extract deep hierarchical representations of the training data. Technically, in a DBN, the hidden layer of each stacked RBM is the visible layer of the next hidden layer. Thus, the training of a DBN consists of:

1. Training the first layer as an RBM that models a raw input vector as its visible layer.
2. Then, using that first layer to obtain a representation of the input.
3. Next, using the obtained representation as training examples for the next layer, considering the first hidden layer a visible layer in the next RBM.
4. Iterating steps 1, 2 and 3 for the desired number of layers.

Figure 1.12 illustrates the structure of an DBN.

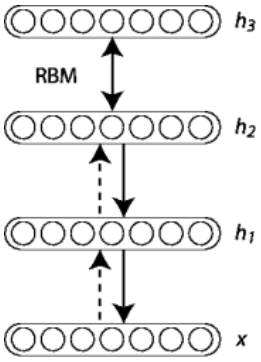


Figure 1.12: The architecture of an DBN

1.9.6 Convolutional Neural Networks

CNNs are FFNNs adapted to data in the form of a grid. CNNs were first introduced in the 1990s by Yann LeCun et al.[31], after being inspired by the studies of the brain's visuel cortex by Dr. Kunihiko Fukushima in the 1980s[32][8], and they have been widely used since then.

A CNN uses a mathematical operation called convolution at least once in its early layers, along with another operation called pooling. In other words, CNNs introduce convolutional and pooling layers in the early stages of a FFNN. In order to explain convolutional and pooling layers we must first talk about the convolution operation mathematically.

[8]Dr. Fukushima himself followed the work of David Hubel and Torsten Wiesel who performed a series of experiments on animals [33, 34], giving crucial insights on the structure of the visual cortex

Convolution Convolution is a mathematical operation denoted by (*), and it is defined as follows: The convolution of two signals x and w with respect to time t is a signal calculated by equation 1.5:

$$s(t) = (x * w)(t) = \int x(a)w(t-a)da \tag{1.5}$$

Realistically, computers cannot process time in a continuous from, therefore, it is discretized. Thus, assuming that time t can then take on only integer values, we can define the discrete convolution using equation 1.6:

$$s(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{1.6}$$

In equation 1.6, the signal x is the input, the signal w is called the convolution kernel whereas the output $s(t)$ is referred to as the feature map.
In CNNs, the input and the kernel are usually multi-dimensional arrays containing a finite number of integer values. That is why we usually assume that their signals (x and w) are zero everywhere but the finite set of points for which we store the values. As a result, we can transform the infinite summation into a finite summation. Finally, we use convolutions over more than one axis at a time depending on the dimensions of the input. For example, a two-dimensional input image I, is convolved using a two-dimensional kernel K according to equation 1.7:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{1.7}$$

Because convolution is commutative, we can equivalently write equation 1.7 as:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \tag{1.8}$$

However, many DL libraries don't use this function precisely, instead they use an operation called **cross-correlation**, which is considered the same as convolution but it uses a slightly edited function defined by equation 1.9:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \tag{1.9}$$

Many libraries implement cross-correlation but call it convolution[23].

Convolutional Layer Convolutional layers are the most important building block for CNNs. Unlike fully connected layers, a neuron in a convolutional layer is not connected to every neuron from the previous layer. Instead, it is connected to only a group of neurons called the receptive field (Figure 1.13).

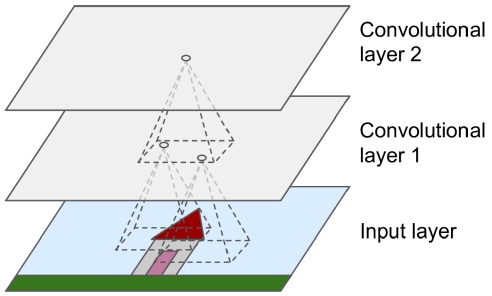


Figure 1.13: An iluustration of how convolutional layers are connected: Each neuron in a convolutional layer is connected to a rectangular small receptive field from the previous layer.

This structure allows the network to focus on low level features, and combine them into high-level features going deeper in the network.

The use of fully connected layers for image recognition tasks will increase the number of learnable parameters exponentially. For example, a 100 by 100 pixel image is flattened to become an input layer with 10,000 neurons. Supposing that the first layer has just 1000 neurons, this implies 10 million connections for just the first layer. CNNs solve this problem by following the sparse interactions principle illustrated in figure 1.14.

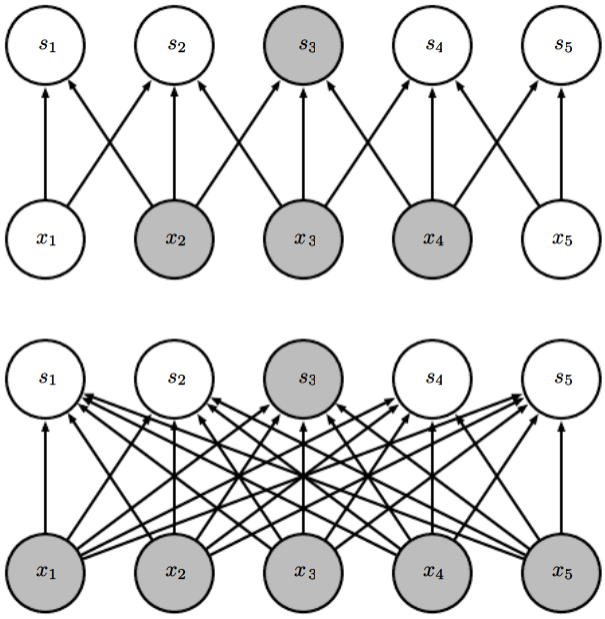


Figure 1.14: The difference between the presence of the sparse interaction principle (top) and its absence (bottom) in a CNN: (Top) s_3 is formed by convolution with a kernel of width 3. (Bottom) s_3 is formed by matrix multiplication.

The second principle that CNNs follow is parameter sharing, which refers to different locations sharing the same weights, rather then learning different sets of weights for each location of the input array. In other words, the weights connecting a receptive

field to a neuron, are the same as the weights connecting another receptive field to another neuron in some other area of the array[23, 24]. Figure 1.15 illustrates the difference between the presence of the parameter sharing principle and its absence.

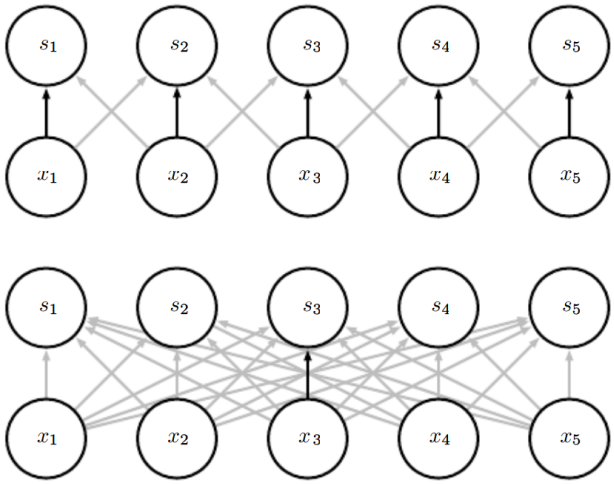


Figure 1.15: The difference between the presence of the parameter sharing principle (top) and its absence (bottom) in a CNN: The black arrows in the top figure indicate the shared uses of the central value of the kernel in a CNN. Whereas the single black arrow in the bottom figure indicates the use of the central element of the weight matrix in a FFNN.

Finally, the last principle that CNNs employ is pooling. It will be explained later. First, we will explain some additional concepts which are substantial to convolution:

1. **Filter**: A filter or a convolution kernel is formed by the weights connecting a receptive field to a neuron. Filters can also be seen as small arrays the size of the receptive field[9]. The choice of these weights controls what kind of features to extract from an input image. The reason is that convolution on an image produces a feature map which enhances the areas in the image that are most similar to the filter used. However, during training, a CNN finds the most useful filters automatically, and combines them to get deeper relevant representations of the input layer in order to do the job[23, 24]. Figure 1.16 shows the feature map (right) resulting from convolution using a filter that looks for horizontal edges. The filter emphasizes on areas where there is a sudden change of pixel intensity.

[9]Generally, filters of small sizes (3 by 3 or 5 by 5) are used. Less commonly, larger sizes are used (7 by 7) but only in the first convolutional layer. Small filters achieve high representational power while also keeping the number of parameters to learn small.

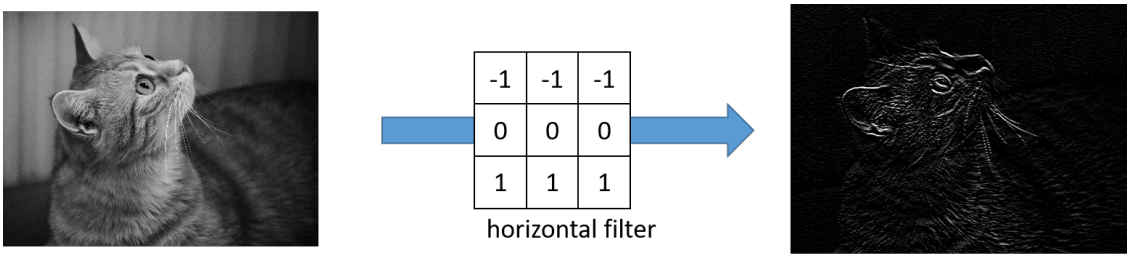


Figure 1.16: The effect of a convolution using a filter that enhances horizontal edges

2. **Valid Convolution**: Using a filter of an odd size causes the feature map to be smaller then the input image. This particular kind of convolution is called valid convolution.
3. **Same Convolution**: In order for a feature map to have the same height and width as the input layer, it is common to add zeros around the input image. It is called zero padding and the operation becomes same convolution[24].
4. **Strides**: In order to make the connections between two layers more sparse, we could use larger strides. A stride is the distance between two receptive fields of two consecutive neurons in a convolutional layer. Hence, it is possible to connect a large input layer to a much smaller layer by using a larger stride, which means spacing out the receptive fields[24].
   If we convolve an N by N image padded with P pixels using an F by F filter and with a stride of S the resulting feature map will be of height $\left\lfloor \frac{N+2P-F}{S} \right\rfloor + 1$ and of width $\left\lfloor \frac{N+2P-F}{S} \right\rfloor + 1$. Figure 1.17 illustrates the difference between a stride of 1 and a stride of 2.

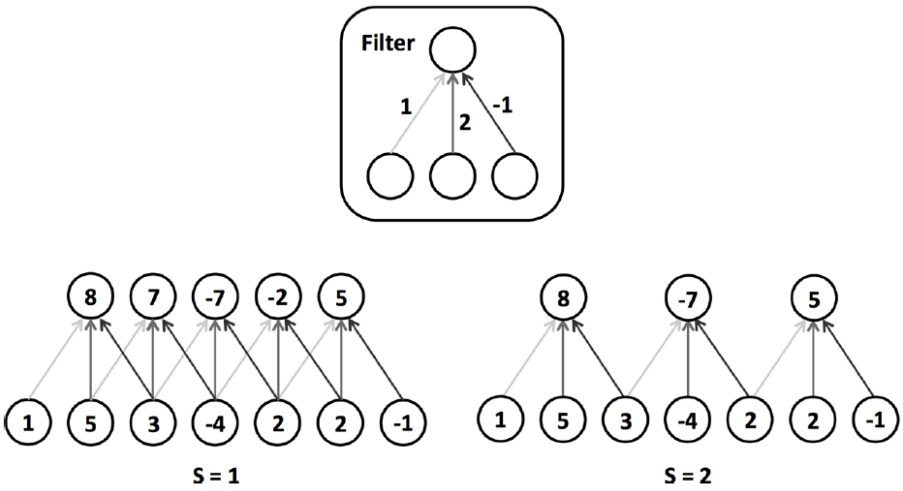


Figure 1.17: An illustration of the sparsity of two strides: Connections between two layers using a stride of 1 (Left), and using a stride of 2 (Right).

Pooling Layer CNNs employ another type of layers called pooling layers. They perform a pooling operation which reduces the size of a feature map given as input, aiming to lower the computational load, memory usage and the number of parameters to learn. Moreover, pooling makes the detected features more robust as it returns a condensed version of the input, plus it introduces the location invariance concept to the network, which means that the network becomes tolerant to small shifts in pixel values, because the values of the pooled output do not change if the input values shifted a little.

Unlike convolutional layers, pooling layers do not have parameters to learn because they apply an aggregation function to the outputs of a small set of units from the previous layer without weighing them. Therefore, pooling summaries the outputs over a small neighborhood which leads to a compact representation with robust features[24]. Different types of pooling layers exist, each one uses a different pooling function:

Max Pooling Returns the maximum within a rectangular neighborhood.

Average Pooling Returns the average of a rectangular neighborhood.

L^2 **Norm Pooling** Returns the L^2 norm of a rectangular neighborhood.

Weighted Average Pooling Returns the weighted average based on the distance from the central pixel.

Figure 1.18 shows the effect of a max pooling layer. It is the most common type of pooling layers. This example uses a pooling kernel of size 2 by 2, a stride of 2, and no padding. Note that only the max input value in each kernel makes it to the next layer. The other inputs are dropped.

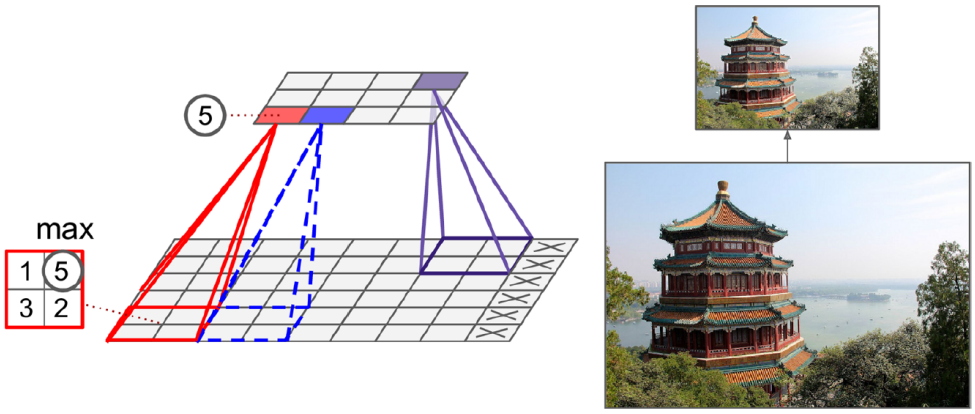


Figure 1.18: The effect of a max pooling layer on an image

Convolutions Over Volumes Nowadays, images are represented using color schemes. The most commonly used one is the Red, Green and Blue (RGB) scheme. Consequently, CNNs use 3D filters to convolve RGB input images and produce one layered feature maps. The filters must have the same depth as the input image, so that the red channel's receptive fields are weighed using the first layer of the filter, the green

channel's receptive fields are weighed using the second layer of the filter and so on. A neuron sums up of all three weighted receptive fields from the previous layer, and does some further calculations. In order diversify features, CNNs use several filters in order to produce an output volume that consists of slices as shown in figure 1.19. The depth of the output volume is equivalent to the number of filters used, and that volume can be the input to another convolutional layer for extracting even higher-level features [35]. In the next subsection, we will explain how did the previously mentioned components were gathered to form famous stat of the art CNN architectures.

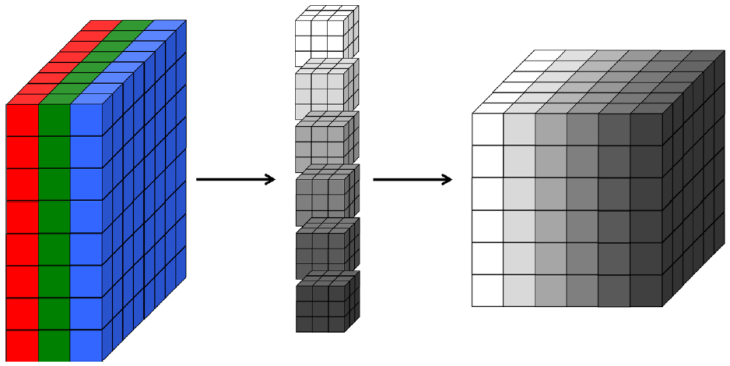


Figure 1.19: A 3D visualization of the convolution of an RGB image with 6 different filters

Convolutional Neural Network Architectures

1. **LeNet-5**: The LeNet-5 is the first CNN architecture designed. It was created by Yann LeCun in 1998 for handwritten digit recognition and it was trained on the MNIST dataset[31]. It contains a total of 7 layers. Three convolutional layers, two pooling layers followed by two fully connected layers.

   The pooling operation used originally in the paper had the four inputs to a unit in the pooling layers, summed then multiplied by a trainable coefficient and added to a trainable bias. Therefore, pooling layers had parameter to learn. The output layer consisted of 10 neurons with a radial basis activation function.
   Figure 1.20 illustrate the architecture of the LeNet-5 network, and table 1.1 summarizes further details about architecture.

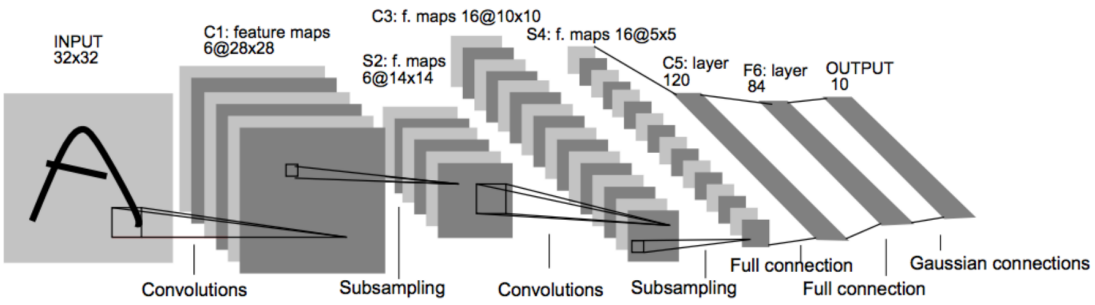


Figure 1.20: The architecture of the LeNet-5 CNN

Table 1.1: Summary of the LeNet-5 architecture

Layer	Type	Output Depth	Size	Kernel Size	Stride	Activation	Number Of Parameters
In	Image	1	32x32	-	-	-	
C1	Convolution	6	28x28	5x5	1	tanh	156
S2	Average Pooling	6	14x14	2x2	2	tanh	12
C3	Convolution	16	10x10	5x5	1	tanh	1,516
S4	Average Pooling	16	5x5	2x2	2	tanh	32
C5	Convolution	120	1x1	5x5	1	tanh	48,120
F6	Fully Connected	-	84	-	-	tanh	10,164
Out	Fully Connected	-	10	-	-	RBF	-

2. **AlexNet**: The AlexNet architecture was developed in 2012, by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton[15] as a contribution in the ImageNet Large Scale Visual Recognition Competition (ILSVRC) challenge that year. This model achieved state of the art classification accuracy when compared to all the existing machine learning approaches[10]. Figure 1.21 and table 1.2 illustrate the AlexNet architecture and its layers. To reduce overfitting, the authors utilized regularization techniques, such as dropouts and data augmentation. They also used a technique called Local Response Normalization (LRN) in order to improve generalization.

   A modified version of AlexNet called ZFNet[36] was developed by Matthew Zeiler and Rob Fergus. It is the same as AlexNet with a few altered hyperparameters (number of feature maps, kernel size, stride, etc.). ZFNet was not strictly the winner of ILSVLC 2013. Instead, Clarifai which had only small improvement over ZFNet, was the winner[11].

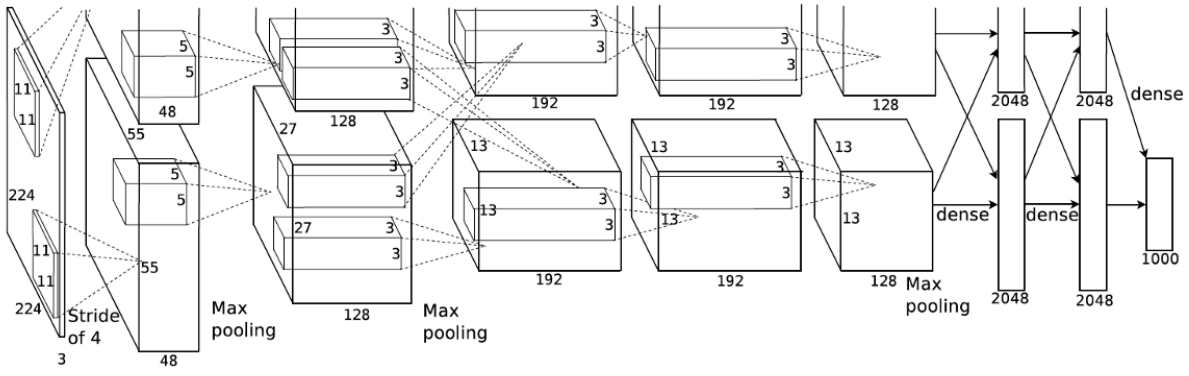


Figure 1.21: The architecture of the AlexNet CNN

[10] http://image-net.org/challenges/LSVRC/2012/results.html
[11] http://www.image-net.org/challenges/LSVRC/2013/results.php

Table 1.2: Summary of the AlexNet network architecture.

Layer	Type	Output Depth	Output Size	Kernel Size	Stride	Padding	Activation	Number Of Parameters
In	Input	3	224x224	-	-	-	-	0
C1	Convolution	96	55x55	11x11	4	Same	RELU	34,944
S2	Max Pooling	96	27x27	3x3	2	Valid	-	0
C3	Convolution	256	27x27	5x5	1	Same	RELU	614,656
S4	Max Pooling	256	13x13	3x3	2	Valid	-	0
C5	Convolution	384	13x13	3x3	1	Same	RELU	885,120
C6	Convolution	384	13x13	3x3	1	Same	RELU	1,327,488
C7	Convolution	256	13x13	3x3	1	Same	RELU	884,992
S8	Max Pooling	256	6x6	3x3	2	Valid	RELU	0
F9	Fully Connected	-	4,096	-	-	-	RELU	37,752,832
F10	Fully Connected	-	4,096	-	-	-	RELU	16,781,312
Out	Fully Connected	-	1,000	-	-	-	SOFTMAX	4,097,000

3. **VGG-16**: In 2014, Karen Simonyan and Andrew Zisserman introduced a deeper network called VGG-16[37]. It improved the AlexNet architecture by replacing the large kernel sized filters (in layers C1 and C3) with multiple consecutive 3x3 kernel-sized filters. VGG16 was trained for weeks using NVIDIA Titan Black GPU's. The VGG-16 architecture is depicted in figure 1.22.

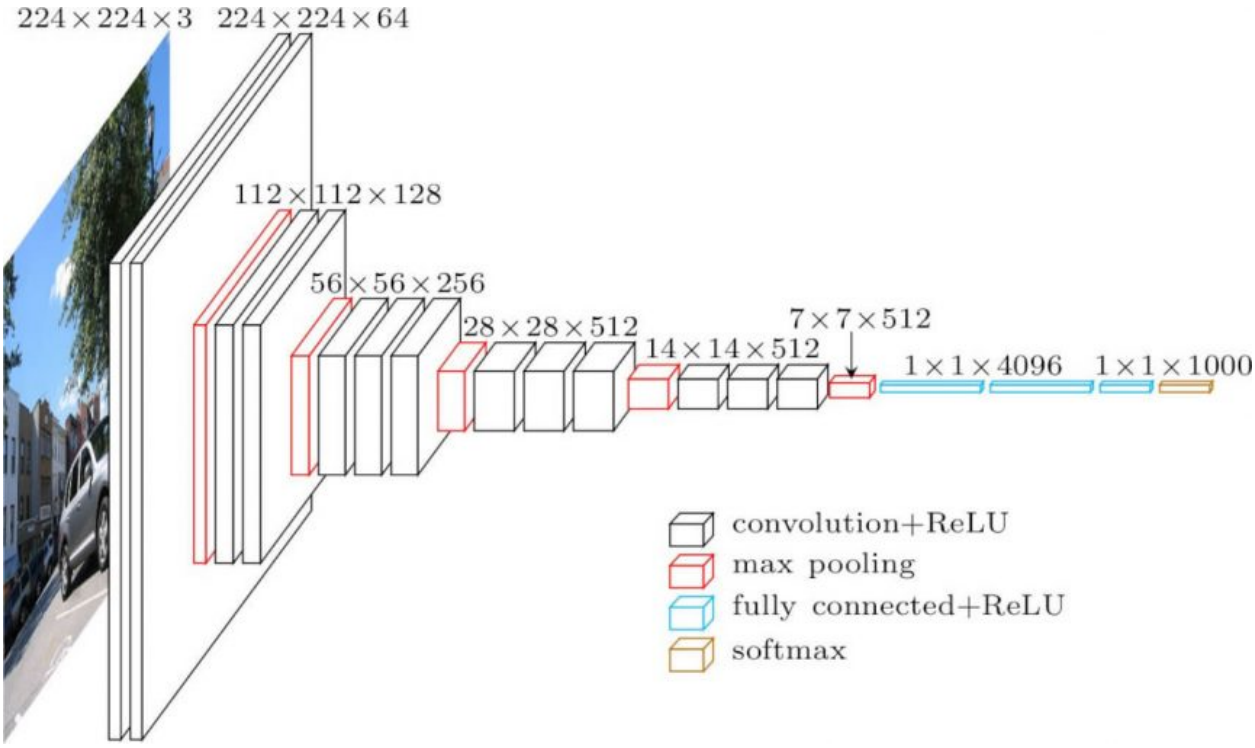


Figure 1.22: The architecture of the VGG-16 CNN

4. **GoogLeNet (Inception)**: In the same year as VGG-16, Google researcher team Christian Szegedy et al. released a model called GoogLeNet[38], and it was the first of the series Inception (Inception V1). Incepetion V1, was the winner of the ILSVRC 2014, with a top-5 error rate of 6.67%[12].

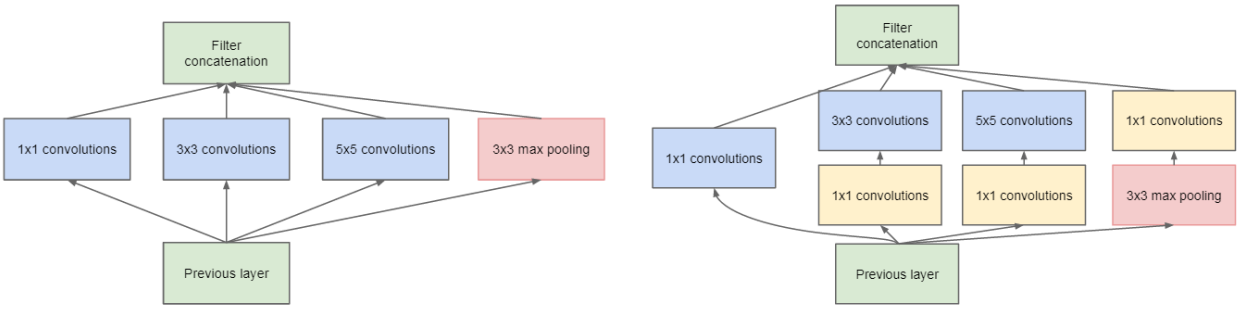
   GoogLeNet introduces the use of 1x1 convolutions for reducing input dimensions, plus, it applies convolutions with multiple sized filters on one input then stacks the output feature maps together, and that is all done thanks to Inception Modules.
   Here is an example that shows how does 1x1 convolution reduces computations. If we have an input volume of dimensions 10x10x256 and we need to perform 5x5 convolution with 20 filters, we have two choices:

   Without 1x1 convolution The total number of operations $= (5 * 5 * 256) * (10 * 10) * 20 = 12,800,000$.

   With 1x1 convolution we apply 1x1 then 5x5 convolution:

   - The operations for the 1x1 convolution $= (1*1*256)*(10*10)*20 = 512,000$
   - The operations for the 5x5 convolution $= (5*5*20)*(10*10)*20 = 1,000,000$
   - The total number of operations $= 512,000 + 1,000,000 = 1,512,000$ which is way better then 12,800,000.

   Figure 1.23 Shows the difference between a naive Inception module without 1x1 convolution (a) and an Inception module with 1x1 convolution(b).
   Inception V1 consists of a 22 layers (Figure 1.24), with about 4 million parameters to train.



(a) Inception module, naïve version (b) Inception module with dimension reductions

Figure 1.23: The naive Inception module and the Inception module with dimension reduction

[12] http://www.image-net.org/challenges/LSVRC/2014/results

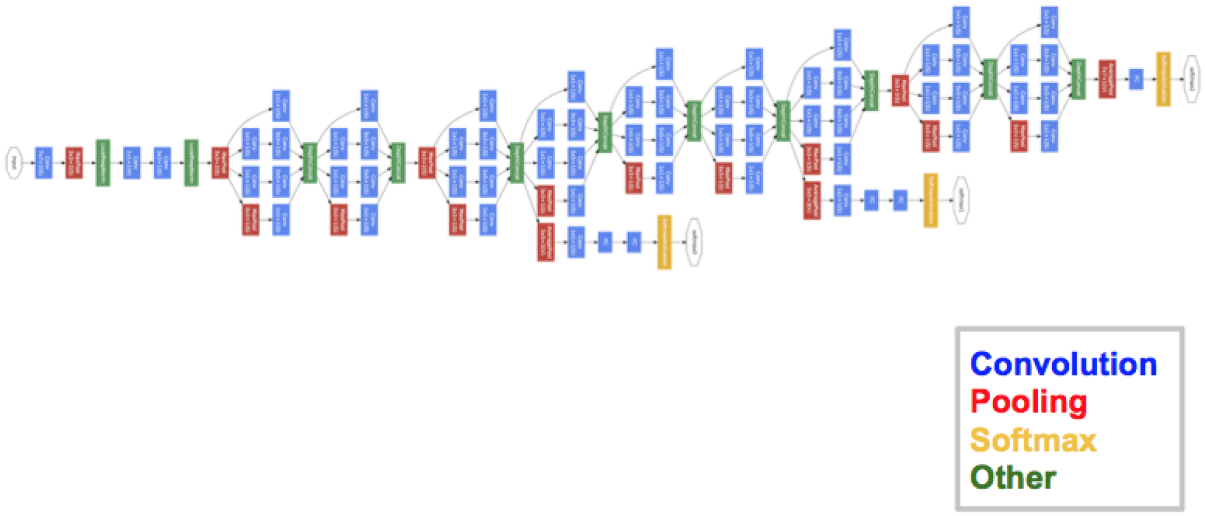


Figure 1.24: The architecture of the GoogLeNet (Inception) model

5. **ResNets**: ResNet was developed in 2015 by Kaiming He et al. It is one of the state of the art models for being the winner of ILSVRC 2015 in image classification, detection, and localization[13], as well as the winner of Microsoft Common Objects in Context (MS COCO) 2015 detection, and segmentation [14].

   Judging by the previous models, its intuitive to think that adding more layers to a network will improve its performance. However, this idea turned out to be wrong. Adding more layers causes the Vanishing/Exploding Gradients problem which leads to accuracy degradation as shown in Figure 1.25. ResNets overcame this challenge by introducing Skip/Shortcut Connection inside what is called a residual block. The original paper states that:
   "Formally, denoting the desired underlying mapping as H(x), we let the stacked nonlinear layers fit another mapping of F(x) := H(x)-x. The original mapping is recast into F(x)+x. We hypothesize that it is easier to optimize the residual mapping than to optimize the original, unreferenced mapping. To the extreme, if an identity mapping were optimal, it would be easier to push the residual to zero than to fit an identity mapping by a stack of nonlinear layers"[39]. These skip connections allowed the the implementation of very deep networks, without any vanishing gradients. In fact the authors of the paper increased the network depth to 152 layers, as a result, a 5.71% top-5 error rate is obtained which is much better than VGG-16, GoogLeNet (Inception-v1).

   Figure 1.26 illustrates the architecture of a residual block, while figure 1.27 shows a 34 layer ResNet side by side with a plain 34 layer network and the VGG-19 network.

[13] http://image-net.org/challenges/LSVRC/2015/results
[14] http://cocodataset.org/#detection-2015

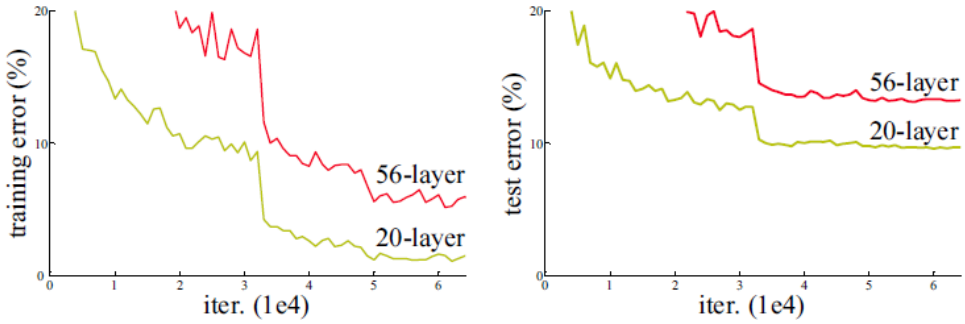


Figure 1.25: The training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks.

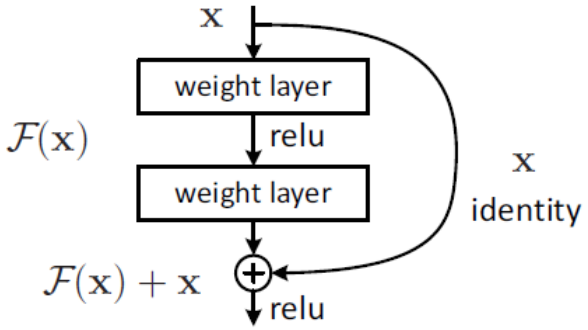


Figure 1.26: The structure of a basic residual block

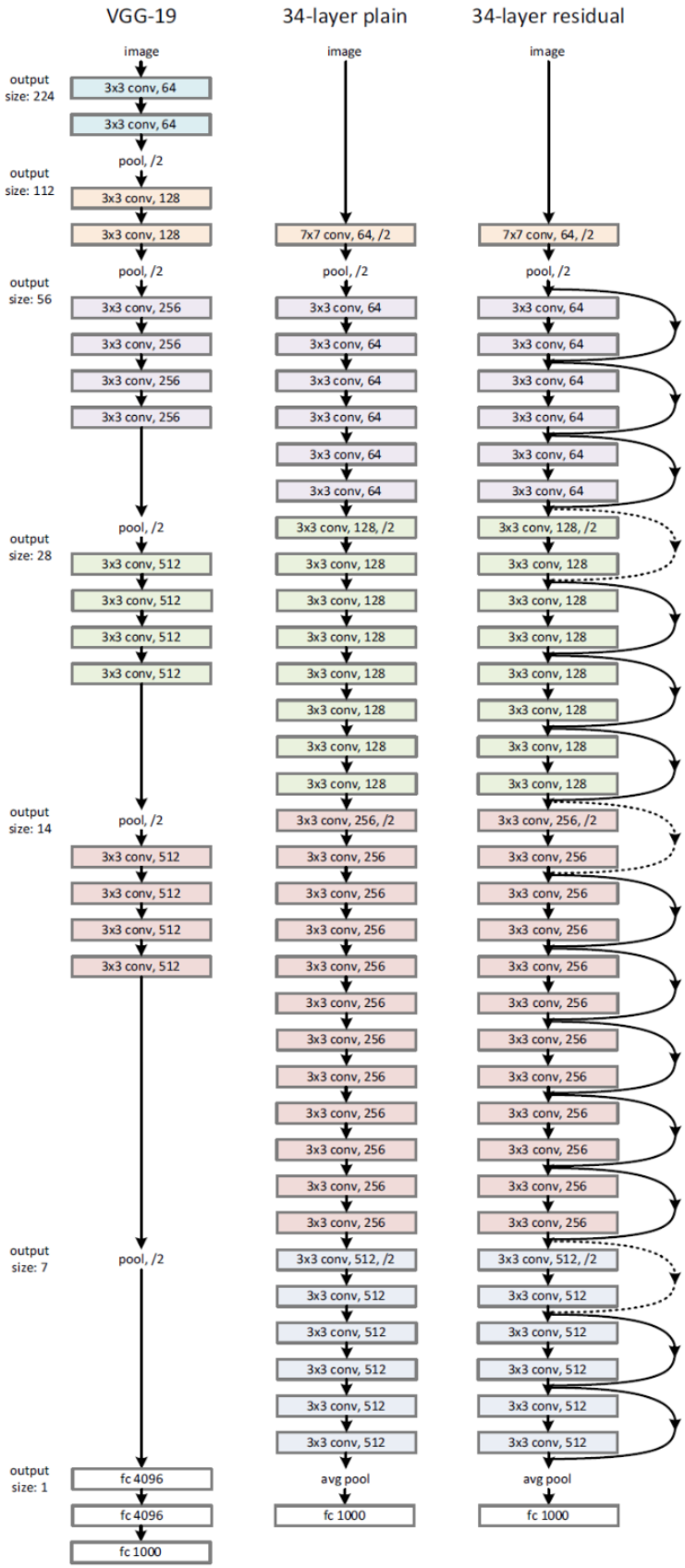


Figure 1.27: A residual network with 34 layers, a plain 34 layer network and the VGG-19 network

Training A Convolutional Neural Network Network training is the procedure of searching for the optimal set of weights for the network. Ideally, the network searches for the set of weights that leads to a correct prediction whenever a new data instance is passed through.

Before we discuss the training of an artificial neural network in detail, we must talk about the image preprocessing phase that takes place before the training.
Image Preprocessing: Image preprocessing consists of editing the dataset images to make them more convenient for training and testing. Here are some of the usual modifications applied before training:

Normalizing Pixel Values Pixel values in RGB images are between 0 and 255, and we usually normalize them to become between 0 and 1 by dividing each pixel by 255. This makes the network's weights converge faster towards the optimal values during training.

Rescaling Images Images in the dataset may be of different sizes, therefore they need to be reshaped in order to have uniform dimensions assuming that the subject of interest is in the middle.

Data Augmentation This technique aims to eliminate a problem called Overfitting. Logically, if we have small training data, our model has a low probability of getting accurate predictions for data that it hasn't yet seen, this is overfitting. To put it simply, if we are training a model to spot cats, and the model has never seen what a cat looks like when lying down, it might not recognize that in future. Data augmentation addresses this problem by applying transformations like rotation, skew, mirroring, ect. on the original dataset images, extending it to beyond what we already have.

All FFNNs including CNNs follow the same training procedure. They start by initializing the weights of the network randomly. At first, the network starts with bad accuracy and it tries to enhance it. Data instances are forward propagated through the network one by one and predictions are made. Then, these predictions are compared to the labels to compute a loss. A higher loss indicates bad accuracy. Of course our goal is to decrease the loss to a minimum value by the end of training. To do that, the weights get altered starting by the output layer and going back in the network. However, instead of editing the weights randomly, they are modified following a specific optimization algorithm, also known as an optimizer. This works because the problem of training a network is equivalent to the problem of minimizing the loss function with respect to the weights. The weights that minimize the loss function are the same as ones that give right predictions.

There are several loss function as well as optimizers. In the next section we will mention and explain some of the well known loss functions in DL.

1.9.7 Recurrent Neural Networks

RNNs are a class of neural networks adapted to processing sequences of inputs thanks to a temporal memory they possess. A memory is useful for tasks like predicting the next

word of a sentence, where previous inputs are important for predicting the next output.The first form of RNNs was popularized by John Hopfield in 1982 and they were called Hopfield Networks[40].

The most basic component of an RNN is the recurrent neuron. At each time step, the neuron receives an input scalar in addition to the output from the previous time step. It passes their weighted sum through a Tanh activation function producing an output. Joining several recurrent neurons produces a basic memory cell which interacts with vectors instead of scalars. An RNN can be represented against the time axis by performing an operation called unrolling, which is a substantial concept for handling RNNs. Figure 1.28 represents a basic RNN unrolled through 3 time steps.

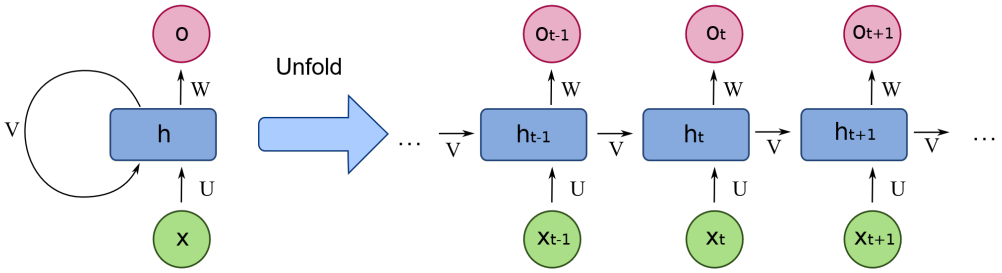


Figure 1.28: A basic unrolled recurrent neural network

h_t signifies the hidden state of the network at time step t and it is computed by equation 1.10:

$$h_t = \phi(X_t \cdot U + h_{t-1} \cdot V + b_h) \tag{1.10}$$

O_t is the output of the network at time step t, it is calculated by equation 1.11:

$$O_t = \phi(h_t \cdot W + b_o) \tag{1.11}$$

As shown in figure 1.28, at time step t, the network receives two input vectors:

- The input vector of the current time step X_t multiplied by a weight matrix U.
- The hidden state from the previous time step h_{t-1} multiplied by a weight matrix V.

These two vectors are summed up along with a bias vector b_h, and passed through an activation function ϕ to produce h_t.
Finally, to get the output O_t we multiply the hidden state vector h_t with a weight matrix W then add a bias vector b_o and pass the result to an activation function ϕ.

In this basic architecture, the output is just a weighted version of the state plus a bias[15]. However that is not always the case as there are more complicated architectures, where the output and the state of the network are computed using complex operations.

[15]In some sources, this simple architecture is refereed to as a Vanilla RNN

The mechanism presented in figure 1.28 allows the network to function in different ways, depending on the size of the sequence we provide as input, and the size of the output sequence we expect. Figure 1.29 summarizes some of the ways RNNs are used.

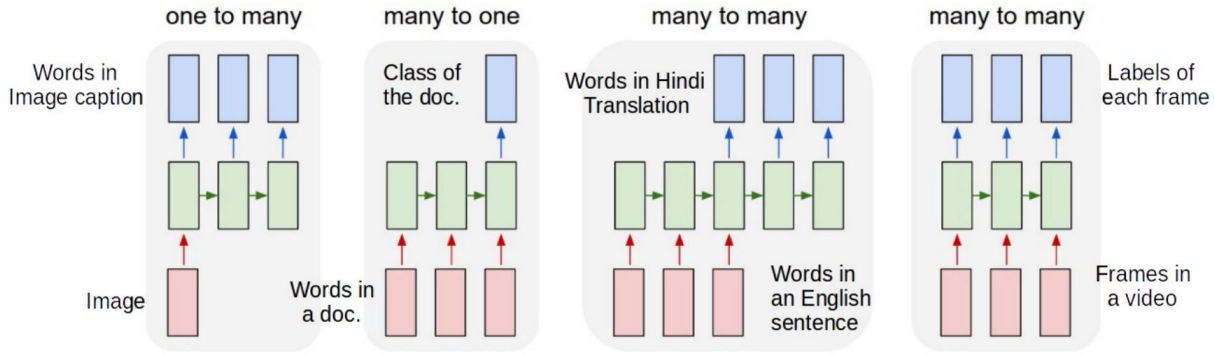


Figure 1.29: The applications of an RNN

The red rectangles in figure 1.29 represent input sequences with different lengths, and the green ones are the RNN unfolded throughout several time steps. Lastly, the blue rectangles resemble the output given by the network in each case. For example, In the one to many model, we can give a trained RNN a sequence of length 1 (an image), and it will output a sequence for each time step (sequence of words or a caption). In the next subsection we will explain how RNN are trained.

Training A Recurrent Neural Network Training an RNN properly requires data to be compatible with the task at hand. For example, if we are training a many to many model (Figure 1.29), the output for each time step must be predefined within the data because it is crucial for computing the loss. The training takes place after unrolling the network through time.

According to the backpropagation algorithm, we initialize the weights, apply the forward pass and get some predictions. Then, we compute the loss depending on the number of outputs we received. The next step is the backward pass in which we update the weights according to the gradients computed at each time step. The version of back propagation for RNN training is called Back Propagation Through Time (BPTT) .

In the backward pass, the algorithm updates the weights of the network by computing the gradients of the loss function with respect to the weights. In a vanilla RNN architecture, the gradients tend to get smaller as we back off to earlier steps, and that is because a gradient is mathematically computed by the product of a set of small factors. Since these gradients contribute dramatically in the weights update. The weights start to alter very slowly to a point where the update is nearly null, therefore the best solution for optimizing the loss function will never be reached. This is called the Vanishing Gradients problem.

There are other scenarios where the gradient gets very high, because the factors that influence it are high, and their product causes an explosion in value. As a result, the

weight update will be exaggerated and the weights will diverge from the minimum. This is the Exploding Gradients problem. These problems were explored in depth by Hochreiter in[41], and by Bengio, et al. in[42].

Another downside of vanilla RNNs is their incapability of modeling the long term dependencies in the data which results from the vaniching gradients. For example, if we are trying to predict the next word in a sentence, and we see an important word in the beginning of the sentence. The model will not be able to use the semantic value of that word to make correct predictions as the sequence gets longer.
Razvan Pascanu et al[43]. proposed Gradient Clipping as a solution for Exploding Gradients, while brand new architectures were invented to cope with the Vanishing Gradients problem.

Long Short Term Memory Long Short Term Memory (LSTM) is an architecture for RNNs introduced by Sepp Hochreiter and Jürgen Schmidhuber in 1997[44]. It was the solution for the vanishing gradients problem encountered by vanilla RNNs. LSTMs also solved the long-term dependency problem as they succeeded to easily capture relevant information throughout very long sequences.

LSTM memory cells keep track of two separate states. The long term cell state denoted by C_t, and the short term cell state denoted by h_t. Furthermore, LSTMs introduce the concept of gates which are sigmoid neural network layers, who's job is to control the flow of the information by outputting numbers between zero and one, deciding how much of the input should be let through and how much should be excluded.
The general intuition behind LSTMs is to learn what to store in the long-term state, what to throw away, and what to read from it. Figure 1.30 illustrates the mechanism behind an LSTM memory cell.

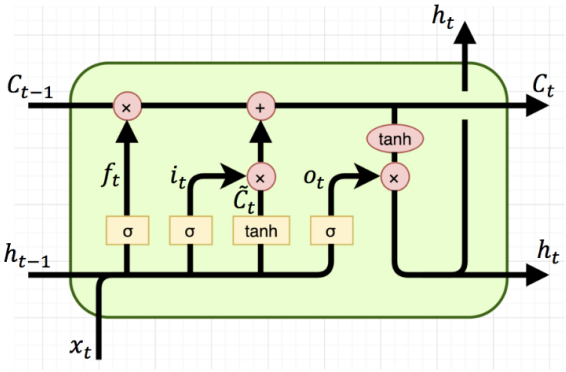


Figure 1.30: The mechanism of an LSTM memory cell

At each time step, the LSTM cell receives an input vector X_t, the short-term cell state from the previous step h_{t-1} and the long term cell state C_{t-1} from the previous step. The previous cell state information C_{t-1} enters the cell from left. First, it drops some information under the influence of the forget gate. Next, it receives some informa-

tion with the help of the input gate. Finally, it is sent straight out with no further modifications. h_{t-1} and X_t are fed to four gates:

The Forget Gate This gate is trained to determine which parts of the long term state should be erased. The output of this gate f_t is forwarded to an element wise multiplication with the long term cell state from the previous time step C_{t-1} to modify it accordingly. The vector f_t is calculated by equation 1.12:

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \tag{1.12}$$

Tanh activation function This is not necessarily a gate but its job is to analyze the current inputs X_t and the previous short term state h_{t-1} to produce a vector of new candidate values for C_t. $\tilde{C}_t$ is computed by equation 1.13:

$$\tilde{C}_t = tanh(W_C \cdot [h_{t-1}, X_t] + b_C) \tag{1.13}$$

Input Gate This sigmoid layer decides what information from the previous vector (candidate values) to add to the cell state. i_t is computed by equation 1.14:

$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i) \tag{1.14}$$

All what is left is to perform the updates to the previous long term state to get the current cell state C_t according to equation 1.15:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \tag{1.15}$$

Output Gate This gate filters the long term state C_t and decides what should be read and output at this time step h_t. It is calculated by equation 1.16:

$$h_t = o_t * tanh(C_t) \tag{1.16}$$

Where o_t is the output of the output gate and its given by equation 1.17:

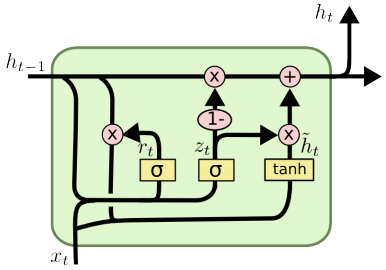
$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o) \tag{1.17}$$

In the equations 1.12, 1.13, 1.14 and 1.17, $[h_{t-1}, X_t]$ refers to the concatenation of the two vectors h_{t-1} and X_t, while W_f, W_C, W_i and W_o are weight matrices learned during training. b_f, b_C, b_i and b_o represent the bias vectors which are also learned during training. Finally the σ refers to the sigmoid activation function.

LSTM is widely used nowadays, thanks to the success it achieved in tasks related to sequence treatment. However, its is not the only architecture which uses gates, as there are many others. In the next subsections we will briefly explain some of them.

1. **LSTM With Peephole Connections**: This variant was proposed by Felix Gers and Jürgen Schmidhuber in 2000[45]. They add an extra input to the the forget gate and the input gate, which is the long term cell state from the previous time step C_{t-1}. Peephole connections turned out to be a good modification because they helped the network generate very stable sequences of highly nonlinear, precisely timed spikes.

2. **Gated Recurrent Unit**: The Gated Recurrent Unit (GRU) cell was proposed by Kyunghyun Cho et al. in 2014[46]. It is considered a simplified version of the LSTM, in which the long term state and the short term state are combined as one state denoted h_t. GRUs also merge the forget and input gates into a single update gate. In addition to a reset gate (see figure 1.31) which also decides how much past information to forget. GRUs are usually faster in training because of their simple structure and lower tensor operations.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Figure 1.31: The mechanism of a GRU memory cell

Other variants of RNNs include Depth Gated RNNs[47], Clockwork RNNs[48]. To answer the question of which RNN architecture is best, Greff, et al. performed a large scale study on variants of the LSTM architecture[49]. They concluded and none of the investigated modifications significantly improve performance although some of them effectively reduce the number of parameters and lower the computational cost of the RNN's training.

1.10 Loss Functions

1.10.1 Cross Entropy

Entropy is the measure of uncertainty associated with a given probability distribution. For example, supposing that X is the random variable that describes the class of a randomly picked image from a dataset. If all dataset images are of the same class then the entropy of X is 0, because there is no uncertainty about the class of any randomly picked image.

The Cross Entropy (CE) is a measure for estimating the difference between two probability distributions over a random variable. Usually, the first distribution is a target denoted as P, and the second is an approximation of the target distribution, and it is denoted Q. CE can be used as a loss function for ANNs, considering the label of an image, a target distribution P, and the model's prediction for that image, an approximated distribution Q. Given a random image from a dataset, the CE between a prediction vector Q and a label vector P is given by the formula 1.18:

$$H(P, Q) = -\sum_{i=1}^{C} P(i) \log(Q(i)) \tag{1.18}$$

Where C is the number of classes[50].

1.10.2 Binary Cross Entropy

This loss function is used in binary decisions, such as a cat/dog classifier. According to Andrew Ng's lecture in Coursera[51], the binary CE loss function is given by equation 1.19:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} Cost(y, \hat{y}) \tag{1.19}$$

Where:

$$Cost(y, \hat{y}) = \begin{Bmatrix} -log(y) & \text{if } \hat{y} = 1 \\ -log(1-y) & \text{if } \hat{y} = 0 \end{Bmatrix} \tag{1.20}$$

Where N is the number of images in the dataset, y is the prediction made by the model and $\hat{y}$ is the actual label. For each image in the dataset, the loss between the label and the prediction is calculated with the help of the log function. If the prediction is different from the label, the model is penalized and the loss for that image is high, on the other hand if the prediction is right the loss is 0. The final loss is obtained by averaging the losses for each image from the dataset.

Professor Andrew also stats in [52], that the previous formula can also be written as equation 1.21:

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=0}^{N} (y_i * \log(\hat{y}_i)) + (1 - y_i) * \log(1 - \hat{y}_i) \tag{1.21}$$

1.10.3 Categorical Cross Entropy

Categorical Cross Entropy is simply the the CE for a multi class classification problem[53], Where a data instance is one of many predefined classes. For example, the MNIST handwritten digit recognition where each image belongs to only one out of 10 classes. It is calculated by equation 1.22:

$$L(y, \hat{y}) = -\sum_{i=0}^{M} \sum_{j=0}^{N} (y_{ij} * \log(\hat{y}_{ij})) \tag{1.22}$$

This function computes the CE between the prediction (the vector of activations in the output layer) with the true distribution (The one-hot encoded label vector) for each image in the dataset. The loss is low when the output vector is close to the label vector. Otherwise, it is high.

1.10.4 Mean Squared Error

Mean Squared Error (MSE) is more convenient for regression problems, such as predicting future house pricing. it is calculated by equation 1.23:

$$L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^{N} (y_i - \hat{y}_i)^2 \tag{1.23}$$

This function simply computes the mean of the squared differences between the prediction and the label for each data instance. Infact the loss increases whenever the Euclidean distance between the predictions and the targeted label increases[23].

1.10.5 Mean Squared Logarithmic Error

Mean Squared Logarithmic Error (MSLE) is a variation of MSE, and it is also used for regression problems. There are two differences between MSLE and MSE:

- MSLE treats small differences between small labels and predicted values the same as big differences between large labels and predicted values. While MSE is sensitive towards outliers, therefore large errors are significantly more penalized than small ones.
- MSLE penalizes under-estimates more than over-estimates, which means that if the prediction is bigger then the label, the penalty will be larger then the case when the prediction is smaller. Which isn't the case for MSE.

MSLE is given by equation 1.24:

$$L(y, \hat{y}) = -\frac{1}{N}\sum_{i=0}^{N}(\log(y_i + 1) - \log(\hat{y}_i + 1))^2 \tag{1.24}$$

MSLE can be interpreted as a measure of the ratio between the label and predicted value. because (Equation 1.25):

$$\log(y_i + 1) - \log(\hat{y}_i + 1) = \log(\frac{y_i + 1}{\hat{y}_i + 1}) \tag{1.25}$$

Because both y_i and $\hat{y}_i$ can be 0, and $\log(0)$ is not defined, we add the number 1 to y_i and $\hat{y}_i$[54].

1.10.6 Mean Absolute Error

Mean Absolute Error (MAE) is similar to the two previous loss functions, plus it is also used for regression. The loss is the mean of the absolute value of the differences between labels and predicted values, expressed by the formula 1.26:

$$L(y, \hat{y}) = \frac{1}{N}\sum_{i=0}^{N}|y_i - \hat{y}_i| \tag{1.26}$$

Unlike MSE, MAE is not sensitive towards outliers. A disadvantage of MAE is that the gradient is harder to compute, and also it can lead to convergence problems due to its gradient's magnitude[24, 55][16].

[16]In the previous subsections the notations are modified and they are not the same as the ones found in the resources cited above in order to have a unified notation for the whole section

1.11 Optimizers

An optimizer is an implementation for the algorithm that helps us minimize the loss function and find the optimal set of weights for the ANN. The most commonly used algorithm for training a network is Gradient Decent (GD) . If we think of the loss function plot as a convex multi dimensional surface, GD tries to reach the lowest valley of this surface which represents the minimum value of the loss, by calculating the derivatives of the loss function (gradient) along every dimension, and taking steps towards the minimum iteratively.

There are different variations for GD, some of which are explained in the next few subsections.

1.11.1 Batch Gradient Decent

Also known as Vanilla GD. This method calculates the loss for the entire dataset, only then it updates the parameters θ to minimize the loss function J according to the rule given by equation 1.27:

$$\theta = \theta - \eta . \bigtriangledown_\theta J(\theta) \tag{1.27}$$

Because of computing the gradients for the whole dataset only to perform one update, batch gradient descent tends to be slow in converging. It is also intractable for large datasets that do not fit in the memory[56].

η is refereed to as the Learning Rate (LR), it determines the size of the step we take when trying to reach a minimum. This parameter is fixed in some implementations and modified in others. Anyways, it can not be too large because that will cause the algorithm to take huge steps and overshoot, but it also can not be too small because it will slow down the training procedure.

1.11.2 Stochastic Gradient Decent

Stochastic Gradient Decent (SGD) is a variation of GD in which the parameters are updated for each training example $x^{(i)}$ and label $y^{(i)}$. It follows the update rule given by equation 1.28:

$$\theta = \theta - \eta . \bigtriangledown_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{1.28}$$

Although SGD is way faster then batch gradient decent, the frequent updates usually cause heavy fluctuations when heading towards the minimum. These fluctuations lower the probability of getting stuck in a local minimum and enables the algorithm to jump to new and potentially better local minimums. However, it has been shown that when we slowly decrease the learning rate, SGD convergences somewhat smoothly, nearly the same as batch gradient descent[56].

1.11.3 Mini-batch Gradient Descent

Mini-batch GD is considered a compromise between the previously mentioned variants. It performs an update for every mini-batch of n training examples, according to the rule

given by equation 1.29:

$$\theta = \theta - \eta . \bigtriangledown_\theta J(\theta; x^{(i:i+n)}; y^{(i:i+n)}) \tag{1.29}$$

This approach leads to a more stable convergence and better efficiency in computing the gradients. Mini-batch gradient descent is typically the algorithm of choice when training a neural network with common mini-batch sizes of 50 to 256[56].

1.11.4 Adagrad

Adagrad is a gradient-based optimization algorithm proposed by John Duchi et al in [57]. Instead of updating all the parameters θ using the same LR η, Adagrad uses different LRs for every parameter θ_i at every time step t.
For simplicity, we will set $g_{t,i}$ to be the gradient of the loss function with respect to the parameter θ_i at time step t. The update rule for Adagrad is given by equation 1.30:

$$\theta_{t+1,i} = \theta_{t,i} - \frac{\eta}{\sqrt{G_{t,ii} + \epsilon}} . g_{t,i} \tag{1.30}$$

Where:

- $\theta_{t+1,i}$ refers to the parameter θ_i at time step $t+1$, or the new parameter value.
- $\theta_{t,i}$ refers to the parameter θ_i at time step t, or the old parameter value.
- G_t here is a diagonal matrix where each diagonal element $G_{t,ii}$ is the sum of the squares of the gradients with respect to θ_i up to time step t.
- ϵ is a very small positive number that eliminates the division by zero just in case (usually ϵ is in the order of 10^{-8}).
- η is an initial value for the learning rate, usually 0.01.

Using Adagrad eliminates the need to manually set the LR. However, the growth of the accumulated squared gradients may cause it to shrink, and become very small to a point where the algorithm cannot reach better solutions[56].
A vectorized implementation for the update rule where $\odot$ refers to matrix vector multiplication is given by equation 1.31:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{G_t + \epsilon}} \odot g_t \tag{1.31}$$

1.11.5 AdaDelta

AdaDelta[58] aims to reduces the aggressive decreasing of the LR performed by Adagrad. Instead of accumulating all squared gradients from previous time steps, this method defines a recursive decaying average of all past squared gradients. This value at time t is denoted $E[g^2]_t$, and it is called the running average. $E[g^2]_t$ depends only on the previous average $E[g^2]_{t-1}$ and the current gradient and its calculated by equation 1.32:

$$E[g^2]_t = \gamma E[g^2]_{t-1} + (1-\gamma) . g_t^2 \tag{1.32}$$

Where γ is set to a small value, usually 0.9.
The term G_t is replaced with the running average$E[g^2]_t$ in the update rule from AdaGrad so we get equation 1.33:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{E[g^2]_t + \epsilon}} g_t \tag{1.33}$$

As a result, the growth of the average is restricted, plus all the past gradients are contributing in the learning rate of time t thanks to the recursive formula[56].

1.11.6 RMSProp

This method was proposed by Geoffery Hinton in a lecture in one of his Coursera Classes[17]. It shares the concept idea with AdaDelta as they both emerged from trying to resolve Adagrad's radically diminishing LR problem. In fact RMSProp uses the same update rule (equation 1.33). Moreover Hinton suggests γ to be set to 0.9, with a default value of 0.001 for the LR[56].

1.11.7 Adam

Adaptive Moment Estimation (Adam)[59] is another method of gradient optimization. It keeps a decaying average of past squared gradients v_t just like RMS Prop And AdaDelta, but also another decaying average of past gradients m_t that plays the role of a Momentum[18]. m_t and v_t are given by equations 1.34 and 1.35 respectively:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{1.34}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{1.35}$$

Because the authors observed that these averages were biased towards zero, they applied bias correction, so they proposed $\hat{m}_t$ and $\hat{v}_t$ which are calculated using equations 1.36 and 1.37 respectively:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{1.36}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{1.37}$$

The update rule is slightly different then AdaDelta and RMSProp and is given by equation 1.38:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t \tag{1.38}$$

The authors recommend a β_1 of 0.9, a β_2 of 0.999 and an ϵ of 10^{-8}, because they show empirical improvements in practice and better performance in comparison with to other algorithms[56].

[17] http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf

[18] Momentum is also a method that helps accelerate SGD in the direction of the best minimum. For detailed explanation refer to the original paper [60]

1.12 Regularization

One of the main challenges encountered by ML is overfitting. we say that a model is overfitting when it is performing good in the training data while failing to classify the test data. Moreover, the model is learning the details and noise in the training data to a point where it can not recognize new data, because the noise learned from the training data does not apply to new data. As a result, the model starts loosing his ability to generalize.
In order to overcome this problem, we use what is called regularization techniques. The next subsections summarize some of the regularization techniques for tackling CNN's overfitting:

1.12.1 Dataset Augmentation

It is intuitive to think that training a model on larger datasets gives the model better performance. Based on this idea, data augmentation aims to extend the data to provide more training examples using only the data at hand. It involves performing some modifications to the dataset images before training, modifications such as flipping the image horizontally or vertically, rotating, rescaling the image by cutting out a section from it, cropping, applying noise to the image making the scene a bit distorted, mirroring, etc(Figure 1.32). The edited images will be inserted in the dataset and treated the same as every other training example. As a result, the model is exposed to more data hopefully reducing overfitting.

Data augmentation is implemented in Keras as the ImageDataGenerator[19] class, such that it happens inside the random access memory and not even effect the original data on the hard disk, we just have to choose what modifications to make and tweak their parameters.

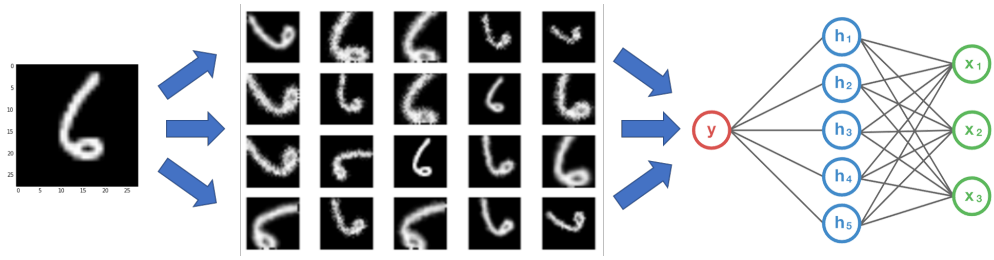


Figure 1.32: An illustration of the data augmentation in action

1.12.2 Early Stopping

Training the model for too long can also be considered a motive for overfitting, as there are many scenarios where the training accuracy and the test accuracy start diverging from each other after an excessive number of epochs, as shown in figure 1.33. Therefore, we could definitely use a tool that halts the training when the model starts overfitting.

This trick is called Early Stopping. It consists of measuring both training and test accuracy at the end of each epoch. The model either stops when the training accuracy reaches a certain value, when the test accuracy reaches a certain value, or when no improvement is observed, hence, further training is meaningless.

[19] https://keras.io/preprocessing/image/#imagedatagenerator-methods

Early stopping is implemented through several methods inside the Callback[20] class In Keras.

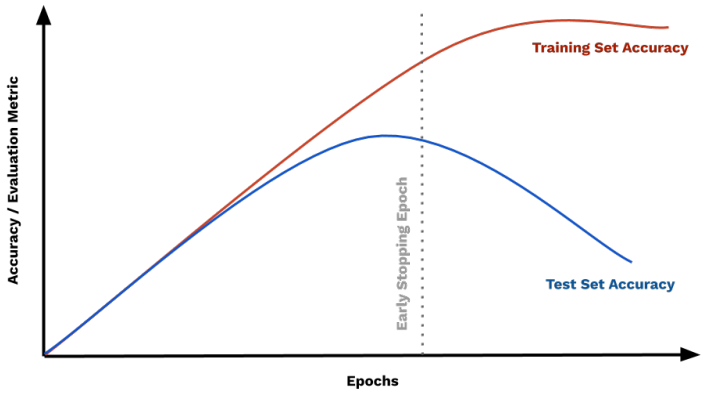


Figure 1.33: Overfitting after an excessive number of epochs

1.12.3 Dropout

Another reason for overfitting in ANNs is units getting dependent on only specific inputs, inputs that usually have larger weights over the others. Dropout, which was first proposed by Geoffrey Hinton et al.[61] addresses this problem by randomly dropping a percentage of units from the layers(Figure 1.34). Dropping out a unit means temporarily erasing it from the network during training along with all its connections. This causes the units in the next layer to not rely on any specific inputs as all inputs have equal chances of being present as well as being absent. This also causes the network to gain uniform weight distributions, hence, no connections are biased with larger training weights. During testing, froward propagation is performed by using the trained uniform weights, and this can be considered an approximation of averaging the predictions of all the thinned networks formed each time a units is dropped during training[62].

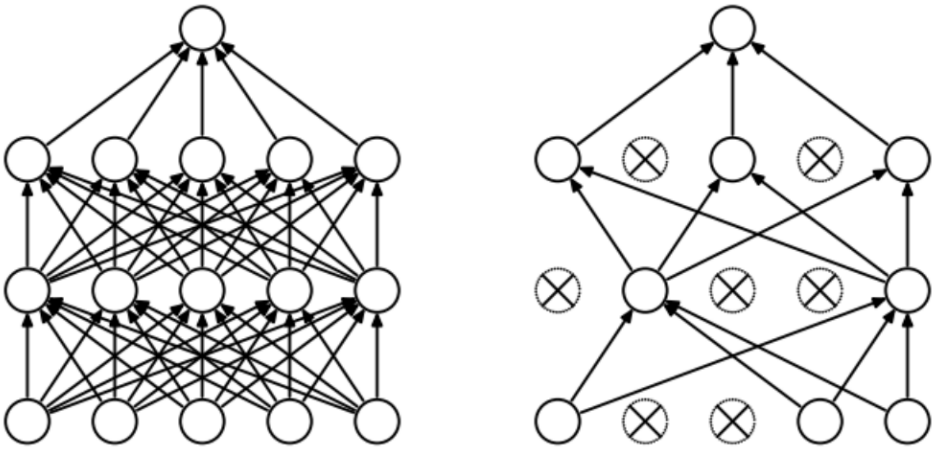
Dropout is implemented in Keras by adding a Dropout layer from the Core Layers[21], which takes the percentage of how many nodes should be dropped as a parameter.

1.12.4 Dense-Sparse-Dense Training

Dense-Sparse-Dense (DSD) training is a method for regularizing deep ANNs and improving their performance. It was proposed by Song Han et al[63].
This method consists of three separate training stages (illustrated in figure 1.35):

- First, we train a dense ANN to learn connection weights and determine the important connections for the network. This is the the first Dense training stage.

[20] https://keras.io/callbacks/
[21] https://keras.io/layers/core/



(a) Standard Neural Net (b) After applying dropout.

Figure 1.34: The difference between a the presence (right) of dropout and its absence (left).

- In the second Sparse training stage, we regularize the network by pruning the connections with small weights, and train the network again, introducing the sparsity constraint this time.
- The final Dense training stage increases the model's capacity by eliminating the sparsity constraint, re-initializing the pruned parameters to zero and training the dense network again.

During training, DSD introduces only one extra hyper-parameter which is the sparsity ratio in the Sparse training step.

DSD training improved the performance for a wide variety of ANNs, including already state of the art models. For example, on ImageNet, DSD improved the Top 1 accuracy of GoogLeNet by 1.1%, VGG-16 by 4.3%, ResNet-18 by 1.2% and ResNet-50 by 1.1%, respectively[22]. DSD training's performance didn't stop there as it improved even RNNs and LSTMs for caption generation and speech recognition[63].

[22]The authors made DSD pretrained models available to download online at https://songhan.github.io/DSD/.

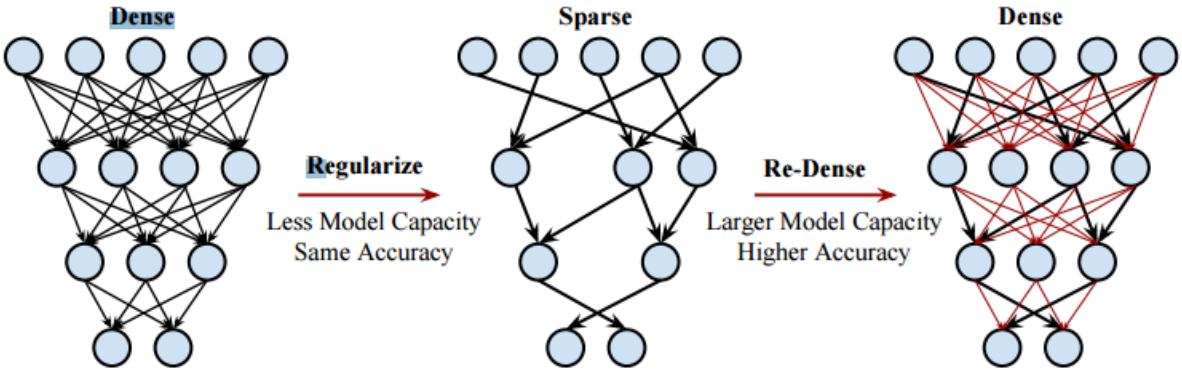


Figure 1.35: The Dense-Sparse-Dense Training Flow.

1.13 Transfer Learning

It is a known fact that powerful models require huge amounts of training data, machines with high computational power and a lot of training time to reach their maximum ability. Nevertheless they give remarkable accuracy. Transfer learning takes advantage of the knowledge acquired by these large scale models, by using it for solving a similar but more specific problems.

Generally, transfer learning refers to the trick of using a model trained on one problem, in some way on a another related problem. It is explained briefly by Jason Yosinski et al. as follows:
"In transfer learning, we first train a base network on a base dataset and task, and then we repurpose the learned features, or transfer them, to a second target network to be trained on a target dataset and task. This process will tend to work if the features are general, meaning suitable to both base and target tasks, instead of specific to the base task"[64]. Precisely, what we are actually doing is copying over some parameters from the first model to our model then adding some layers and training them while locking (freezing) the copied parameters. It means that the backward pass in training will edit only the weights of the last layers, not including the copied weights. The resulting model will be somewhat specified for our task by using the features extracted by the copied parameters. This form of transfer learning is referred to as inductive learning.

For image classification problems, it is common to use one of the state of the art models such as the AlexNet, GoogleNet, etc. These models are available online and can be incorporated directly into new models that expect image data as input. For example:

- Oxford VGG Model[23].
- Google Inception Model[24]
- Microsoft ResNet Model[25]

[23] http://www.robots.ox.ac.uk/~vgg/research/very_deep/
[24] https://github.com/tensorflow/models/tree/master/research/inception
[25] https://github.com/KaimingHe/deep-residual-networks

Transfer learning can be considered a shortcut to saving time and getting better performance. We can rely on it if we have less data and unworthy hardware, as it can enable us to develop skillful models that are usually out of our reach because we don't have the adequate GPUs that offer great computational power.

1.14 Conclusion

In this chapter, we introduced AI, ML and DL and highlighted their relationship. Furthermore, we went through DL in more depth explaining its origins, its inspirations, its challenges, the tools it employs and a short recap of the maths behind it. Until now, we've successfully layed down all the preliminary knowledge that the reader needs to progress further in this report.

The next chapter will be dedicated to the main problem which is video classification.

Chapter 2

Video Classification using Deep Learning

2.1 Introduction

The tremendous increase in the internet bandwidth and storage space, has made images and videos two of the most consumed data types. Consequently, the AI community took an interest in these two data types. Images might have been considered more than videos, but that is only because of the lack of organized video data for video processing, and the cost that comes with it. However, many experiments and studies were conducted lately trying to exploit videos and take advantage of their dynamic nature.

In this chapter we will first talk about CV and give a historical brief about how DL improved some of its related tasks. Next, we will mention some of the tasks involving videos as data for training and testing as well as the video datasets availability nowadays. Our interest is mainly on the video classification task, so we will define it and explain the different challenges encountered with it. Finally we will explain the different approaches made by the AI community to solve it.

2.2 Computer Vision

CV is a trending research field within AI. According to Ballard and Brown, CV is:
"The construction of explicit, meaningful descriptions of physical objects from images"[65].
However, Trucco and Verri defined CV as:
"Computing properties of the 3D world from one or more digital images"[66]. Another definition made by Sockman and Shapiro was:
"To make useful decisions about real physical objects and scenes based on sensed images"[67].
CV treats several problems:

Object Detection It is the task of recognizing an object and its location within an image. The algorithm usually surrounds all the detected objects with labeled tight rectangular boxes. To achieve that, the first developed architecture used Region based Convolutional Neural Networks (R-CNN)[68] .The detection speed was later improved by the model Fast R-CNN[69]. However, in the same year, the model Faster R-CNN[70] was invented and even though it achieved much better speeds and higher accuracy, it wasn't the best. More efficient detection systems emerged, like You Only Look Once (YOLO)[71], Single Shot multibox Detector (SSD)[72] and Region-Based Fully Convolutional Networks (R-FCN)[73].

Semantic Segmentation The idea of this task is to not just classify the objects in an image, but to classify each pixel as the object it contributes in. In other words, to understand the semantic role of each pixel in the image. The first approach was patch classification and it involved feeding patches of different sizes to a trained CNN, but this method turned out to be inefficient and computationally expensive. Therefore several papers shifted towards Fully Convolutional Networks (FCNs)[74, 75, 76, 77] as an alternative, because they accepted images of any size and were also much faster compared to the patch classification approach.

Image Classification For image classification, we are given a set of images, each one is labeled, and belongs to a specific class. Our goal is to classify another set of never seen images, and measure the accuracy of the classification. For humans, this task is considered trivial. However, it remains a challenge for computers. Image classification has been approached using hand-engineered features like in Scale-Invariant Feature Transform (SIFT)[78], and Histograms of Oriented Gradients (HoG)[79]. But their results were incomparable with the first CNN architecture AlexNet[15] for image classification. Later, CNNs became the best fitting tool for this task among various others in CV.

In the next section, we will dive into the video classification problem starting with a proper definition for video data.

2.3 Video Analytics Tasks

Video data is just as important as image data, judging by the amount of content and information videos can acquire thanks to the temporal aspect they add to images. Granted, massive amounts of video data is being produced nowadays. Under those circumstances, the need to structure and analyze this data grows and a lot of research exists to achieve that. Here are some of the tasks that involve videos as training/testing data:

Action Recognition is the problem of identifying events performed by humans given a video[80].

Video Captioning is the process of summarizing the content of a video into a short textual form[81].

Video Segmentation consists of dividing a video into separated collections of consecutive frames that are homogeneous according to a predefined criteria[82].

Video Classification is the problem of classifying a set of videos as being one of several predefined classes.

2.4 Video Definition

Suppose that we have a set of white pages, each page has an image drawn in it. If we flip through the pages at high speed, the scene would appear animated and the shapes in the pages would appear moving. That is the intuition behind videos.

A video is defined as a series of still images that, when viewed successively at a high speed, gives the appearance of motion.

Moreover, videos are a powerful data source because they can provide us with valuable information in any field. Especially with the evolution of video capturing devices such as cameras, cellphones, drones, ect.

2.5 Video Classification

Video Classification is basically the same task as image classification, but instead of images we classify videos. We want to build a model that can predict the class of a video, based on the knowledge extracted from a dataset of videos. Each video belongs to a different predefined class. Of course the model will be trained on a training set and later tested on a validations set.

In the next few sections, we will explain the motives behind this task, the challenges and the data availability for it.

2.5.1 Video Classification Motivations and Challenges

Because of their dynamics structure, Videos are considered way more expressive then images, as they add an additional property which is time. Moreover, video data is continuously being generated, published and spread nowadays, becoming an element of great value for data hungry deep learning algorithms. All things considered, videos are nothing but a set of images. That implies that video classification is an extension of image classification, as the former usually builds up the success made in the latter. As a result, it's expected that during implementing the two tasks, the same difficulties are encountered.

First there's the computational cost of training. It is known that the image classification state-of-the-art models took weeks of training on powerful machines. Therefore training a video classifier will definitely be more expensive and time-consuming. Because we are not only trying to classify one RGB image at a time. Instead, for every video there are thousands of images to classify.

The second challenge encountered while classifying videos, is the lack of large video datasets, and this has been the case for image classification in its early stages. This problem is mainly related to the storage size of videos. A video allocates more memory than an image, and the size of one second of footage is equivalent to the size of 30 images for a regular video. That means that image datasets are very small when compared to video datasets. That doesn't mean that there is no video datasets whatsoever, but this problem is still an obstacle worth mentioning. In the next section we will discuss some of the most famous available datasets for video classification.

2.5.2 Video Classification Datasets

YouTube Sports-1M The YouTube Sports-1M is an academic dataset, that accompanied the paper "Large-scale Video Classification with Convolutional Neural Networks"[2] by Andrej Karpathy et al. It was constructed in 2014 and it included 1,133,158 youtube videos annotated with 487 sports labels. The creators put together this dataset by gathering URLs of youtube videos of different sport activities and the labeling was done using the YouTube Topics API[1]. To make the dataset available for every one the

[1]The YouTube Topics API is used to search for videos matching specific search terms, topics, locations, publication dates, etc. For more details refer to the official website https://developers.google.com/youtube/v3/docs/search/list

creators uploaded a github repository[2], which contains all the related files including a text file named "sports_mids.txt" with all the machine IDs needed to retrieve videos from youtube using the topics search API mentioned above.

YouTube-8M YouTube-8M is a large-scale labeled video dataset. In fact, it is considered the largest multi-label video classification dataset ever. It was announced by Google Research in 2016[83] consisting of about 8 million videos accumulated to 500 thousand hours of footage from 4,800 classes[3]. For the labeling , the creators used Knowledge Graph entities to describe the main theme of each video. For example, a video of biking on dirt roads and cliffs would have a central topic/theme of Mountain Biking, not Dirt, Road, Person, Sky, and so on. Therefore, the aim of the dataset is not only to understand what is present in each frame of the video, but also to identify the few key topics that best describe what the video is about.

UCF-101 Another widely used dataset is the UCF-101 dataset. It was originally put together for the experiment performed by Somorro et al. in 2012[84]. Although it was meant for action recognition tasks, a lot of researchers use it nowadays for video classification. UCF-101 consists of 13320 unconstrained videos downloaded from Youtube, and divided into 101 action classes. The videos belong to five general categories: Human-Object Interaction, Body-Motion Only, Human- Human Interaction, Playing Musical Instruments and Sports. At the time, UCF-101 was claimed to be the most challenging dataset of actions due to its large number of classes, large number of clips and also unconstrained nature of such clips.
In the next section we will discuss the different approaches made by the DL community to tackle this problem.

2.6 Video Classification State of The Art

Most researchers often treated video classification using one of three approaches: Image based video classification, Advanced CNN architectures or Modeling the long term temporal dependencies using an RNN. In the next subsection, we will explain each approach and highlight some of the famous papers that followed it.

2.6.1 Image-Based Video Classification

The first approach relies on the fact that a video is just a collection of images. Therefore, feature representations are extracted separately for each frame, then these features are aggregated to produce video-level representations, which will be input for a classic classifier such as an SVM or an ANN.

Many classical hand engineered image features have been generalized to videos. Features like 3D Scale Invariant Feature Transforms (3D-SIFT)[85], extended SURF[86], 3D

[2]The repository is available at https://github.com/gtoderici/sports-1m-dataset

[3]The YouTube-8M can be explored online via this link. https://research.google.com/youtube8m/explore.html

Histograms Of Optical Gradients (HOG3D)[87]. However, one of the most referenced works involving hand designed features is proposed by Wang et al. in [88, 89].

In 2012, Wang et al. proposed a video representation based on dense trajectories and motion boundary descriptors[88]. Their idea was to sample feature points on a dense grid in each frame. Then, track these feature points using a dense optical flow algorithm to create dense trajectories which are useful for capturing the local motion information of the video[4]. Furthermore, they obtained video level representations by extracting hand designed descriptors such as Motion Boundary Histograms (MBH) along the trajectories, then encoding them as Bag Of Features (BoF) representations. Video level representations were later classified using a non-linear SVM classifier.

In 2013, Wang et al. enhanced the dense trajectories method by explicitly estimating camera motion. The paper [89] shows that the performance can be significantly improved by removing background trajectories and only focusing on more relevant feature points.
The methods used in [88, 89] can be classified as image based video classification because the order of frames was not considered due to the BoF encoding.

Zha et al. studied image based video classification and proposed [1] in 2015. Their goal was to test the performance of CNNs trained for image classification on video classification. Thus, they experimented on frame level features extracted from different hidden layers of AlexNet[15] and VGG[37] networks. These features were aggregated with spatio-temporal pooling and normalization to get video level features which were fed to a trained linear SVM classifier(Figure 2.1).

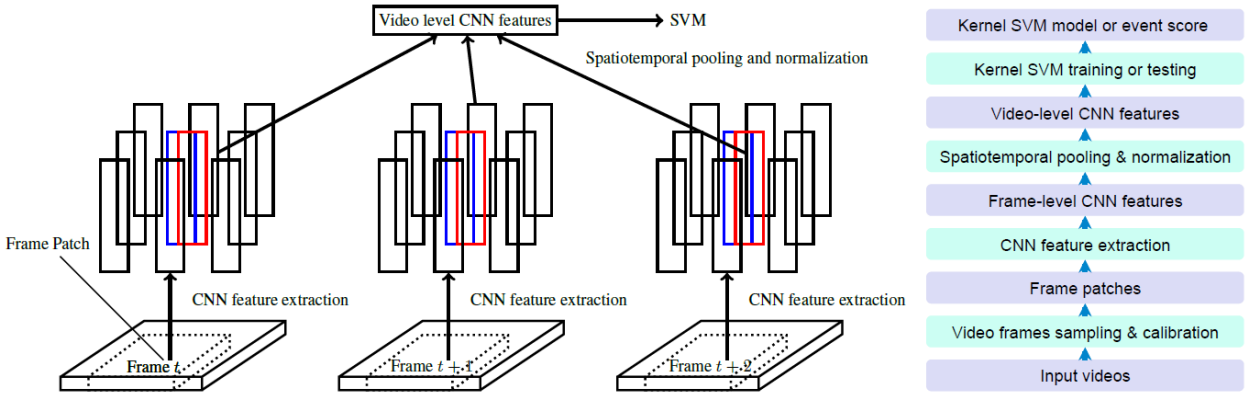


Figure 2.1: An overview of the proposed video classification pipeline by Zha et al. in [1].

Zha et al. also extracted frame level features using the standard SIFT descriptors[78], and the more sophisticated motion-based Improved Dense Trajectories (IDT)[89], then encoded these low-level feature descriptors as a fixed-length video-level FVs.
Fernando et al.[91] followed a similar pipeline as [1] in 2016, when they applied Rank Pooling on frame level features extracted from a CNN and demonstrated that such a model can be

[4]Wang et al. used the algorithm in [90] to extract the dense optical flow fields for each feature point. For more explanation about Optical Flow refer to https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html#dense-optical-flow-in-opencv

trained in an end-to-end fashion. Other orderless aggregation techniques have been widely used. For example, in [92], NetVLAD[93] which is an altered version of Vectors Of Locally Aggregated Descriptors (VLAD)[94] is used for for aggregating video and audio features from the Youtube-8M dataset, along with Context Gating for modeling inter dependencies between network activations.

Recently, many researches followed a different path in which they build video representations like Deep Features Video Matrices [95] or Semantic Streams[96] based on the visual objects present in video frames.

2.6.2 Advanced CNN Architectures

The success that CNNs achieved in learning deep features from raw image data was inspiring, and researchers started thinking of training CNNs on video data instead of images with the objective of learning hidden spatio-temporal patterns.

In 2014, Andrej Karpathy et al. performed a large-scale video classification[2] on the Sports-1M dataset which was first introduced with this work. The experiment was based on CNN architectures extended in time domain. The team empirically tested four techniques for fusing temporal information extracted from video frames. The idea is to divide the video into equal time windows, and sample frames in various ways, then feed these frames to several CNN architectures aiming to reach an architecture that extracts the best spacio-temporal features for video classification. Figure 2.2 shows the architectures proposed in the paper.

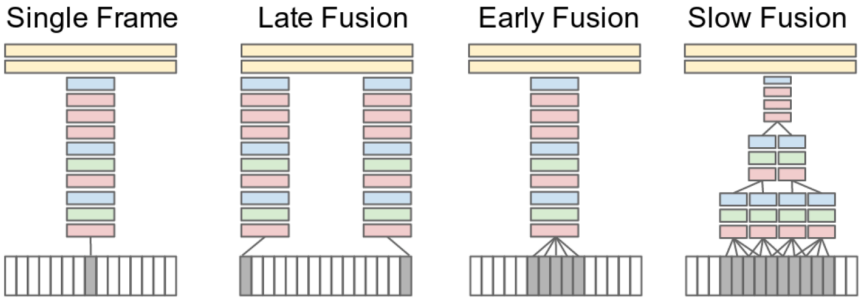


Figure 2.2: The proposed CNN architectures for fusing time information by Karpathy et al. in [2]. Red, green and blue boxes indicate convolutional, normalization and pooling layers respectively.

Karpathy et al. also worked on speeding up the training time. To do that they proposed a multi resolution architecture in which two networks are presented with two input streams (Figure 2.3). A Contex Stream which receives the frames at a downsampled lower resolution, and a Fovea Stream which receives the center region of the frames at the original resolution. Finally, the activations from both streams are concatenated and fed into the a fully connected layer with dense connections.

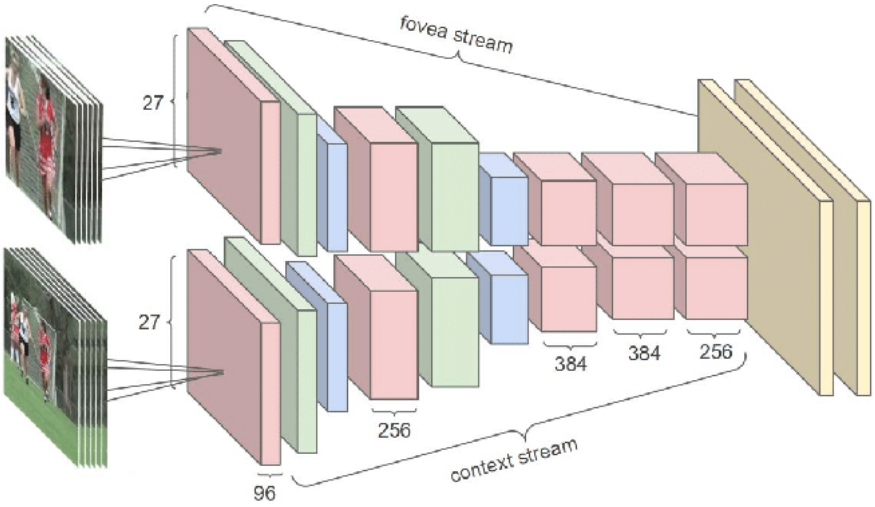


Figure 2.3: The proposed multi-resolution CNN architecture by Karpathy et al. in [2].

In the same year, Tran et al. worked on deep 3D CNNs, as feature extractors for several video analysis tasks in general[97]. Their network succeed to extract generic spatio-temporal features via 3D convolutions and 3D pooling operations instead of plain 2D convolutions. In 2019, Channel-Separated Convolutional Networks [98] was introduced as a way of factorizing 3D convolutions, reducing computations and improving accuracy.

Another interesting approach was adopted by Karen Simonyan and Andrew Zisserman in 2014, and it was called the Two-Stream Convolutional Network[3]. Although it was aimed for action recognition but we had to mention it because the next work on video classification relies entirely on it. In this method, spatial features are extracted from frames separately using a spatial CNN. Whereas the temporal features are extracted from stacks of multi-frame optical flow images[5]. Both spatial clues and temporal clues are extracted from the two streams and the soft max scores are combined using fusion methods.
Figure 2.4 illustrates the proposed two-stream architecture by Simonyan and Zisserman.

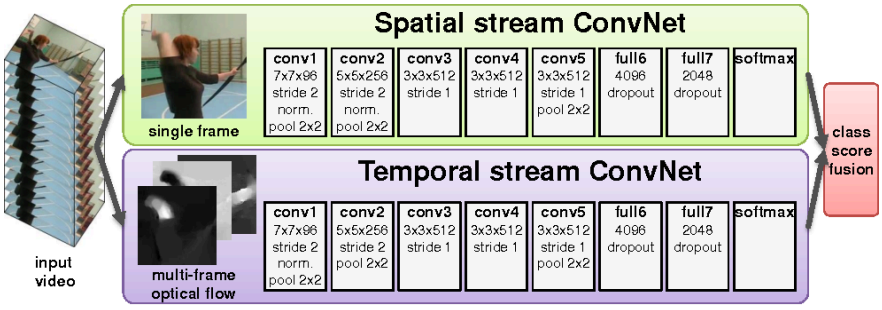


Figure 2.4: The proposed two-stream architecture for video classification by Simonyan and Zisserman in [3].

In 2015, Hao Ye et al.[99] conducted a study based on [3] to investigate important implementation options that may affect the performance of deep networks on video classification.

[5]Multi-frame optical flow image explicitly describes the motion between video frames.

They tested several implementation options such as network architectures, fusion strategies, learning parameters, prediction options to reveal the effect of each option on the model's performance. Finally, they concluded that deeper models are definitely better as long as the training data is sufficient. Plus, combining predictions from the spatial and the temporal streams is useful.

The two stream CNN inspired many other works such as [100] in which Xie et al. recently proposed a similar architecture, where spatial information is modeled with a spatial network, and temporal information is modeled with a temporal network that receives differential images instead of optical flow images. Moreover, instead of plain CNNs, Xie et al. used a Multi-scale Pyramid Attention (MPA) layer to capture multi-scale features from different stages of the spatial network and the temporal network, and then combined these multi-scale information into new video representations.

2.6.3 Modeling Long-Term Temporal Dependencies

Image based video classification[1] was efficient to an extent. However, the features gathered from single frames are treated as a BoF for which the order is not considered. This means that the semantic relation between frames isn't exploited at all. On the other hand, using advanced CNN architectures also had this drawback, because CNNs captured the semantic relation between only adjacent frames. The two stream CNNs[3] used optical flow images which only depicts movements within a short time window. While neither 3D CNNs[97], nor extended CNNs[2] could take a large number of stacked frame as input because that will be computationally very expensive.

Motivated by the fact that multiple important actions for classification happen in separated moments throughout videos. Researchers tried to leverage RNNs, for modeling the temporal relations between video frames. Specifically, LSTMs were the best fit for the task, judging by their ability to capture the long term dependencies without suffering from any vanishing gradients problems.

In 2015, Wu et al, used two-stream CNNs[3] to develop a hybrid learning framework[4] that can model several important aspects of the video data. They divided the videos into spatial and motion streams modeled by two CNNs separately. The spatial stream is sampled individual frames, which are useful for capturing the static information in videos like scene backgrounds and basic objects. Whereas, the motion stream is a set of stacked optical flow images that help capture short term temporal clues. The feature maps from both streams are forwarded to an RNN that uses LSTM memory cells to capture long-term temporal dependencies between frames. The same feature maps are also fused using Average pooling to generate video-level features. Finally, the outputs of the LSTM networks and video-level features are combined as the final predictions. Figure 2.5 illustrates the proposed hybrid framework.

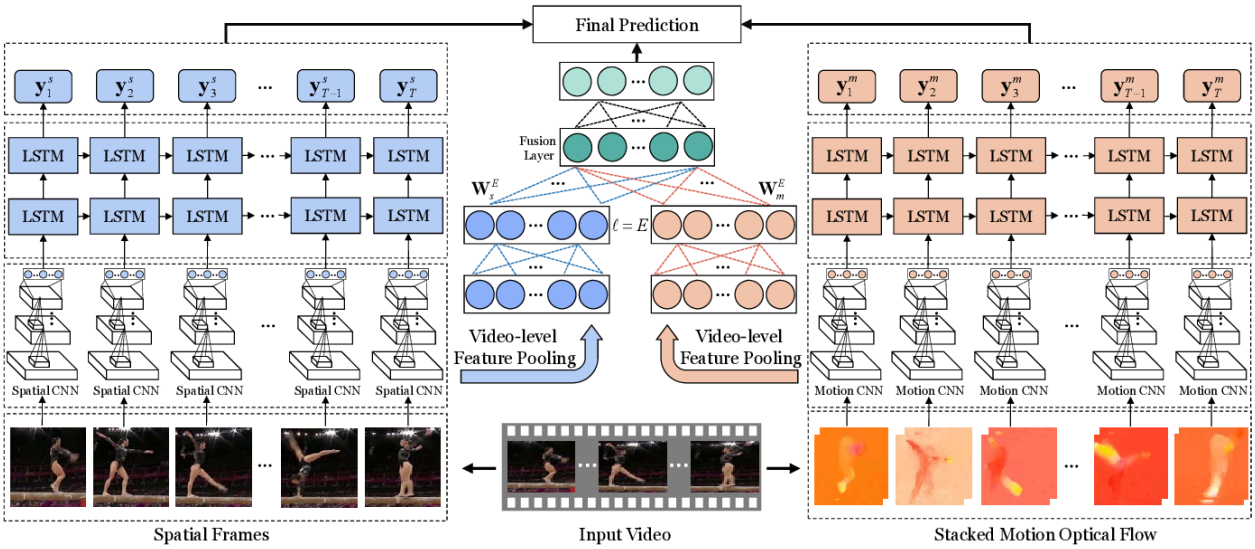


Figure 2.5: The proposed hybrid framework for video classification by Wu et al. in [4].

In the same year, Ng et al. also experimented on LSTMs in [5]. They proposed two approaches capable of handling full length videos, rather then just small clips.
In the first approach, they explored various temporal feature pooling neural network architectures. Applying temporal pooling directly as a layer enabled them to experiment with different locations of the pooling layer with respect to the network architecture.
In the second approach, they connected an RNN to the outputs of a GoogLeNet and an AlexNet networks which were fine tuned on video frames and optical flow images as well (Figure 2.6). The models where trained on the Youtube-1M and the UCF-101 datasets with up to two minute videos[6].
Ng et al. concluded from this work that incorporating information across longer video sequences enables better video classification. In addition they confirmed the necessity of optical flow images for obtaining good classification results.

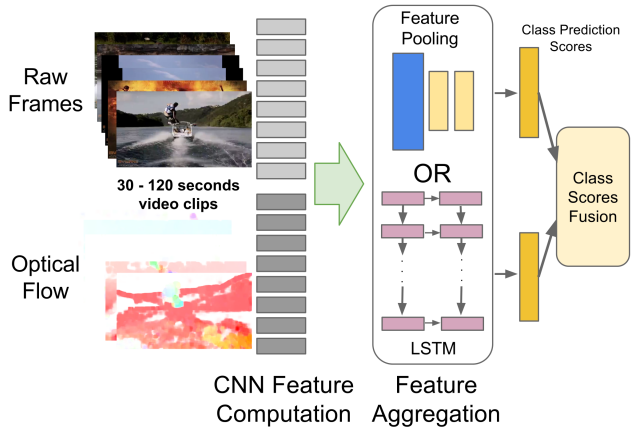


Figure 2.6: The proposed approach for video classification by Ng et al. in [5].

[6]Single frames from each minute of footage were sampled. Therefore a two minute video will give 120 frames. This trick helps with processing long videos.

In 2016, Wu et al. added auditory clues to the mix[101]. They applied the Short-Time Fourier Transformation to convert 1D soundtracks into 2D images (namely spectrograms), then trained standard CNNs on these audio spectrograms along with the other video aspects explored in [4]. Moreover, they proposed an effective fusion method to combine the outputs of individual networks which led to promising results. Multi modal feature extraction was also adopted recently in [102].

One year later, Yang et al. presented an interesting contribution[6]. They proposed to train RNNs on raw video frames from scratch, without any feature extractors being involved, aiming to take fully advantage of the RNN's ability to handle sequences of variable length. Training such a model turned out to be impractical due to the large input-to-hidden weight matrix that the input frames require.
To overcome this challenge, Yang et al. resorted to factorizing the matrix with the Tensor-Train Decomposition (TTD) technique[7] first introduced in [103]. The approach was inspired by [104] in which Tensor Train was applied to fully connected feed forward layers, so that they could consume raw image pixels[8]. In essence, Yang et al. used the Tensor-Train Factorization to formulate a so called Tensor-Train Layer[104]. Then, they replaced the weight matrix mapping from the input vector to the hidden layer in RNN models with the Tensor-Train Layer. This model was called the Tensor-Train Recurrent Neural Network (TT-RNN) . Figure 2.7 shows an overview of the approach throughout only 6 frames for illustrative purposes.

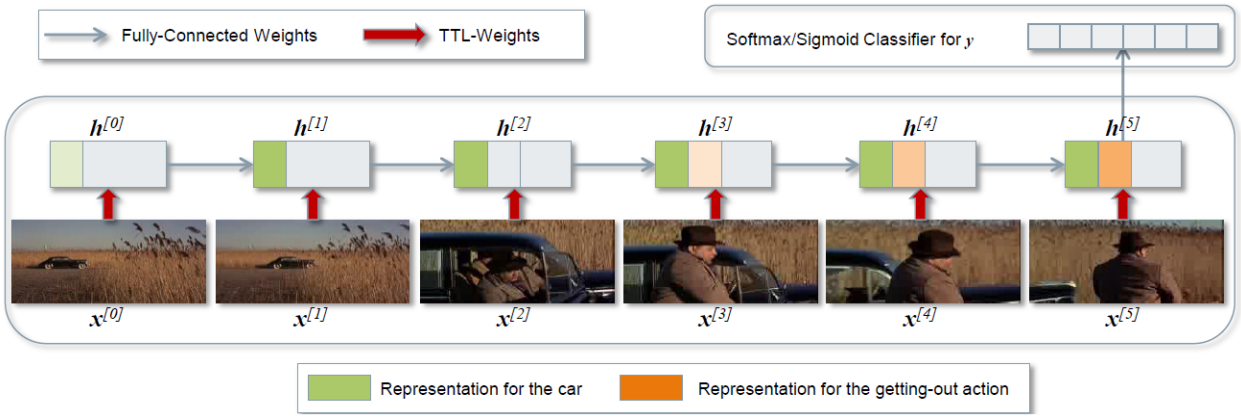


Figure 2.7: The Architecture of the proposed model based on TT-RNN by Yang et al. in [6].

The computational cost of video classification was an element of interest for Zhu et al. in 2019. They claimed that using the same feature extractor for all sampled clips[9] is computationally expensive. Thus, they proposed the FASTER framework[7], which aims

[7]Tensor-Train Decomposition is a method for representing a tensor compactly as factors and it allows to work with the tensor via its factors without materializing the tensor itself.

[8][104] showed that a fully connected layer can be reshaped into a high-dimensional tensor and then factorized using Tensor-Train. This was applied to compress very large weight matrices in deep neural networks where the entire model was trained from scratch.

[9]Videos are usually divided to small clips, and the predictions for these small clips are aggregated to give the video classification

to leverage the redundancy between neighboring clips[10] and reduce the computational cost by learning to aggregate the representations from models of different complexities using an innovative recurrent unit called FAST-GRU(see figure 2.8).

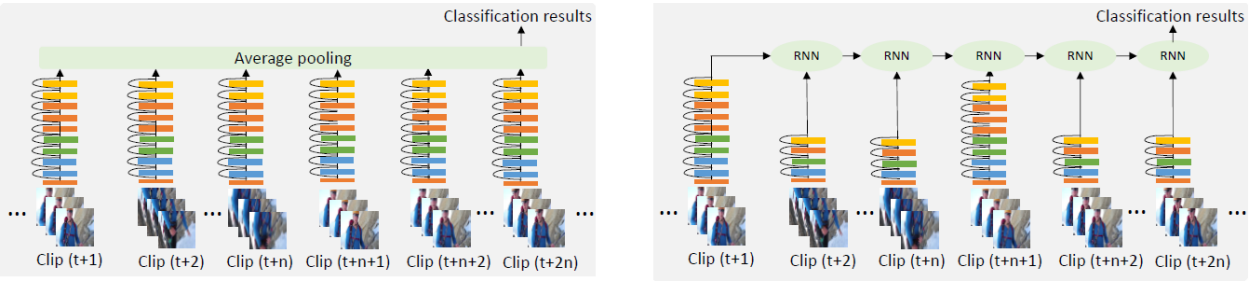


Figure 2.8: The difference between other video classification models (left) and the FASTER framework proposed in [7] (right): FASTER (right) exploits the combination of expensive and cheap networks and aggregates their results with the FAST-GRU unit.

In the same year, Bhardwaj et al. worked on efficient low cost video classification[105] by implementing two models. A Teacher model which computes a representation for the video by processing all frames in the video. And a Student model which is trained to process only a few frames but produce a similar representation to the one computed by the teacher. In other words, their goal was to process fewer frames and hence perform a lower number of Float Operations (FLOPs), all while maintaining the same performance.

2.7 Conclusion

In this chapter, We introduced the CV branch and highlighted the contribution of DL to it. Most importantly, we explored the details of the video classification task starting with its definition, challenges, motivations as well as the data availability for it. Finally we investigated and presented the biggest stat of the art methods tried by the CV community to tackle this problem.

In the next chapter we will experiment with DL based methods for video classification.

[10]Because adjacent clips tend to be semantically similar, Zhu et al. argued that it is computationally inefficient to process many clips close in time with a computationally expensive video model. So they used cheap models to cover scene changes and expensive models to capture subtle motion information.

Chapter 3

Implementation

3.1 Introduction

In this chapter, we will present the implementation of two different deep learning based models for video classification on the UCF-101 dataset. We will discuss the two approaches and briefly explain the software, hardware and the dataset we used for the experiment.

3.2 Architecture

Aiming to implement an adequate video classification model, we adopted two different methods. The first one is image based, meaning it classifies the frames captured from the video, each one separately. Only then, it predicts the general video class disregarding the temporal dependencies between frames. However the second method exploits these temporal dependencies using the learning ability of RNNs.
In the next subsections, we will explain each method in detail.

3.2.1 First Approach: CNN Frames Classification with Predictions Aggregation

The model implemented in the first approach was simple. It predicts the class of a video by capturing a collection of its frames, then passing them one by one through a trained CNN. The general class for the video is the class that was most repeated among the frames predictions.

Training the Video Classification Model In the training phase of the first model we went through three main steps. The first step is to extract some frames from every training video using the OpenCv [1] library, and save them to the disk along with their labels. In the second step we extract some frames from the testing set so we can use them as a validation set, while training the CNN. Finally, we design a CNN and train it on the labeled frames extracted earlier.

Sampled Training/Testing Frames From the training videos, and knowing that the frame rate for all videos is 25 FPS[84]. We sampled 8 frames from each 25 frames, which is equals to 8 sampled frames per each second of footage. This extraction rate was convenient enough to gather a reasonable number of training frames for the CNN[2], we were also limited by the storage space offered by Google Collab. For the testing videos we used the same frame extraction rate, and the frames extracted from the testing videos were used as validation data.

CNN's Architecture The CNN is the most important element in this model. Therefore it needs to be reliable at extracting distinct features that boost the classification quality. To acquire such a network, we had to use transfer learning. We

[1] https://opencv.org/

[2] Its important to note that the videos of the dataset cannot produce the same number of frames, as they are not of the same length. Thus some videos will produce more frames then others. Sampling 8 frames per second will enable the short videos to contribute to the training of the CNN more.

deployed an InceptionV3 network with weights pre-trained on ImageNet[3] as the base model, with excluding its last softmax layer. We appended the three layered FFNN shown in figure 3.1 and fine tuned its layers while freezing the InceptionV3 layers aiming to make the overall network more specific to our dataset.
The InceptionV3 network was sufficient at extracting features from the training frames so we weren't obliged to append a deeper FFNN. All we did was add a flattening layer to flatten the InceptionV3 feature maps, then add one fully connected layer to reduce the size of the flat feature maps and finally, add a softmax layer with 101 units.
Last but not least, in order to reduce overfitting we added two dropout layers between the three layers of the appended FFNN with a rate of 50%.
Figure 3.1 describes the architecture of the CNN used, Table 3.1 summarizes the details for the training process of the CNN, and figures 3.2, 3.3 and 3.4 display the plot of training accuracy/loss and validation accuracy/loss on splits 01, 02 and 03 respectively.

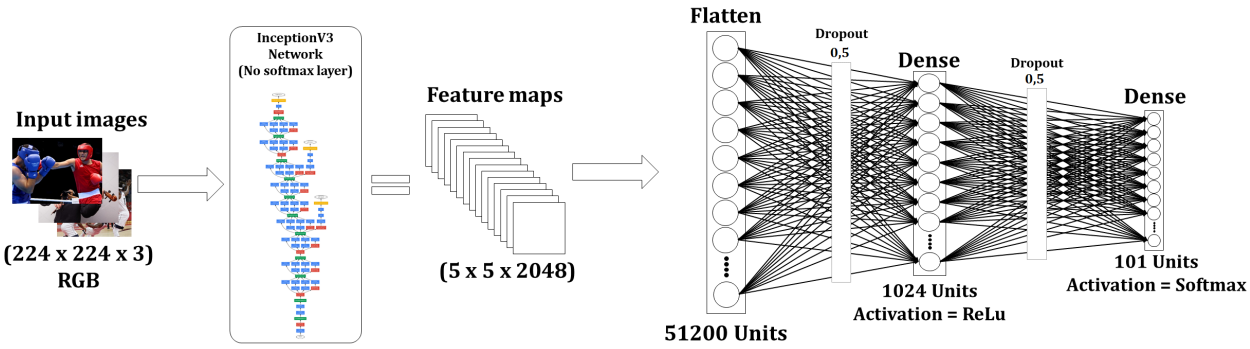


Figure 3.1: The architecture of the CNN used in the first approach

Table 3.1: Summary of the details for training the CNN in the first approach.

Total number of parameters	74,336,133
Number of trainable parameters	52,533,349
Number of non trainable parameters	21,802,784
Loss function	Categorical crossentropy
Optimizer	SGD(learning rate=0.001, momentum=0.6, nesterov=False)
Number of training frames (Splits 01, 02, 03)	598363, 599335, 597730
Number of validation frames (Splits 01, 02, 03)	233597, 232625, 234230
Number of epochs (Splits 01, 02, 03)	223, 226, 224
Total training duration in minutes (Splits 01, 02, 03)	96.68, 95.61, 96.24
Average epoch duration in seconds (Splits 01, 02, 03)	26.02, 25.38, 25.78

[3] http://www.image-net.org/

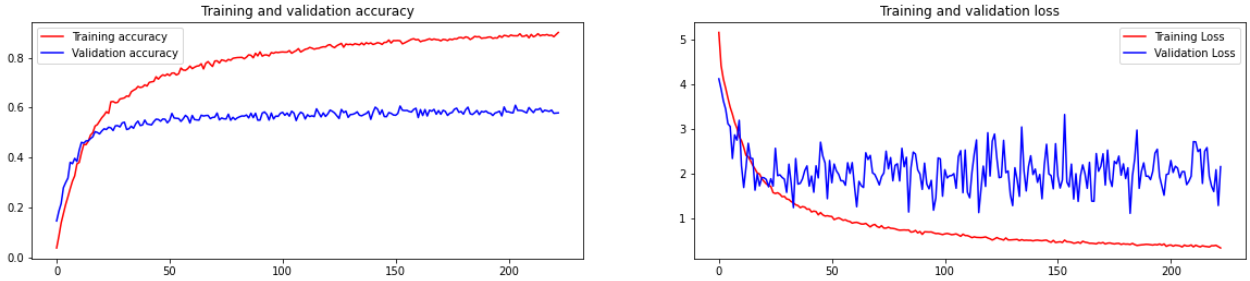


Figure 3.2: The training and validation accuracy plots (left) and the training and validation loss plots (right) while training the CNN on frames extracted from **split 01**

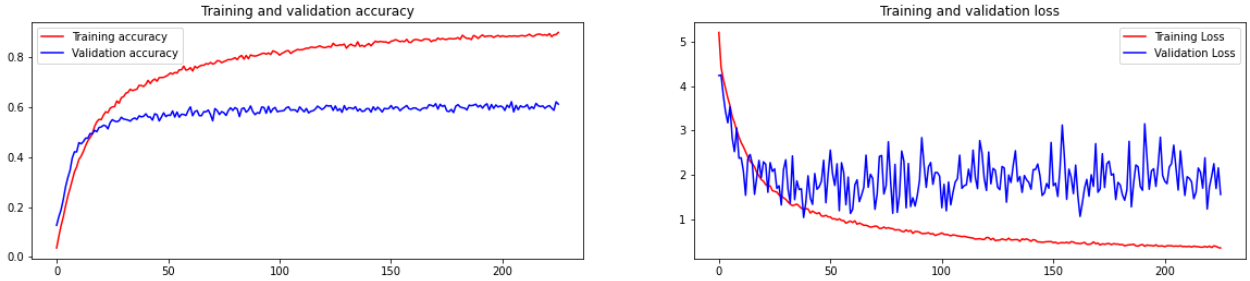


Figure 3.3: The training and validation accuracy plots (left) and the training and validation loss plots (right) while training the CNN on frames extracted from **split 02**

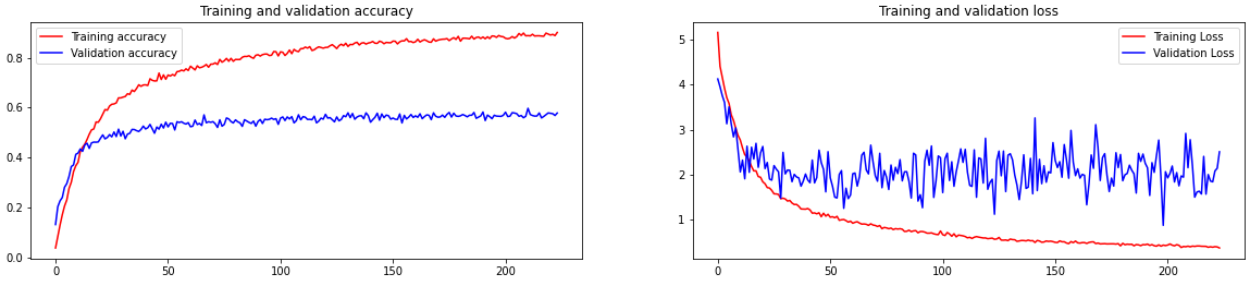


Figure 3.4: The training and validation accuracy plots (left) and the training and validation loss plots (right) while training the CNN on frames extracted from **split 03**

Testing the Video Classification Model Until this point, all we did was design a tool for image classification on the extracted frames, which is considered the first component of the video classification model. The second component is the aggregation function that determines the general class of the video starting with the predicted classes for its frames (Most repeated prediction).

Much like any other ML model, ours need to be tested. However, in order to truly evaluate the performance of our model we need to test it on two sets: Training videos

and testing videos. Because the training and testing videos were only involved in the training of the CNN and the accuracy achieved by the CNN on the frames does not reflect the video classifier's performance. It needs to be tested all over again as one unified block (CNN plus the aggregation function) on the training and testing videos.

Evaluate the Video Classifier on the Training Videos Measuring the video classifier's performance on the training video involved creating two empty lists, one is for real video labels and the other is for predicted video labels. We populated these lists by iterating through the training videos, and predicting the class of each video[4] along with saving its actual label, then appending each information to its corresponding list. Finally we computed the accuracy of the video classifier on the training set.

Evaluate the Video Classifier on the Testing Videos Measuring the video classifier's performance on the testing video followed the exact same procedure as the training set. Moreover, we saved the confusion matrix in addition to the testing accuracy.

3.2.2 Second Approach: CNN Feature Extraction with LSTM Prediction

For the second approach, we employed two DL tools, a CNN and an RNN. It predicts the class of a video by capturing some of its frames. Then, it uses a trained CNN to extract a feature vector from each sampled frame, and maintains their chronological order intact. The video becomes represented by a sequence of feature vectors. A sequence which is pre-processed and fed to a trained RNN that predicts its class.

Training the Video Classification Model Training such a model can be summarized in four steps. The first step is training a good CNN on the frames captured from the training videos. The second step is to prepare a training set for the RNN used later. This involves using the CNN to generate sequences of feature vectors from all training videos then pre-processing and labeling them as well. The last remaining step is to design and train an RNN on the prepared sequences.

CNN's Architecture Since we already implemented a CNN and trained it on the frames of the training videos, it would be pointless to repeat the process again. Therefore we rebuilt the same CNN used in the first approach and loaded its optimized weights. Moreover, we removed its last softmax and dropout layers, because we need the new CNN to output feature vectors, and not probability distributions. The architecture of the CNN used in this approach is described by the figure 3.5.

[4]We used the same frame extraction rate as in the training process, 8 sampled frames per each second of footage.

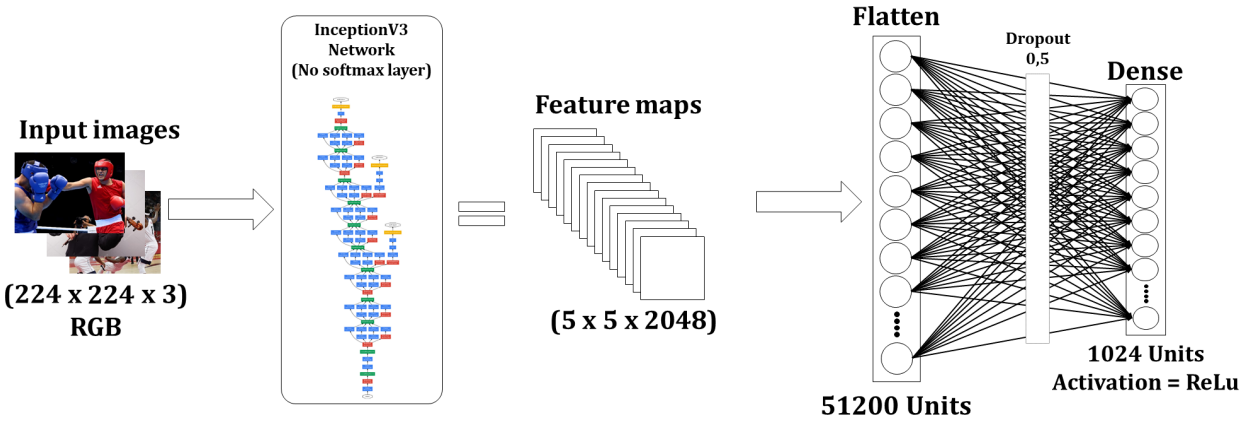


Figure 3.5: The architecture of the CNN used in the second approach

RNN's Training Data In order to gather proper training data for the RNN, from each video in the training set, we captured a number of frames (8 frames per second) and kept them in chronological order. Next, we extracted a feature vector from each captured frame using the edited CNN. This gave us a labeled sequence of feature vectors for each training video.
The fact that the training videos are not uniform in length[84], caused the training data for the RNN to have sequences of different time steps for different videos[5]. As a solution, we proposed a fixed number of times steps for all sequences. Accordingly, all the sequences in the training data for the RNN were either padded with vectors of zeros, or truncated to respect the uniform sequence length which was set to 80 feature vectors per sequence. By the end of this procedure, we had a training set composed of labeled sequences of length 80 time steps. Each time step is a feature vector which contains 1024 features.
Note that we used all the sequences extracted from the training videos as training data and no validation data was included during the RNN's training because the dataset comes with three training/test splits and doesn't include validation data. If we take away any percentage from the training split to use as validation data, that will severely effect the performance of our model because we are already dealing with a lack of training data problem. Plus we will not be able to compare our model to other models because the data used for training is not the same.

RNN's Architecture The used RNN is simple but yet very efficient (Figure 3.6). It is a two layered sequential network with an input shape of (80x1024). The first layer contains 1024 LSTM cells with the dropout parameter set to 20%. The output layer is a softmax layer with 101 units. A dropout of rate of 50% was applied between layers to reduce overfitting. Table 3.2 summarizes more details about the RNN training process, and figures 3.7, 3.8 and 3.9 display the plot of training accuracy/loss on splits 01, 02 and 03 respectively.

[5]This problem occurs because longer videos produce higher numbers of frames. Hence, longer sequences of feature vectors and vice versa.

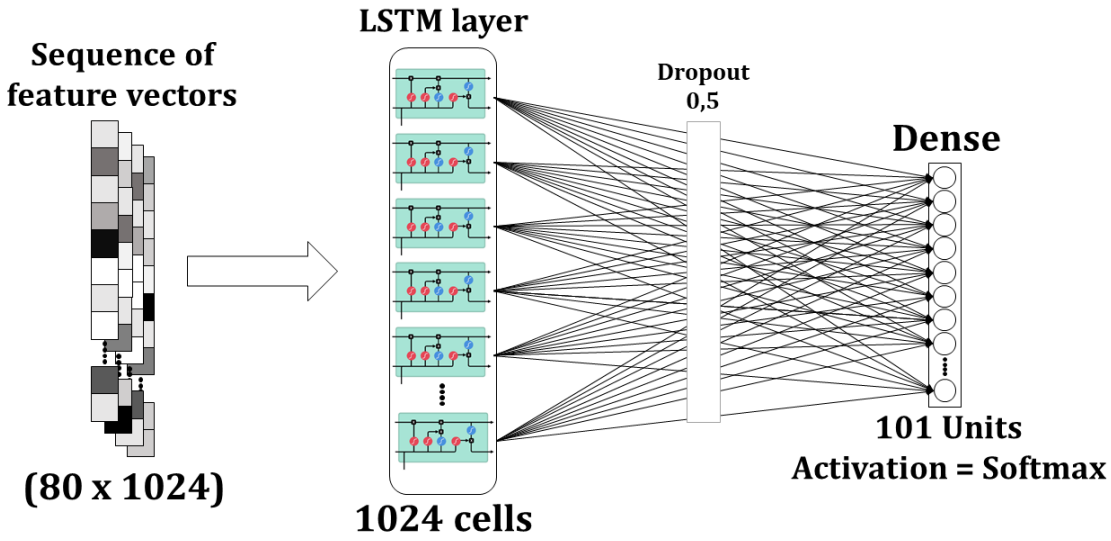


Figure 3.6: The architecture of the RNN used in the second approach for predicting the video class

Table 3.2: Summary of the details for training the RNN in the second approach.

Total number of parameters/trainable parameters	8,496,229
Loss function	Categorical crossentropy
Optimizer	Adam()
Number of training examples (Splits 01, 02, 03)	9537, 9586, 9624
Number of validation examples (Splits 01, 02, 03)	0, 0, 0
Number of epochs (Splits 01, 02, 03)	20, 21, 18
Total training duration in minutes (Splits 01, 02, 03)	10.47, 10.94, 07.27
Average epoch duration in seconds (Splits 01, 02, 03)	31,40, 31,27, 24,22

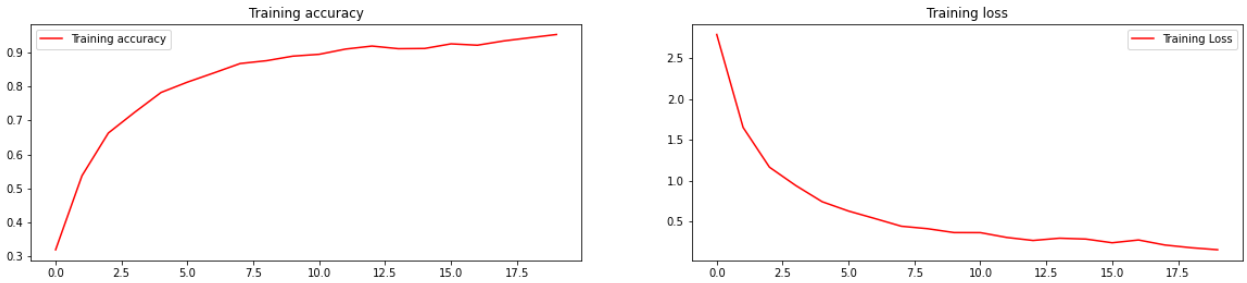


Figure 3.7: The training accuracy plot (left) and the training loss plot (right) while training the RNN on sequences from **split 01**

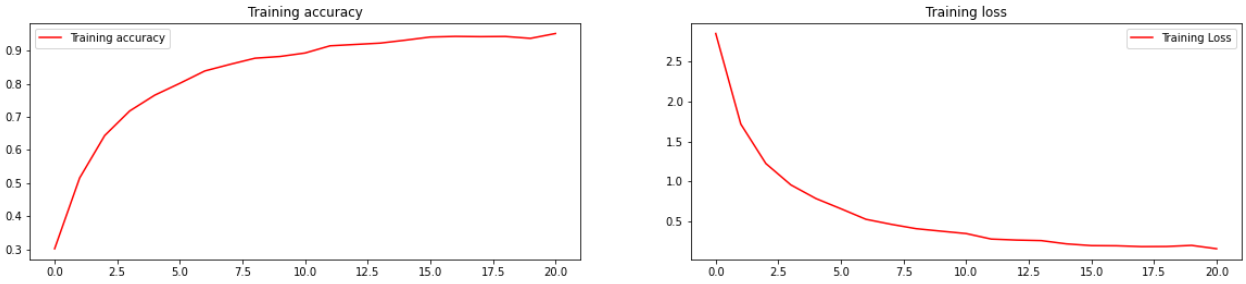


Figure 3.8: The training accuracy plot (left) and the training loss plot (right) while training the RNN on sequences from **split 02**

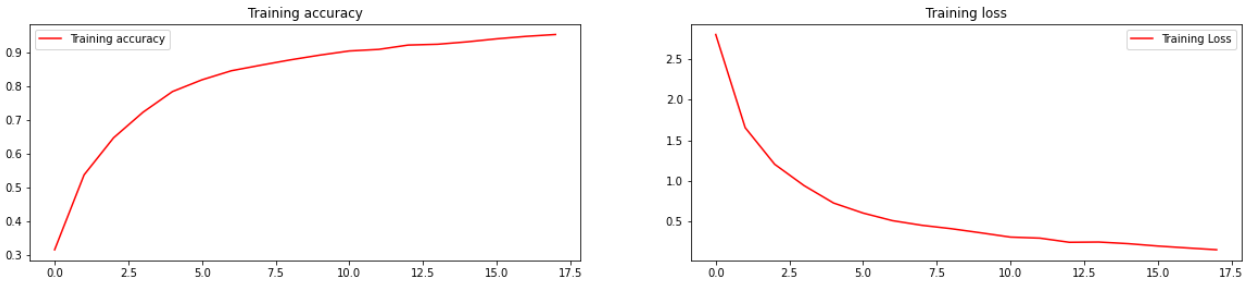


Figure 3.9: The training accuracy plot (left) and the training loss plot (right) while training the RNN on sequences from **split 03**

Testing the Video Classification Model Testing this model follows a simple procedure. First, we create two empty lists to keep track of the ground truth labels and the predicted labels. Next, we iterate through all the videos in the test set and encode each video as a sequence of feature vectors and set its length to 80 time steps either by padding or truncating it. We predict the class of the sequence by feeding it to the trained RNN. The two lists that we created are used to compute the accuracy of the classifier on the test set, compute the confusion matrix, and the classification report.

3.3 Software

3.3.1 Python

Python[6] is an interpreted, object-oriented, high-level programming language launched in december 1989 by Guido Van Rossum. Python is used in several domains like web development and creating software prototypes. But most importantly, the vast majority of AI and ML practitioners use it to implement their models due to its simplicity and consistency, plus the access to a great number of pre-implmented libraries and frameworks. These libraries make the coding part much easier for the developer. Granted, Python offers a platform independence which allows the developer to implement projects and run them on different platforms regardless of their type[7].

3.3.2 Tensorflow

TensorFlow[8] is an open source platform for machine learning, that consists of a myriad of tools, libraries and resources that enable researchers to push the state-of-the-art in ML and helps developers to easily build and deploy ML applications.
TensorFlow offers multiple levels of abstraction through high level APIs (Keras) as well as the lower-level library (Tensorflow) which provides more flexibility allowing developers to customize the operations implemented across the ML model. The high level APIs provided by Tensorflow make it so easy to implement ML models. Thus, any one can get familiar with implementing complex ML model very fast.
Tensorflow can be used on different platforms meaning you get to use ML in servers, edge devices, or the web not to mention its compatibility with other programming languages like Javascript.

Tensorflow was developed by the Google Brain team for internal Google use only. However, On November 2015 It was released as an open-source package under the Apache 2.0 license and made available online[106].

3.3.3 Keras

Keras[9] is a high-level API, which represents a simple interface, that minimizes the number of user actions required for common use cases.
Keras is easy to use and focused on user experience. It also makes it easier to run experiments, as it can easily scale up to large clusters of GPUs or entire TPU pods. One of its advantages is having the low-level flexibility to implement arbitrary research ideas but at the same time, it offers optional high-level convenient features to speed up experiment deployment, meaning

[6] https://www.python.org

[7] To read more about why is Python considered the best programming language for ML an DL, refer to the following web article: https://medium.com/towards-artificial-intelligence/why-is-pythons-programming-language-ideal-for-ai-and-data-science-e3f75a5d0e2b

[8] https://www.tensorflow.org

[9] https://keras.io

that all the tools that you might need are modules that can be called and fine tuned very easily with minimum amount of coding.

3.3.4 Google Colab

Google Colaboratory[10], or Colab for short, is a free cloud service offered by Google, for students, data scientists and AI researchers to write, run and share code, using executable documents called notebooks. In other words, it's a Jupyter notebook[11] environment that requires no setup to use and runs entirely in the cloud.
Colab provides a powerful virtual machine in which all the modules and libraries needed for ML and DL are already installed. Moreover, the machine comes with a reasonable package of hardware for free including a CPU, a RAM, a hard disk, and most importantly, the Graphical Processing Unit (GPU) and the Tensor Processing Units (TPU). Colab offers outstanding GPUs like the Nvidia Tesla K80, the NVIDIA Tesla P100 PCIe 16 GB and the NVIDIA Tesla T4 for the user to deploy and train ML models at ease. Furthermore the user can take advantage of the TPU technology for distributed training with 8 units available for implementing even more complex models and accelerating their training. That's not all, for more storage space the user gets to mount a google drive account to the virtual machine allocated. Therefore, the progress can be saved and this is a very useful feature for DL practitioners, considering the need to save the weights of a model for later use. Last but not least, Colab employs cells which contain either Python code, or formatted text. This feature enables the AI community to share ideas and experiences easily, because the code becomes well organized and comprehensible to anyone thanks to these text cells.
Figure 3.10 shows the Google Colab interface.

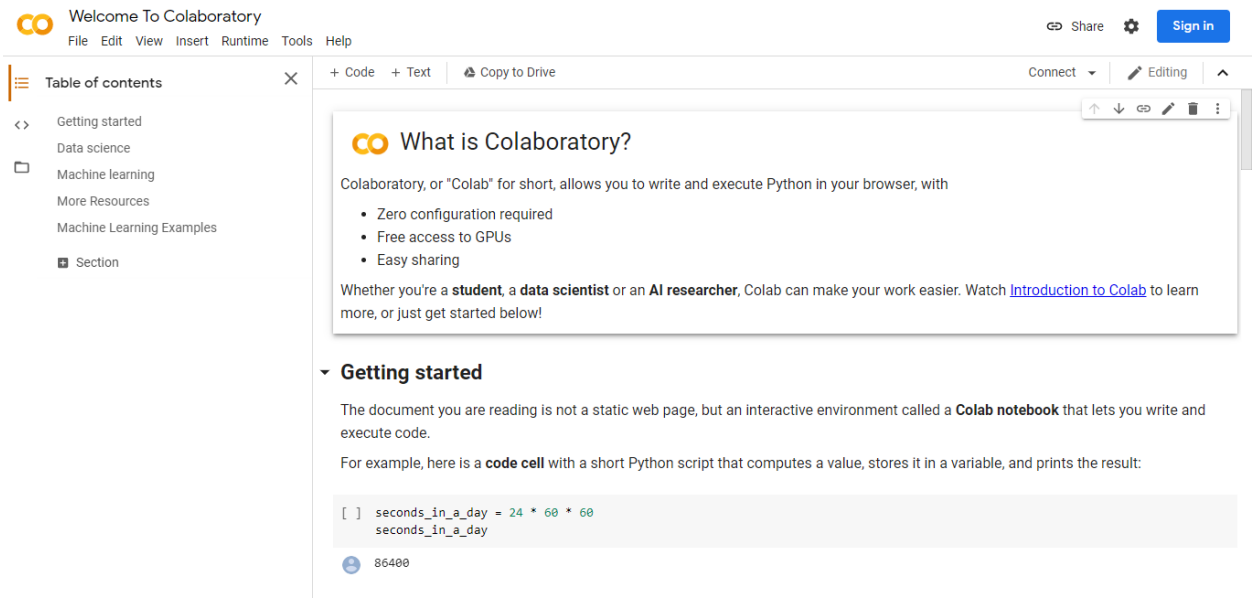


Figure 3.10: The interface of Google Colab.

[10] https://colab.research.google.com/notebooks/intro.ipynb
[11] https://jupyter.org/

3.4 Hardware

Throughout the whole implementation process, we used Google Colab to take advantage of the hardware it offers for free. The hardware we used is summarized in table 3.3

Table 3.3: The hardware used for the implementation.

CPU	Intel(R) Xeon(R) CPU @ 2.00GHz
GPU	Tesla P100-PCIE-16GB
RAM	25.51 GB
Disk	68.40 GB

3.5 Dataset

UCF-101[12] is a large dataset of human actions assembled and organized by Sommoro et al.[84] in 2012. It consists of 13320 labeled videos that belong to 101 action classes, making a total of 27 hours of video data. The dataset consists of realistic user-uploaded videos which contain obvious camera motion and cluttered backgrounds.
UCF-101 includes action classes which are divided into five types:

- Human-Object Interaction.
- Body-Motion Only.
- Human Human Interaction.
- Playing Musical Instruments.
- Sports.

More specifically, all UCF-101 videos are one of the classes shown in figure 3.11.

The clips from one action class are divided into 25 groups which contain 4 to 7 clips each. The clips in one group share some common features, such as the background or actors.
The dataset comes with three predefined training/test splits. However, if the user wants to create his own training/test splits, it is very important to keep the videos belonging to the same group separated in training and testing. Because the videos used for training that belong to the same group are similar and using them again for testing will yield a misleading high performance.

The zipped file of the dataset[13] includes 101 folders each containing the clips of one action class. The name of each clip has the following form: v_**X**_g **Y**_c**Z**.avi, where X, Y and Z represent the action class label, group and clip number respectively. For example, v_ApplyEyeMakeup_g03_c04.avi corresponds to the clip 4 of group 3 of action class ApplyEyeMakeup.

[12] https://www.crcv.ucf.edu/data/UCF101.php
[13] https://www.crcv.ucf.edu/data/UCF101/UCF101.rar

Figure 3.11 shows sampled frames from the 101 action classes in UCF-101, while table 3.4 summarizes the characteristics of the UCF-101 dataset.

Table 3.4: The characteristics of the UCF-101 dataset.

Actions	101
Clips	13320
Groups per action	25
Clips per group	4-7
Mean clip length	7.21 sec
Total duration	1600 min
Min clip length	1.06 sec
Max clip length	71.04 sec
Frame rate	25 fps
Resolution	320x240
Format	.avi
Audio	yes (51 action classes)

Figure 3.11: One sampled frame from each of the 101 action classes in UCF-101

3.5.1 Results And Discussion

1. **First Approach's Results**: First, let us lay down the best results achieved by the CNN during its training on frames classification. The whole training process on splits 01, 02 and 03 was already described earlier by the figures 3.2, 3.3 and 3.4 respectively. Table 3.5 summarizes the best accuracies/losses reached by the CNN during training (on splits 01, 02 and 03).

Table 3.5: The best results reached by the CNN during training on frames classification.

Image classification accuracy/loss	Split 01	Split 02	Split 03
Maximum training accuracy (%)	90.07	90.01	90.07
Minimum training loss	0.352	0.362	0.362
Maximum validation accuracy (%)	60.93	62.26	59.74
Minimum validation loss	1.124	1.047	0.867

The next result to be presented is the accuracy of the video classifier on the training set and the testing set of videos (splits 01, 02 and 03). It will be summarized by the table 3.6.

Table 3.6: The accuracy of the video classifier on the training set and the testing set of videos (splits 01, 02 and 03).

Video classification accuracy	Split 01	Split 02	Split 03
Accuracy on training videos (%)	78.60	79.19	79.60
Accuracy on testing videos (%)	57.41	56.08	57.01

We also saved the confusion matrix for the performance of the video classifier on the test set of each split, since it summarizes the prediction results of the model. It also helps us figure out if our model is confused between classes and gives us insight about the types of errors that are being made. Figures 3.12, 3.13 and 3.14 represent the confusion matrices for the video classifier's performance on the testing videos from splits 01, 02 and 03 respectively.

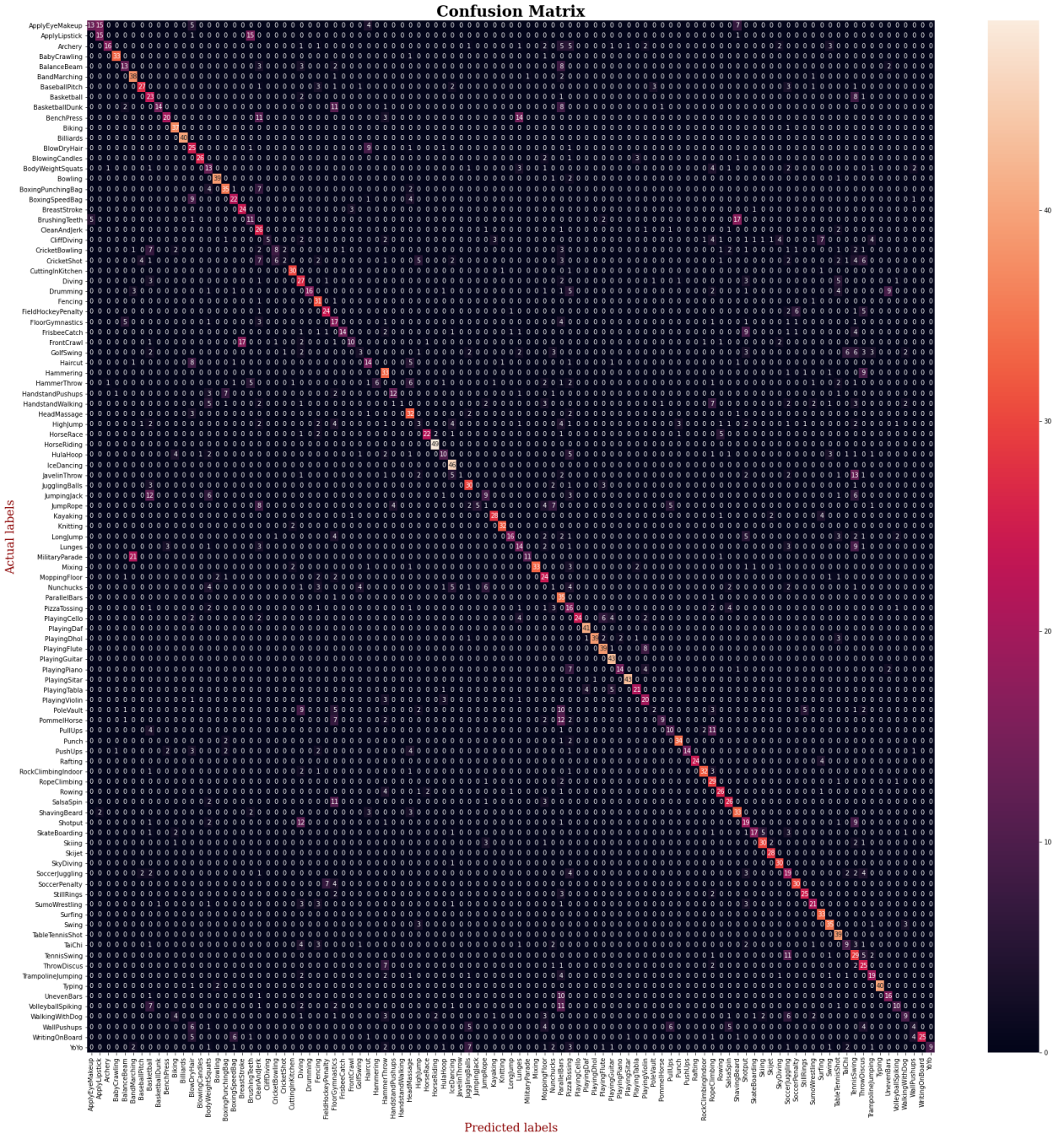


Figure 3.12: The confusion matrix for the first classifier's performance on the testing videos from **split 01**

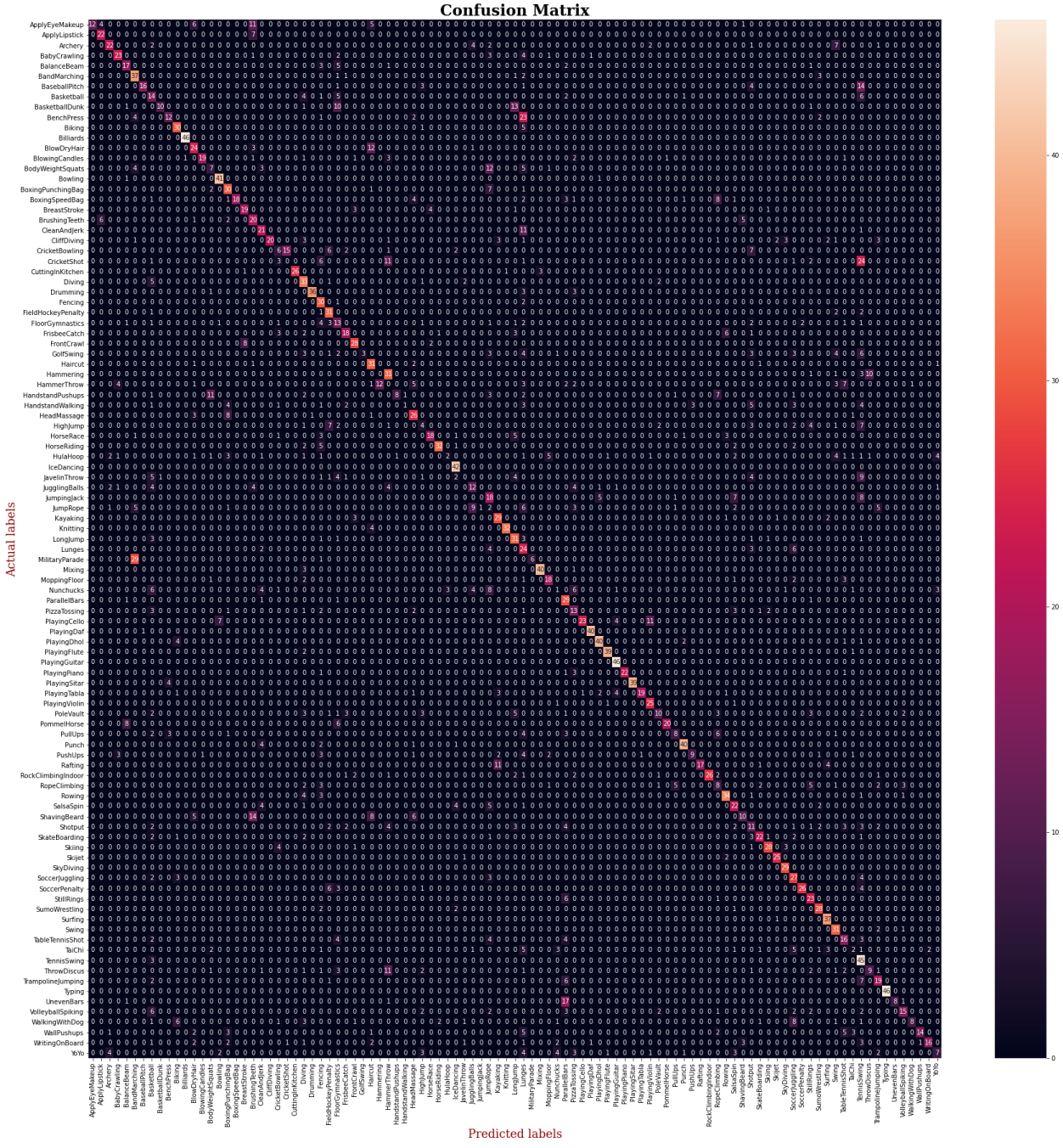


Figure 3.13: The confusion matrix for the first classifier's performance on the testing videos from **split 02**

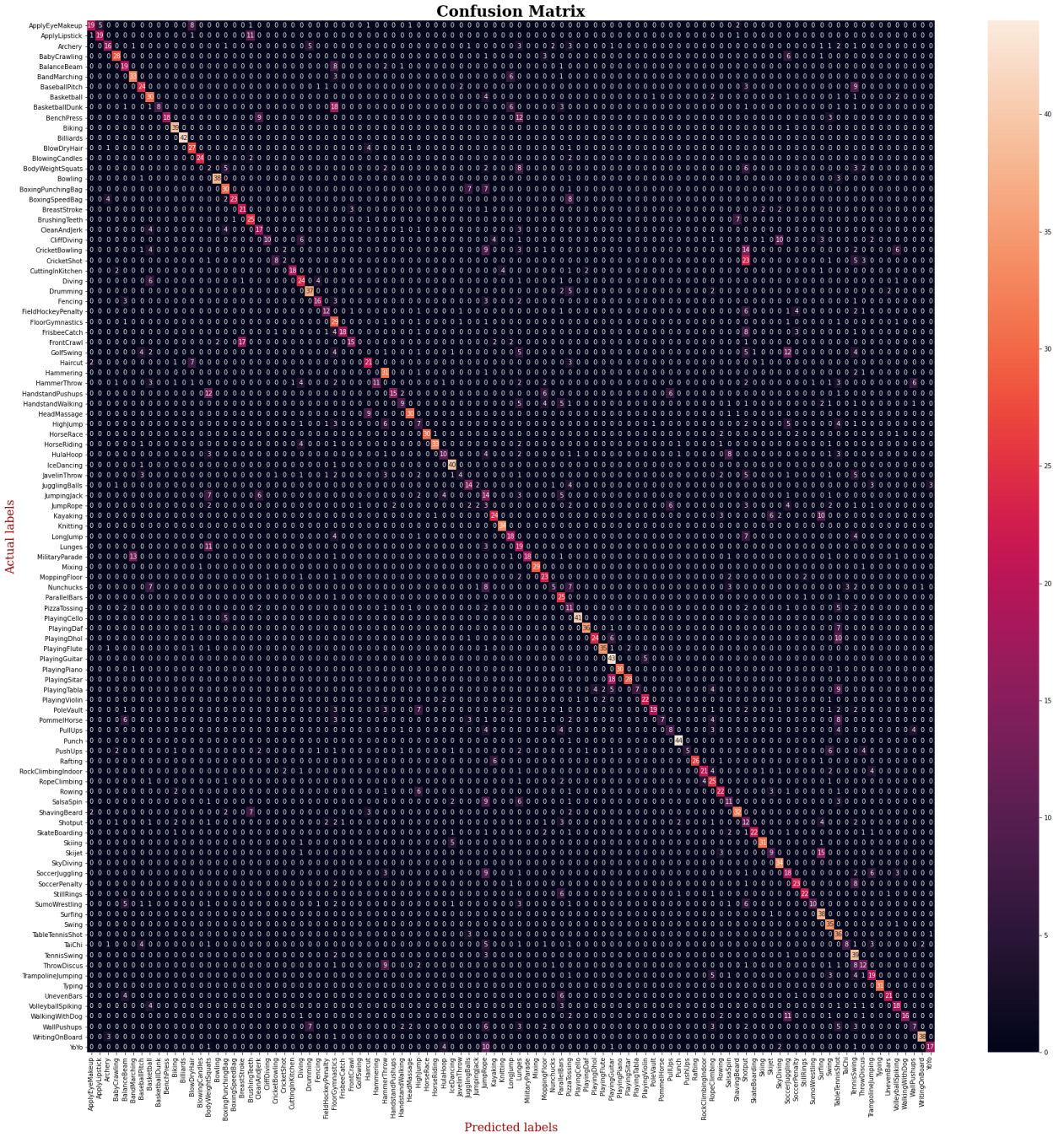


Figure 3.14: The confusion matrix for the first classifier's performance on the testing videos from **split 03**

2. **Second Approach's Results**: The first result to be presented is the best accuracy/loss that the RNN achieved while training on the feature sequences obtained from the training videos (on three splits). Table 3.7 summarizes that.

Table 3.7: The best training accuracy/loss reached by the RNN during training on the sequences encoded from the training videos.

Video classification accuracy/loss (training)	Split 01	Split 02	Split 03
Maximum training accuracy (%)	95.40	95.27	95.50
Minimum training loss	0.156	0.159	0.150

The accuracy of the video classifier on the testing set of videos (on splits 01, 02 and 03) is presented by the table 3.8.

Table 3.8: The accuracy of the video classifier on the testing set of videos (splits 01, 02 and 03)

Video classification accuracy (testing)	Split 01	Split 02	Split 03
Accuracy on testing videos (%)	66.27	66.87	65.40

Figures 3.15, 3.16 and 3.17 represent the confusion matrices for the classifier's performance on the testing videos on splits 01, 02 and 03 respectively.

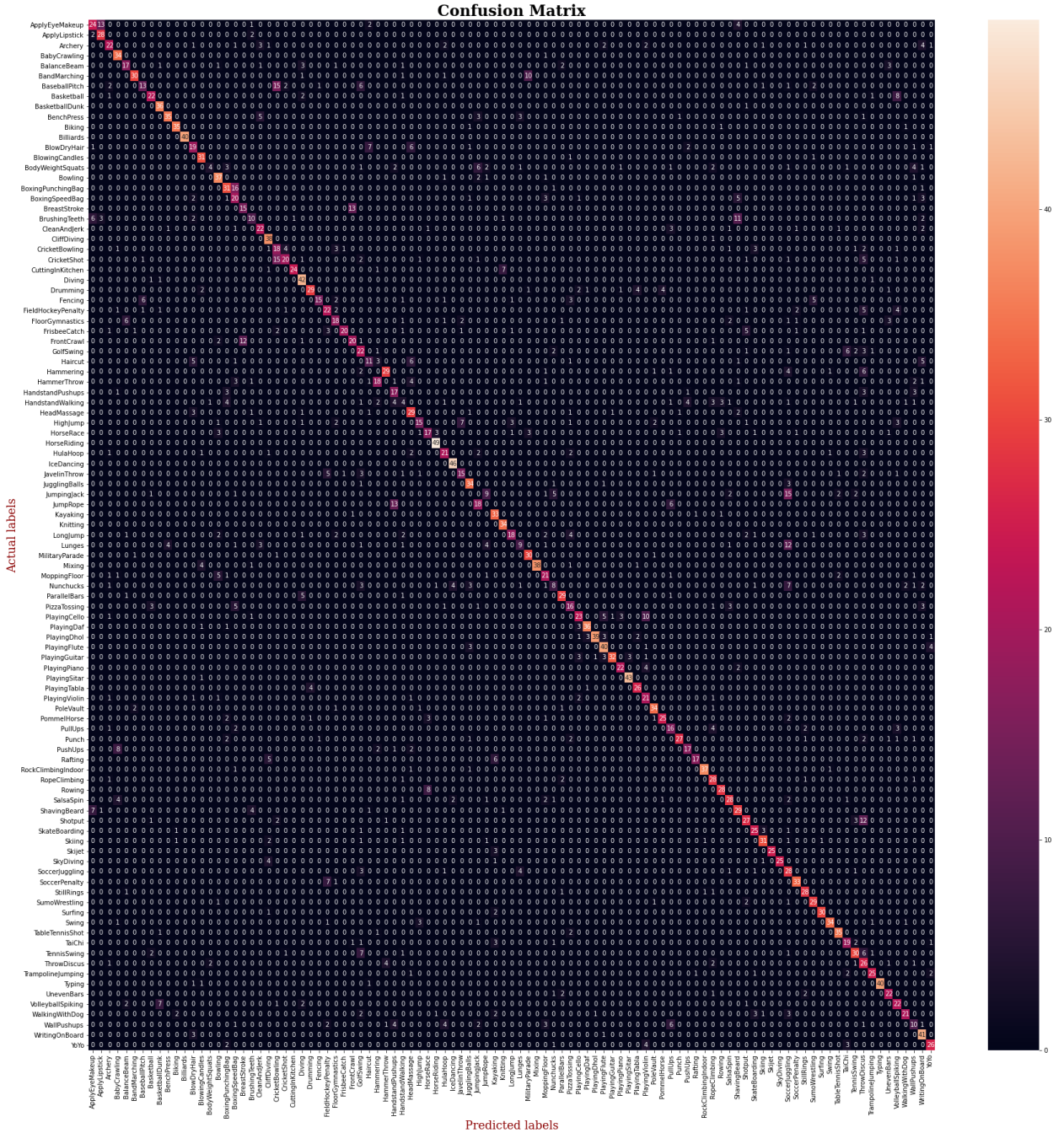


Figure 3.15: The confusion matrix for the second classifier's performance on the testing videos from **split 01**

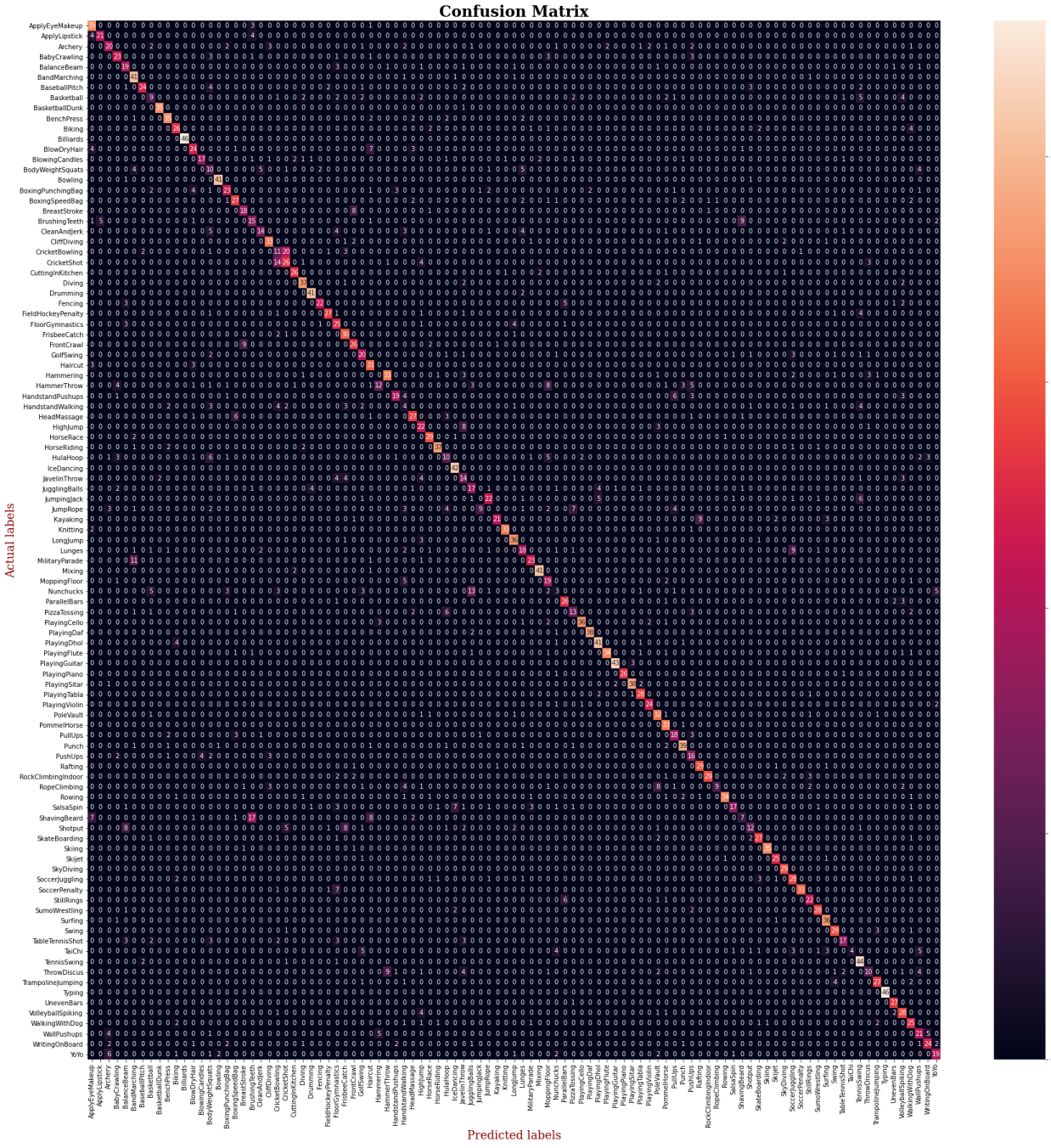


Figure 3.16: The confusion matrix for the second classifier's performance on the testing videos from **split 02**

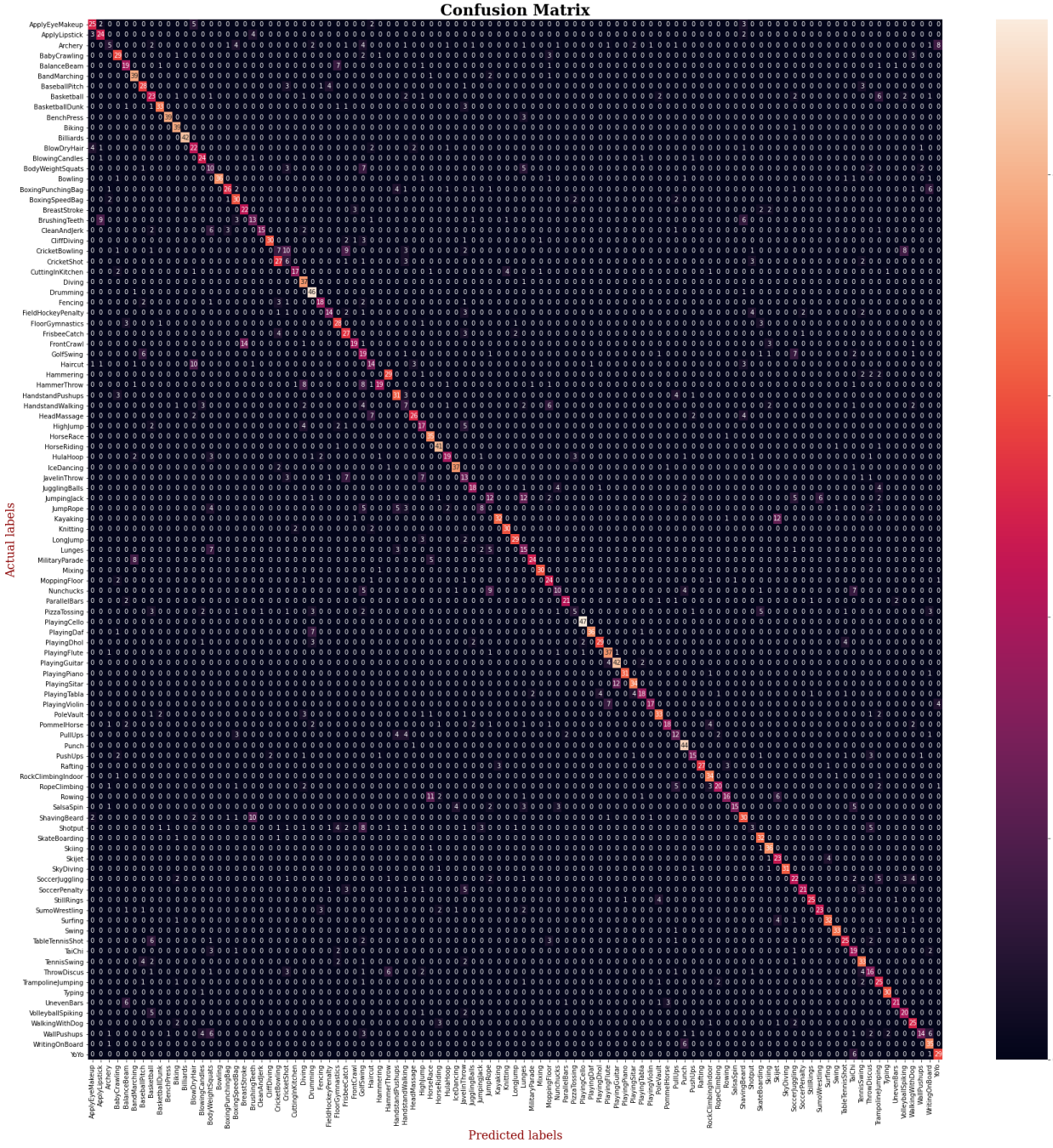


Figure 3.17: The confusion matrix for the second classifier's performance on the testing videos from **split 03**

In the first method, the CNN used for frame classification reached 90% accuracy on the training frames. However, on the testing frames it settled at 62% accuracy in the best scenario (split02). The CNN clearly overfitted, and that is mainly because the testing videos are somewhat different then the training videos when it comes to backgrounds, actors, camera motion etc, making it harder for the CNN to generalize the features learned from the training frames to the testing frames.

The video classification task using the first approach was achieved with an accuracy of 79% on the training videos and 57% on the testing videos. This performance was reasonable judging by the results reported by the used CNN for frame classification. This method also turned out to be a bit naive in the sense that the video classification quality is strictly dependent on the CNN's ability to learn from the frames, and that is a downside.

On the other hand, the model from the second approach achieved the best video classification accuracy on the training videos with 95%. Not only that, it improved the classification on the testing videos to become 66% accuracy. The reason behind this improvement is the use of the RNN who was able to learn the temporal dependencies present in the feature vectors extracted from the training videos. Furthermore, it used that knowledge on top of the CNN's experience to improve the accuracy of the classification on the testing videos. Table 3.9 represents a comparison between the results of our two experiments with other similar approachs.

Table 3.9: A comparison between the results of our two experiments with other similar approach.

Author	Technique	Testing accuracy
Sommoro et al.[84]	Harris3D corners/HOG/HOF descriptors + SVM	43.9%
Pulkit Sharma[107].	VGG16 + FFNN for frame classification + Frame voting	44.80%
HHTseng[108].	2D CNN (Trained from scratch) + LSTM	54.62%
Ours (first model)	InceptionV3 + FFNN for frame classification + Frame voting	57.41%
Ours (second model)	InceptionV3 feature extraction + LSTM	66.78%
Gokhan Cagrici[109].	InceptionV3 feature extraction + LSTM	72% - 73%
Matt Harvey[110].	InceptionV3 feature extraction + LSTM	74%

3.6 Conclusion

In this chapter, we implemented two DL based models for video classification. The best results were reported by the second model which took advantage of the CNN's ability to learn spatial features from the frames of a video in addition to the RNN's ability to capture the temporal features present in those frames. On the other hand the CNN alone, struggled with the video classification on the data at hand.

Conclusion

Conclusion

This research aims to explore the most convenient ANN types to implement a video classification model with the best performance possible. Accordingly, we presented the best candidate deep learning tools for the task, those being CNNs and RNNs. Furthermore, we explained video classification in detail and mentioned the state of the art methods for it.

Based on the experiments presented in the third chapter, we conclude that using a CNN for extracting spatial features from the frames of a video, along with capturing the temporal dependencies using an LTSM is best for the classification. On the other hand, neglecting the relationship between frames and treating the video as a collection of separate frames is a naive idea and it doesn't report the best results.
We also conclude that sufficient training data is definitely a key factor for executing efficient video classification.

Much like any other task in CV, training time was a bit long. However, that was a problem we could work around thanks the hardware offered by Google Collab. Besides that challenge, there is a lot of data pre-processing that goes into training a video classifier, combined with the need for reasonable storage space.

As perspectives, it will be important for our future works to test the approaches implemented here on bigger datasets like the Youtube-1M dataset and see what kind of improvements would it have on the model's performance.

Bibliography

[1] Shengxin Zha, Florian Luisier, Walter Andrews, Nitish Srivastava, and Ruslan Salakhutdinov. Exploiting image-trained cnn architectures for unconstrained video classification. *arXiv preprint arXiv:1503.04144*, 2015.

[2] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[3] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[4] Zuxuan Wu, Xi Wang, Yu-Gang Jiang, Hao Ye, and Xiangyang Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 461–470, 2015.

[5] Joe Yue-Hei Ng, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga, and George Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4694–4702, 2015.

[6] Yinchong Yang, Denis Krompass, and Volker Tresp. Tensor-train recurrent neural networks for video classification. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3891–3900. JMLR. org, 2017.

[7] Linchao Zhu, Laura Sevilla-Lara, Du Tran, Matt Feiszli, Yi Yang, and Hong xiu Wang. Faster recurrent networks for video classification. *ArXiv*, abs/1906.04226, 2019.

[8] Richard Bellman. *An introduction to artificial intelligence: Can computers think?* Thomson Course Technology, 1978.

[9] Alan M Turing. Computing machinery and intelligence. In *Parsing the Turing Test*, pages 23–65. Springer, 2009.

[10] Arthur L Samuel. Some studies in machine learning using the game of checkers. *IBM Journal of research and development*, 3(3):210–229, 1959.

[11] Thomas M. Mitchell. *Machine Learning*. McGraw-Hill, Inc., USA, 1 edition, 1997.

[12] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[13] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemysław Dębiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680*, 2019.

[14] Tomáš Mikolov, Anoop Deoras, Daniel Povey, Lukáš Burget, and Jan Černockỳ. Strategies for training large scale neural network language models. In *2011 IEEE Workshop on Automatic Speech Recognition & Understanding*, pages 196–201. IEEE, 2011.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[16] Antoine Bordes, Sumit Chopra, and Jason Weston. Question answering with subgraph embeddings. *arXiv preprint arXiv:1406.3676*, 2014.

[17] Sébastien Jean, Kyunghyun Cho, Roland Memisevic, and Yoshua Bengio. On using very large target vocabulary for neural machine translation. *arXiv preprint arXiv:1412.2007*, 2014.

[18] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[19] Frank Rosenblatt. *The perceptron, a perceiving and recognizing automaton Project Para*. Cornell Aeronautical Laboratory, 1957.

[20] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, 1943.

[21] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*, 2015.

[22] Wenling Shang, Kihyuk Sohn, Diogo Almeida, and Honglak Lee. Understanding and improving convolutional neural networks via concatenated rectified linear units. In *international conference on machine learning*, pages 2217–2225, 2016.

[23] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

[24] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, 2019.

[25] Hossam Faris, Ibrahim Aljarah, and Seyedali Mirjalili. Evolving radial basis function networks using moth–flame optimizer. In *Handbook of neural computation*, pages 537–550. Elsevier, 2017.

[26] Teuvo Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990.

[27] Umut Asan and Secil Ercan. *An Introduction to Self-Organizing Maps*, pages 295–315. Atlantis Press, Paris, 2012.

[28] Geoffrey E Hinton. A practical guide to training restricted boltzmann machines. In *Neural networks: Tricks of the trade*, pages 599–619. Springer, 2012.

[29] Geoffrey Hinton. Boltzmann machine. *Scholarpedia*, 2(5):1668, 2007.

[30] Geoffrey E Hinton and Ruslan R Salakhutdinov. Reducing the dimensionality of data with neural networks. *science*, 313(5786):504–507, 2006.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[32] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[33] David H Hubel. Single unit activity in striate cortex of unrestrained cats. *The Journal of physiology*, 147(2):226–238, 1959.

[34] David H Hubel and Torsten N Wiesel. Receptive fields of single neurones in the cat's striate cortex. *The Journal of physiology*, 148(3):574–591, 1959.

[35] Nikhil Buduma and Nicholas Locascio. *Fundamentals of deep learning: Designing next-generation machine intelligence algorithms*. " O'Reilly Media, Inc.", 2017.

[36] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

[37] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[38] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[39] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[40] John J Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the national academy of sciences*, 79(8):2554–2558, 1982.

[41] Sepp Hochreiter. Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, 91(1), 1991.

[42] Yoshua Bengio, Patrice Simard, and Paolo Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[43] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International conference on machine learning*, pages 1310–1318, 2013.

[44] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[45] Felix A Gers and Jürgen Schmidhuber. Recurrent nets that time and count. In *Proceedings of the IEEE-INNS-ENNS International Joint Conference on Neural Networks. IJCNN 2000. Neural Computing: New Challenges and Perspectives for the New Millennium*, volume 3, pages 189–194. IEEE, 2000.

[46] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[47] Kaisheng Yao, Trevor Cohn, Katerina Vylomova, Kevin Duh, and Chris Dyer. Depth-gated lstm. *arXiv preprint arXiv:1508.03790*, 2015.

[48] Jan Koutnik, Klaus Greff, Faustino Gomez, and Juergen Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

[49] Klaus Greff, Rupesh K Srivastava, Jan Koutník, Bas R Steunebrink, and Jürgen Schmidhuber. Lstm: A search space odyssey. *IEEE transactions on neural networks and learning systems*, 28(10):2222–2232, 2016.

[50] Naoki. Demystifying cross entropy. https://naokishibuya.medium.com/demystifying-cross-entropy-e80e3ad54a8, 28-10-2018. Accessed: 2020-06-08.

[51] Andrew Ng. Cost function - logistic regression. https://www.coursera.org/lecture/machine-learning/cost-function-1XG8. Accessed: 2020-06-08.

[52] Andrew Ng. Simplified cost function and gradient descent logistic regression. https://www.coursera.org/lecture/machine-learning/simplified-cost-function-and-gradient-descent-MtEaZ. Accessed: 2020-06-08.

[53] Kevin P. Murphy. *Machine learning: a probabilistic perspective*. Adaptive computation and machine learning series. MIT Press, Cambridge, MA, 2012.

[54] Mean squared error loss function. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-squared-error. Accessed: 2020-06-08.

[55] Mean absolute error loss function. https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/mean-absolute-error. Accessed: 2020-06-08.

[56] Sebastian Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[57] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of machine learning research*, 12(Jul):2121–2159, 2011.

[58] Matthew D Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[59] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[60] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural networks*, 12(1):145–151, 1999.

[61] Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[62] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.

[63] Song Han, Jeff Pool, Sharan Narang, Huizi Mao, Enhao Gong, Shijian Tang, Erich Elsen, Peter Vajda, Manohar Paluri, John Tran, et al. Dsd: Dense-sparse-dense training for deep neural networks. *arXiv preprint arXiv:1607.04381*, 2016.

[64] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[65] Dana Harry Ballard and Christopher M Brown. *Computer vision*. Prentice Hall, 1982.

[66] Emanuele Trucco and Alessandro Verri. *Introductory techniques for 3-D computer vision*, volume 201. Prentice Hall Englewood Cliffs, 1998.

[67] Linda G Shapiro and George C Stockman. *Computer vision*. Prentice Hall, 2001.

[68] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[69] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[70] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[71] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[72] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[73] Jifeng Dai, Yi Li, Kaiming He, and Jian Sun. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in neural information processing systems*, pages 379–387, 2016.

[74] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[75] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[76] Guosheng Lin, Anton Milan, Chunhua Shen, and Ian Reid. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1925–1934, 2017.

[77] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *arXiv preprint arXiv:1412.7062*, 2014.

[78] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[79] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, volume 1, pages 886–893. IEEE, 2005.

[80] AJ Piergiovanni, Chenyou Fan, and Michael S Ryoo. Learning latent sub-events in activity videos using temporal attention filters. *arXiv preprint arXiv:1605.08140*, 2016.

[81] Alok Singh, Thoudam Doren Singh, and Sivaji Bandyopadhyay. Nits-vc system for vatex video captioning challenge 2020. *arXiv preprint arXiv:2006.04058*, 2020.

[82] Nevenka Dimitrova, Lalitha Agnihotri, Mauro Barbieri, and Hans Weda. *Video Segmentation*, pages 3308–3313. Springer US, Boston, MA, 2009.

[83] Sami Abu-El-Haija, Nisarg Kothari, Joonseok Lee, Paul Natsev, George Toderici, Balakrishnan Varadarajan, and Sudheendra Vijayanarasimhan. Youtube-8m: A large-scale video classification benchmark. *arXiv preprint arXiv:1609.08675*, 2016.

[84] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[85] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 357–360, 2007.

[86] Geert Willems, Tinne Tuytelaars, and Luc Van Gool. An efficient dense and scale-invariant spatio-temporal interest point detector. In *European conference on computer vision*, pages 650–663. Springer, 2008.

[87] A. Klaeser, M. Marszalek, and C. Schmid. A spatio-temporal descriptor based on 3d-gradients. In *Proceedings of the British Machine Vision Conference*, pages 99.1–99.10. BMVA Press, 2008. doi:10.5244/C.22.99.

[88] Heng Wang, Alexander Kläser, Cordelia Schmid, and Cheng-Lin Liu. Dense trajectories and motion boundary descriptors for action recognition. *International journal of computer vision*, 103(1):60–79, 2013.

[89] Heng Wang and Cordelia Schmid. Action recognition with improved trajectories. In *Proceedings of the IEEE international conference on computer vision*, pages 3551–3558, 2013.

[90] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In *Scandinavian conference on Image analysis*, pages 363–370. Springer, 2003.

[91] Basura Fernando and Stephen Gould. Learning end-to-end video classification with rank-pooling. In *ICML*, 2016.

[92] Antoine Miech, Ivan Laptev, and Josef Sivic. Learnable pooling with context gating for video classification. *arXiv preprint arXiv:1706.06905*, 2017.

[93] Relja Arandjelović, Petr Gronát, Akihiko Torii, Tomás Pajdla, and Josef Sivic. Netvlad: Cnn architecture for weakly supervised place recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5297–5307, 2016.

[94] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. Aggregating local descriptors into a compact image representation. *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3304–3311, 2010.

[95] Zein Al Abidin Ibrahim, Marwa Saab, and Ihab Sbeity. Videotovecs: a new video representation based on deep learning techniques for video classification and clustering. *SN Applied Sciences*, 1:1–7, 2019.

[96] Ji Zhang, Kuizhi Mei, Yu Zheng, and J. Ho-Yin Fan. Exploiting mid-level semantics for large-scale complex video classification. *IEEE Transactions on Multimedia*, 21:2518–2530, 2019.

[97] Du Tran, Lubomir D. Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. C3d: Generic features for video analysis. *ArXiv*, abs/1412.0767, 2014.

[98] Du Tran, Heng Wang, Lorenzo Torresani, and Matt Feiszli. Video classification with channel-separated convolutional networks. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[99] Hao Ye, Zuxuan Wu, Rui-Wei Zhao, Xi Wang, Yu-Gang Jiang, and Xiangyang Xue. Evaluating two-stream cnn for video classification. In *Proceedings of the 5th ACM on International Conference on Multimedia Retrieval*, pages 435–442, 2015.

[100] De Xie, Cheng Deng, Hao Wang, C. C. Li, and Dapeng Tao. Semantic adversarial network with multi-scale pyramid attention for video classification. In *AAAI*, 2019.

[101] Zuxuan Wu, Yu-Gang Jiang, Xi Wang, Hao Ye, Xiangyang Xue, and Jun Wang. Fusing multi-stream deep networks for video classification. *arXiv preprint arXiv:1509.06086*, 2015.

[102] Haiman Tian, Yudong Tao, Samira Pouyanfar, Shu-Ching Chen, and Mei-Ling Shyu. Multimodal deep representation learning for video classification. *World Wide Web*, 22(3):1325–1341, 2019.

[103] Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5):2295–2317, 2011.

[104] Alexander Novikov, Dmitrii Podoprikhin, Anton Osokin, and Dmitry P Vetrov. Tensorizing neural networks. In *Advances in neural information processing systems*, pages 442–450, 2015.

[105] Shweta Bhardwaj, Mukundhan Srinivasan, and Mitesh M Khapra. Efficient video classification using fewer frames. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 354–363, 2019.

[106] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin

Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[107] Pulkit Sharma. Step-by-Step Deep Learning Tutorial to Build your own Video Classification Model. https://medium.com/analytics-vidhya/step-by-step-deep-learning-tutorial-to-build-your-own-video-classification-model-2 September 2019. Accessed: 2020-07-06.

[108] HTseng. HHTseng/video-classification. https://github.com/HHTseng/video-classification, July 2020. Accessed: 2020-07-06.

[109] Gokhan Cagrici. Video Classification with Deep Learning. https://medium.com/@gcagrici/video-classification-with-deep-learning-3840b20b7949, March 2019. Accessed: 2020-07-06.

[110] Matt Harvey. Five video classification methods implemented in Keras and TensorFlow. https://blog.coast.ai/five-video-classification-methods-implemented-in-keras-and-tensorflow-99cad29cc0b5 September 2017. Accessed: 2020-07-06.