UNIVERSITÀ DEGLI STUDI DI
MODENA E REGGIO EMILIA
Dipartimento di Ingegneria “Enzo Ferrari”

Corso di Laurea Magistrale in Ingegneria Informatica

Action recognition to estimate Activities of Daily Living (ADL) of elderly people

Relatore:
Prof.ssa Sonia Bergamaschi

Candidato:
Matteo Gabrielli

Correlatore:
Pietro Leo

Anno Accademico 2017 - 2018

Keywords:

Action recognition
Prediction models
Transfer learning
CNN
Video dataset

Table of contents

Abstract

This thesis project aimed to design and develop an AI system able to recognize the development of certain actions in video about residential internal context carried out by elderly people.

This system allows to estimate several parameters of two domains, specifically Activities of Daily Living and the Instrumental Activities of Daily Living, that form the Multidimensional Prognostic Index (MPI), a predictive index of mortality not linked to a specific disease, widely accepted and validated at international level and built on data obtained through a Multidimensional Valuation (VMD) of the elderly subject.

The proposed solution involves the creation of neural network models, in particular Convolutional Neural Networks that are trained on data obtained from the "Moments in Time" dataset, which includes a collection of one million labeled 3 second videos. Moments is a research project in development by the MIT-IBM Watson AI Lab.

Abstract in Italian

In questa tesi, viene progettato e sviluppato un sistema di riconoscimento di determinate azioni in filmati acquisiti all'interno del contesto abitativo. Questo sistema permette di stimare diversi parametri che compongono due dei domini che formano il MPI: l'Activities of Daily Living e l'Instrumental Activities of Daily Living.

L'MPI (Indice Prognostico Multidimensionale) è un indice predittivo di mortalità non legato ad una malattia specifica, ampiamente accettato e validato a livello internazionale e costruito su dati ottenuti mediante una Valutazione Multidimensionale (VMD) del soggetto anziano.

La soluzione proposta vede la realizzazione di modelli di reti neurali, in particolare di Convolutional Neural Networks che vengono allenate su dati ricavati dal dataset "Moments in Time", il quale contiene 1 milione di video di azioni classificate di 3 secondi. Moments è un progetto di ricerca supportato dal MIT-IBM Watson AI Lab.

Acknowledgements

I would like to offer special thanks to my family and friends for their support, encouragement and friendship throughout my studies and my life.

I owe special thanks to my friend Federico Campo, for his useful and constructive comments and advice in several choices I made in my life and for the support we have given ourselves throughout our studies.

The realization of this thesis was an opportunity provided by the IBM Italy team during my internship, so I would like to thanks the whole team and in particular my tutors Silvia Peschiera and Marco De Ieso, as well as the co-advisor Pietro Leo and the Director of Research Innovation & Technology for IBM Italia, Fabrizio Renzi.

I would like to express my deep gratitude to my advisor Professor Sonia Bergamaschi and Giovanni Morrone for the valuable advices.

Finally, I would like to extend my gratitude to Daniele Sancarlo and Francesco Giuliani, part of “IRCCS Casa Sollievo della Sofferenza”, who provided the use case that was taken into consideration in order to do this master thesis.

Thank you!

1 Introduction

Because of the rising aging of population and overstretched healthcare resources, technology-driven healthcare delivery to support independent living has attracted increasing amounts of attention both from research as well as from the business perspective.

In this context, the concept of Smart Home emerged as a viable mainstream approach to achieve this goal. A Smart Home system is a residential home setting augmented with a diversity of multimodal sensors, actuators, and devices along with Information and Communication Technologies (ICT)-based services and systems. By monitoring environmental changes and inhabitant's activities, an assistive system in a Smart Home can process perceived sensor data, make timely decisions, and take appropriate actions to assist an inhabitant perform Activities of Daily Living (ADL), thus extending the period of time living independently within their own home environment. [1]

Currently, there are a number of Smart Home projects being developed for the purpose of proof-of-concept demonstration in addition to the establishment of systems in real living environments. Those projects address simple monitoring scenario to advanced forms of assistance. For example, advanced ADL systems intend to provide just-in-time activity guidance for elderly people and those suffering from cognitive deficiencies such as Alzheimer's disease, in completing their ADLs.

In all cases, a key task of all those systems is concerned with the ability to recognize ADL in a residential context. ADL are basic activities in a typical human life such as “cooking” or “cleaning the house”. So an action can be considered like a sequence of primitive actions that fulfil a function or simple purpose.

Along the time, multiple solutions have been proposed to support ADL recognition that leverage various kinds of technologies. The problem is inherently multimodal and it provides a great baseline to apply AI methods.

In this thesis we explored the ADL recognition problem by focusing on video-based observations.

Specifically, vision-based human action and activity recognition are becoming increasingly relevant among the computer vision community with multiple applications including visual surveillance, human-computer interaction, autonomous driving and several more.

Making the recognition from video can be particularly effective in assisted living and smart home contexts, as behavioural and lifestyle profiles can be constructed through the recognition of ADLs over time. Moreover, starting from visual data, this creates a

useful tool for unobtrusive home environment monitoring, doing temporal localization of those actions in videos.

If the system recognizes ADL properly, for instance, it is applicable to nursing services, rehabilitation, lifestyle habit improvements and monitoring.

Action recognition in a residential context presents several challenges. Individuals have different lifestyles, habits, or abilities and as such have their own way of performing ADLs. Though ADLs usually follow some kind of pattern, there are no strict constraints on the sequence and duration of the actions. For example, to prepare a meal one can first turn on the cooker and then place a pan on the cooker, or vice versa. The wide range of ADLs and the variability and flexibility in the manner in which they can be performed require an approach that is scalable to large scale activity modelling and recognition.

In addition, it is of particular interest, as videos of a person's daily life cannot be effectively parsed by human operators due to their huge volume and the fact that they often contain long segments with uninteresting content.

This project thesis concentrates in the context mentioned above, focusing on ADL recognition task and proposing as a technical solution to the problem the adoption of deep learning (machine learning) methods.

For what is concerned the training material, the availability of various large video datasets is playing a key role in the increasing level of interest on vision-based human action recognition. In recent years, several new specific datasets have been created for action recognition, so there is a variability introduced by various actors performing the actions and different modalities of interactions with the environment scenes around the actors.

The use of publicly available datasets has two main advantages. On one hand, they save time and resources because there is no need to record new video-sequences or pay for them, so it becomes possible to focus on the particular algorithms and implementations instead on data generation. On the other hand, the use of the same datasets facilitates the comparison of different approaches and gives insight into the abilities of the different methods.

Due to advances in deep learning (machine learning) most of the Computer-Vision applications are now using deep learning to get better accuracy. Even then, Computer Vision’s low level tasks are being used for image pre-processing, before feeding into deep learning networks.

Next sections provide a more detailed description about the business problem as well as about the technical solution designed and developed by this project.

1.1 Business problem description

The main business problem that this thesis project focus on, is concerned with the design and implementation of a non-invasive system to support the estimate of the Multidimensional Prognostic Index (MPI) about an elderly person that lives in its residential context.

In particular, the Multidimensional Prognostic Index (MPI) is a prognostic tool based on a standard Comprehensive Geriatric Assessment (CGA) that predicts short- and long-term mortality in elderly subjects in order to define the severity grade of mortality risk: low, moderate and severe.

The MPI is a derived index based on six commonly used geriatric assessment scales exploring cognitive, functional, nutritional and clinical status, as well as on information about drugs taken and patient's social support and was carried out using six standardized scales and information on medications and social support network, for a total of 63 items in eight domains.

Its long-term predictive value has been established in the overall hospitalized population as well as in older subjects hospitalized for specific clinical conditions including pneumonia, dementia, heart and renal failure, and transient ischemic attack. [2]

Several studies showed that the prognostic accuracy of the aggregated MPI was significantly greater than the prognostic accuracy of its individual components considered singularly. Thus the multidimensional approach can be successfully used in elderly patients for both clinical and administrative purposes. [3]

It is possible to see an MPI in the *Appendix A*.

So, estimating the MPI index is a crucial tool from a geriatric practice perspective. According to the geriatric team which I worked with during this thesis project, several items of the MPI can be estimated in advance, directly at the patient's home, before entering in hospital, and this is crucial to have more effective treatments.

In this thesis project we focused on two CGA domains such as Activities of Daily Living (ADL) and Instrumental Activities of Daily Living (IADL) and trying to estimate them using a vision-based action recognition system.

1.2 Technical objective of the thesis project

This thesis project designed and developed a solution to the business problem described in the previous section that leverages Convolutional Neural Networks (CNNs) to create a Deep Learning model that is able to detect different Activities of Daily Living in a non-invasive way, through the video provided by an RGB camera inside a room in a home environment.

Building a deep learning model capable of doing action recognition based on the integration of spatial, temporal and audio components is very complex in terms of model construction and necessary resources, so it was decided to create a proof-of-concept using the videos provided only considering the spatial component without taking into account the other 2 components during the deep learning model developing.

To train the model we leveraged a recent open-source video dataset produced and released by the MIT-IBM Watson AI Lab initiative that includes a collection of one million labeled 3 second videos, many of them in relation to the actions to be recognized, so this data source will be analysed and used.

To develop deep learning models, multiple approaches were explored and carried on during the thesis project.

Specifically, three different ADL action recognition models was developed: the first using an Auto ML (Auto Machine Learning) approach, while the other two using a transfer learning approach starting from different pre-trained models.

For the Auto ML approach, it has been utilized a tool called "NeuNetS" which allows to automatically synthesize deep neural network models, by only providing data. This allows non-expert users to build neural networks for specific tasks and datasets in a fraction of the time would take a more experienced figure.

Eventually, the thesis project aimed also to build a web-application with a dashboard in order to allow users to interact with the model by giving the possibility to upload a video and see in real time which Activities of daily living are predicted.

1.2 Outline of contents

This thesis is structured as follows:

- Chapter 2 introduces the datasets used to train Action Recognition models, focusing on the "Moments in Time" dataset. It also highlights all data management and data pre-processing activities carried on to prepare the training dataset.

- Chapter 3 provides an overview of the deep learning technologies that are utilized by this thesis to implement the ADL action recognition models.

- In Chapter 4 it is described in details how the training data is retrieved and pre-processed from the original dataset.

- Chapter 5 reports the deep learning models developed.

- Chapter 6 describes the web-based application interfacing the deep learning models developed, by scoring uploaded images and video.

- Chapter 7 provide conclusions and highlights for possible future works.

- The Appendix A shows a Multidimensional Prognostic Index

2 Sources of data to train machine learning models

To train the ADL action recognition model we evaluated multiple options and eventually ended to select a data sources already ready and validated, because otherwise it would have been very time consuming to generate the data.

This chapter provides a review of the dataset we evaluated and details the “Moment in Time” dataset adopted by this project.

2.1 Action recognition datasets

There are multiple datasets that can provide a good source of data to train an action recognition model that has been released recently. For instance, there are datasets about to train model to detect abandoned objects, recognition of activities of daily living (ADL), crowd behaviour, detection of human falls, gait analysis, or pose and gesture recognition. [4]

Action recognition task involves the identification of different actions from video clips (a sequence of 2D frames) where the action may or may not be performed throughout the entire duration of the video.

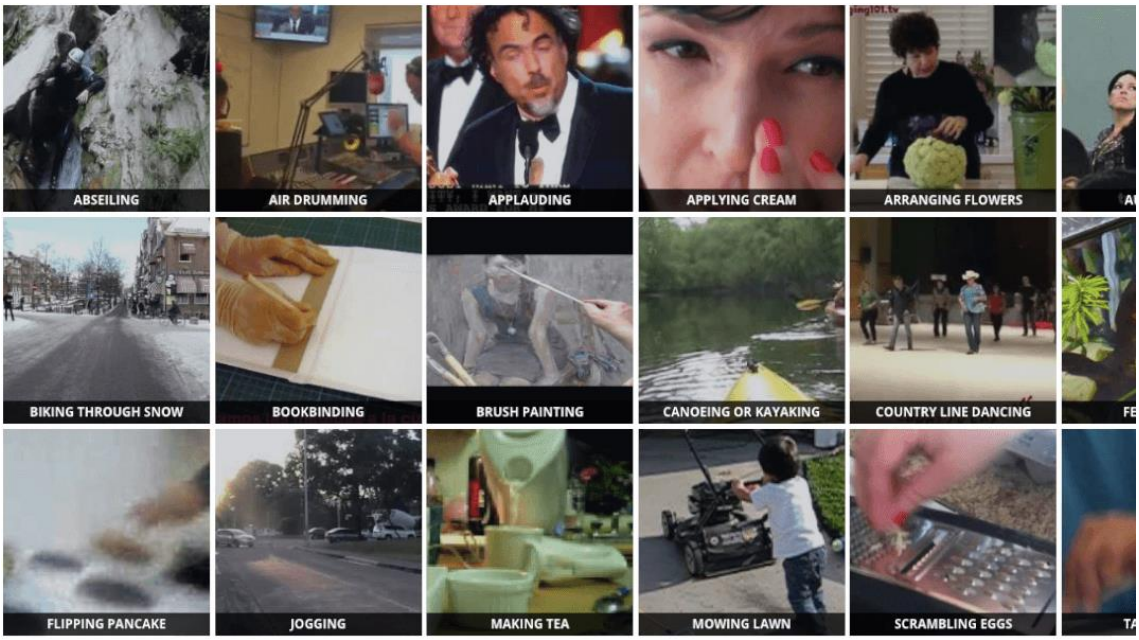


Figure 1 Examples of classes in a Datasets for Action Recognition [5]

This task looks similar to the image classification problem, but in this case, the task is extended to multiple frames with further aggregation of the predictions from each frame. And we know that after the introduction of the ImageNet [6] dataset, deep learning algorithms are doing a pretty good job in image classification, but there are several things that turn action recognition into a much more challenging task. This includes huge computational cost, capturing long context, and of course, a need for good datasets. [5]

Some of the most popular dataset for Action Recognition are:

- KINETICS-600 (year 2018)
  It's a dataset introduced by Google's DeepMind team. This is a large-scale, high-quality dataset of YouTube URLs created to advance models for human action recognition. Its last version includes around 500,000 video clips that cover 600 human action classes with at least 600 video clips for each action class which last around 10 seconds each.

  The clips have been through multiple rounds of human annotation using a single-page web application for the labeling task, in order to use this signal later during model training.

  The actions cover a broad range of classes including human-object interactions such as playing instruments, arranging flowers, mowing a lawn, scrambling eggs and so on.

- Moments in Time (year 2018)
  It is another large-scale dataset, which was developed by the MIT-IBM Watson AI Lab. With a collection of one million labeled 3-second videos, it is not restricted to human actions only and includes people, animals, objects and natural phenomena, that capture the gist of a dynamic scene.
  The dataset has a significant intra-class variation among the categories. For instance, video clips labeled with the action "opening" include people opening doors, gates, drawers, curtains and presents, animals and humans opening eyes, mouths and arms, and even a flower opening its petals.

  There are 339 different action classes in the dataset and each video is labeled with only one action class.

- Sparsely Labeled ACtions Dataset - SLAC (year 2017)
  It is introduced by the group of researchers from MIT and Facebook. The dataset is focused on human actions, similarly to Kinetics, and includes over 520K untrimmed videos retrieved from YouTube with an average length of 2.6

minutes. 2-second clips were sampled from the videos by a novel active sampling approach. This resulted in 1.75M clips, including 755K positive samples and 993K negative samples as annotated by a team of 70 professional annotators.

The distinctive feature of this dataset is the presence of negative samples and includes 200 action classes taken from the ActivityNet dataset, but is still not available for download.

- VLOG (year 2017)
  The VLOG dataset contains 114,000 videos and differs from the previous datasets in the way it was collected. The traditional approach to getting data starts with a laundry list of action classes and then searching for the videos tagged with the corresponding labels. However, such an approach runs into trouble because everyday interactions are not likely to be tagged on the Internet. The people tend to tag unusual things like for example, jumping in a pool, presenting the weather, or playing the harp. As a result, available datasets are often imbalanced with more data featuring unusual events and less data on our day-to-day activities.

  To solve this issue, the researchers from the University of California suggest starting out with a superset of what we actually need, namely interaction-rich video data, and then annotating and analysing it after the fact. They start data collection from the lifestyle VLOGs – an immensely popular genre of video that people publicly upload to YouTube to document their lives.

  As the data was gathered implicitly, it represents certain challenges for annotation. The researchers decided to focus on the crucial part of the interaction, the hands, and how they interact with the semantic objects at a frame level. Thus, this dataset can also make headway on the difficult problem of understanding hands in action. [7]

- Something something (year 2017)
  It is a database of video prediction tasks whose solutions require a common sense understanding of the depicted situation, providing detailed physical aspects about actions and scenes. The database currently contains more than 100,000 videos across 174 classes, which are defined as caption-templates.

2.1.1 Dataset specialized in Activities of Daily Living

Wanting to deepen the search for datasets for action recognition, and in particular for the ADL, we come across three main types of datasets, which contains:

- data retrieved from sensors, for example acceleration data coming from a wrist-worn wearable device or using a wireless sensor network, with the associated label that expresses the action performed [8] [9]
- videos taken in first-person camera views [10]
- videos acquired by a camera in a certain position in the room [11]

In all these cases, the specific datasets for the recognition of the ADL are very specific for the objective to be achieved.

Since the environment and the subjects taken as end-user in this thesis are very different from each other, there is the needs to have a set of actions that is varied in order to generalize as much as possible, the "Moments in Time dataset" was therefore the best candidate to support our specific need.

2.1.2 The "Moments in Time" dataset

The "Moments in Time" dataset is a large-scale human-annotated collection of one million short videos corresponding to dynamic events unfolding within three seconds, developed by the MIT-IBM Watson AI Lab.

Temporal events of such length correspond to the average duration of human working memory, which is a short-term memory-inaction buffer specialized in representing information that is changing over time. Additionally, three seconds is a temporal envelope which holds meaningful actions between people, objects and phenomena (e.g. wind blowing, objects falling on the floor, shaking hands, playing with a pet, etc.). Compound activities that occur at longer time scales can be represented by sequences of three second actions.

This dataset, designed to have a large coverage and diversity of events in both visual and auditory modalities, can serve as a new challenge to develop models that scale to the level of complexity and abstract reasoning that a human processes on a daily basis.

The dataset has a significant intra-class variation among the categories. For instance, video clips labeled with the action “opening” include people opening doors, gates, drawers, curtains and presents, animals and humans opening eyes, mouths and arms, and even a flower opening its petals.

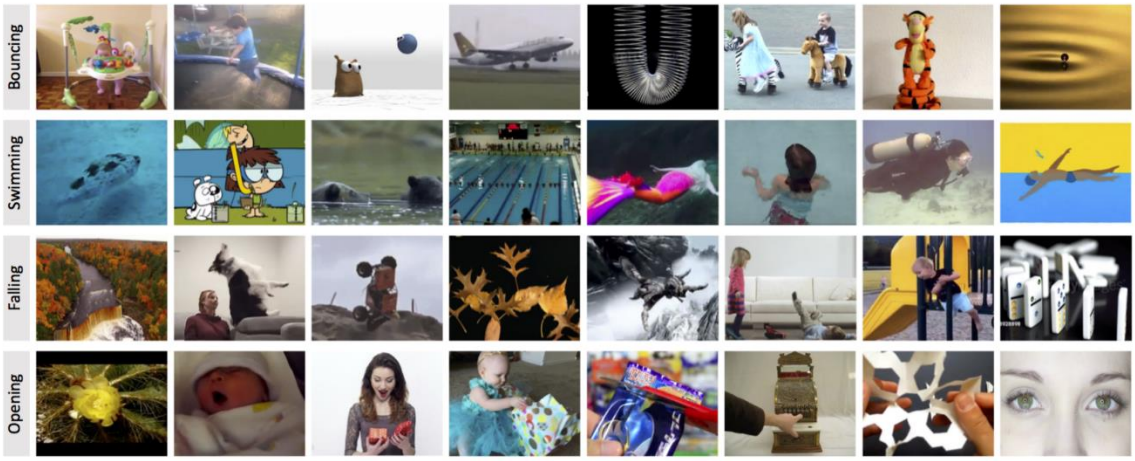


Figure 2 Intra-class variation among the categories. For example, car engines, books, and tulips can all open. [12]

The action classes in Moments in Time dataset are chosen such that they include the most commonly used verbs in the English language, covering a wide and diverse semantic space. So, there are 339 different action classes in the dataset and each video is labeled with only one action class. [12]

It is not restricted to human actions only and includes people, animals, objects and natural phenomena, that capture the gist of a dynamic scene. it is important to consider that visual and auditory events can be symmetrical in time ("opening" is "closing" in reverse), and either transient or sustained. Nevertheless, this thesis is considering mostly data related to people because the end-user are elderly people and the video is pre-processed in order to extract a frame that convey the action, so only the spatial component is taken into account, without considering the temporal and auditory parts.

2.2 Video pre-processing

Given the dataset, it is common to come across lots of raw data which is not fit to be readily processed by machine learning algorithms. We need to pre-process the raw data before it is fed into various machine learning algorithms.

While doing any kind of analysis with data it is important to clean it, as raw data can be highly unstructured with noise or missing data or data that is varying in scales which makes it hard to extract useful information. Data Preprocessing is the process of preparing the data for analysis. This is the first step in any machine learning model.

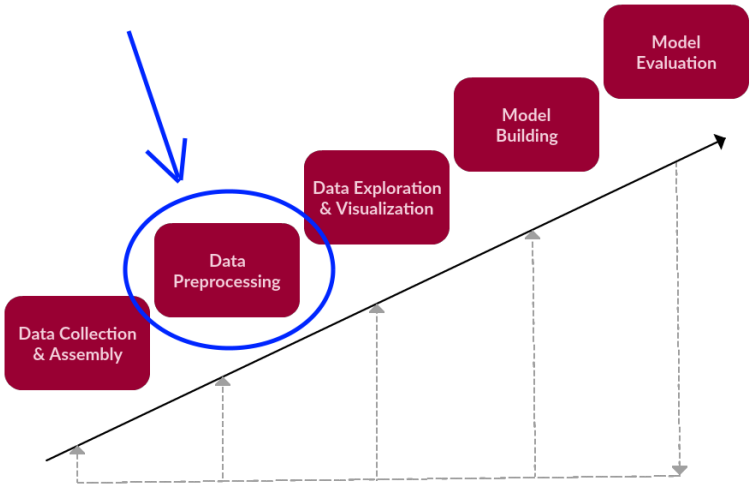


Figure 3 Data preprocessing is nothing but the readying of data for experimentation, transforming raw data for further processing [13]

For achieving better results from the applied model in Machine Learning projects the format of the data has to be in a proper manner. Some specified Machine Learning model needs information in a specified format, for example, Random Forest algorithm does not support null values, therefore to execute random forest algorithm null values have to be managed from the original raw data set.

Having in this case the dataset downloaded on the PC, a way of interacting with the files in filesystem is necessary. One of the widely used programming languages in order to do that is Python.

Python is a general-purpose high level programming language that is widely used in data science and for producing deep learning algorithms, mainly because is an interpreted, dynamically-typed language with a precise and efficient syntax.

In chapter 4 the code that performs the described part will be shown in more detail, using Python library as:

- *MoviePy* for video editing, which can be used for basic operations (like cuts, concatenations, title insertions), video compositing (a.k.a. non-linear editing), video processing, or to create advanced effects.
- *OpenCV* (Open Source Computer Vision Library) is an image and video processing library available in different high level programming languages which includes several hundreds of computer vision algorithms.
  It is used for all sorts of image and video analysis, like facial recognition and detection, license plate reading, photo editing, advanced robotic vision, optical character recognition, and a whole lot more.
- *os* is an inbuilt python package that allows the interaction with files in the computer, providing a portable way of using operating system dependent functionality. It can be used to display content in directories, create new folders and even delete folders

Another very powerful tool used is FFmpeg (fast forward MPEG), which is not a python library, but it is a free software project consisting of a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams. It is designed for command-line-based processing of video and audio files. It has to be invoked in python via os.system, because this allows to execute shell commands.

3 Technology overview

This chapter will cover the concepts, theories and technologies that will be referred to further in this thesis. First of all, deep learning will be introduced, analysing the types of networks that will be used in the models. Afterwards an overview of the computational resources necessary for the creation of deep learning models will be provided. Then an IBM Cloud tool that allows non-expert users to build neural networks is introduced. Finally, the Python library "Flask" that allows to create web-applications will be described.

3.1 Deep Learning and neural networks

Deep learning is a collection of algorithms involved in machine learning, used to model high-level abstractions in data through model architectures, which are composed of multiple nonlinear transformations. It is part of a broad family of methods employed for machine learning that are based on learning representations of data.

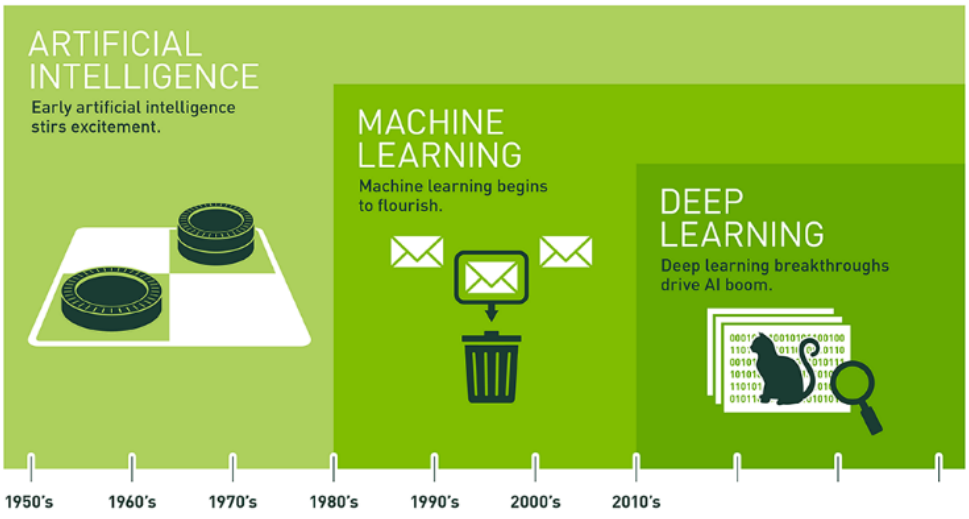


Figure 4 Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions. [14]

Most deep learning methods use neural network architectures, which are considered highly promising decision-making nodes.

To make sense of observational data (like photos or audio), neural networks pass data through interconnected layers of nodes. When information passes through a layer, each node in that layer performs simple operations on the data and selectively passes the results to other nodes. Each subsequent layer focuses on a higher-level feature than the last, until the network creates an output.

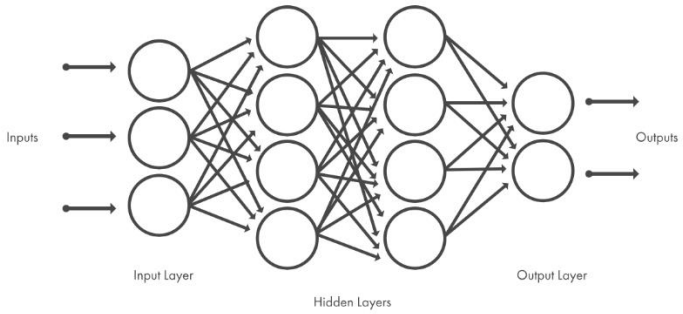


Figure 5 Basic structure of a Neural Network

The term “deep” usually refers to the number of hidden layers in the neural network. Traditional neural networks only contain 2-3 hidden layers, while deep networks can have as many as 150. [15]

Between the input layer and the output layer there are several hidden layers. And here’s where users typically differentiate between neural networks and deep learning: a basic neural network might have one or two hidden layers, while a deep learning network might have dozens or even hundreds. For example, a simple neural network with a few hidden layers can solve a common classification problem. Increasing the number of layers and nodes can potentially increase the accuracy of your network. However, more layers means your model will require more parameters and computational resources and is more likely to become overfit.

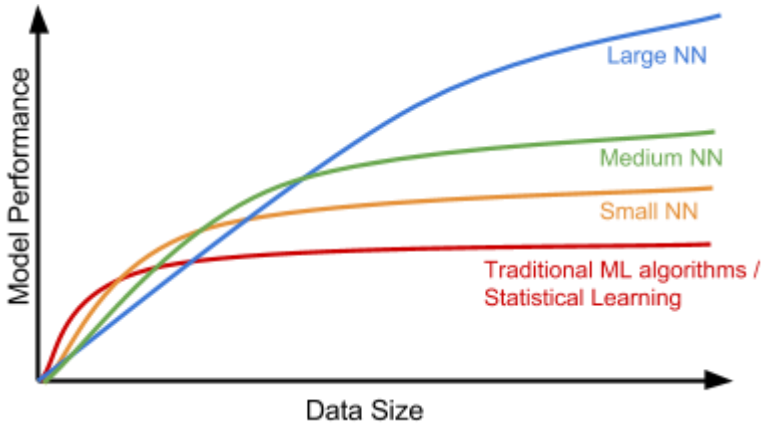


Figure 6 The data scale versus the model performance.

Another key difference is deep learning algorithms scale with data, so they often continue to improve as the size of your data increases, whereas shallow learning converges. Shallow learning refers to machine learning methods that plateau at a certain level of performance when you add more examples and training data to the network.

Deep learning models are trained by using large sets of labeled data. The more data you train it on, the more accurate your deep learning model will be. Training means that you're feeding data to the model, getting an output, and then using that output to make adjustments. For example, if a model is trained on a bunch of pictures of cats and then feed it new cat photos it has never seen before, it should be able to pick out the cats in the new photos. If it doesn't, you can change the way the network's nodes are weighing certain characteristics of the images (the presence of whiskers and a tail, for instance). Weight, in this case, is a number that represents the importance of a characteristic. The higher the weight, the higher the influence that characteristic has on the nodes.

But how did the model even know to look for whiskers? Typically, a data scientist will engineer features for the model to consider and feed it labeled data during the training process (e.g., a series of labeled photos of cats). But one of the amazing thing about deep learning is that you don't necessarily have to complete this step.

A machine learning workflow starts with relevant features being manually extracted from data in order to create the model. With a deep learning workflow, features are directly learnt from the data without the need for manual feature extraction. In addition, deep learning performs "end-to-end learning" – where a network is given raw data and a task to perform, such as classification, and it learns how to do this automatically.

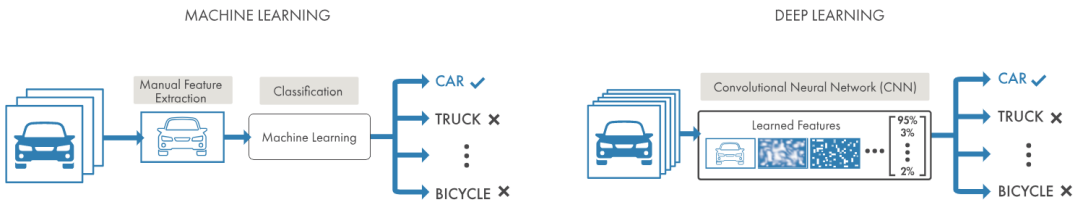


Figure 7 In machine learning, you manually choose features and a classifier to sort images. With deep learning, feature extraction and modelling steps are automatic.

Deep learning removes the manual identification of features in data and, instead, relies on whatever training process it has in order to discover the useful patterns in the input examples. This makes training the neural network easier and faster, and it can yield a better result that advances the field of artificial intelligence.

Deep learning is a key technology behind driverless cars, enabling them to recognize a stop sign, or to distinguish a pedestrian from a lamppost. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. Deep learning is getting lots of attention lately and for good reason: it's achieving results that were not possible before, because models can now achieve state-of-the-art accuracy, sometimes exceeding human-level performance.

3.1.1 Convolutional Neural Networks

One of the most popular types of deep neural networks is known as Convolutional Neural Networks (CNN or ConvNet). A CNN convolves learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images.

Convolution is a mathematical operation on two functions, here referring to an operation between two matrices. The convolutional layer has a fixed small matrix defined, also called kernel or filter. As the kernel is sliding, or convolving, across the matrix representation of the input image, it is computing the element-wise multiplication of the values in the kernel matrix and the original image values. Specially designed kernels can process images for common purposes like blurring, sharpening, edge detection and many others, fast and efficiently.

CNN is a type of feed-forward artificial neural networks, in which the connectivity pattern between its neurons is inspired by the organization of the visual cortex system:

- The primary visual cortex (V1) does edge detection out of the raw visual input from the retina.
- The secondary visual cortex (V2), also called prestriate cortex, receives the edge features from V1 and extracts simple visual properties such as orientation, spatial frequency, and colour.
- The visual area V4 handles more complicated object attributes. All the processed visual features flow into the final logic unit, inferior temporal gyrus (IT), for object recognition.
- The shortcut between V1 and V4 inspires a special type of CNN with connections between non-adjacent layers: Residual Net (He, et al. 2016) containing “Residual Block” which supports some input of one layer to be passed to the component two layers later.

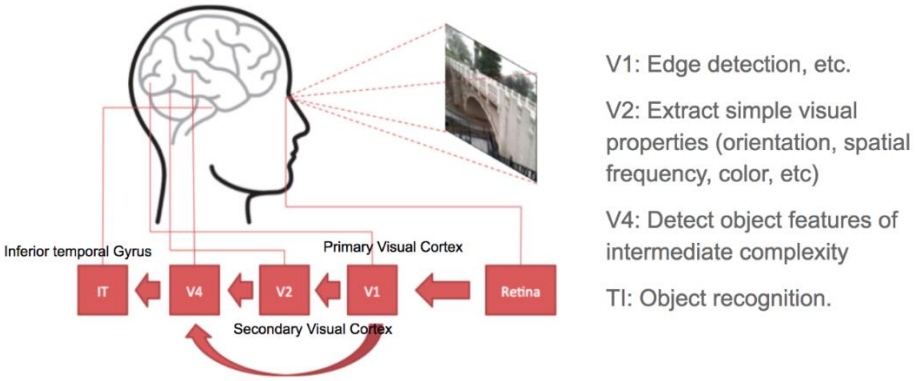


Figure 8 Illustration of the human visual cortex system [16]

Convolutional Neural Networks have a different architecture than regular Neural Networks. Regular Neural Networks transform an input by putting it through a series of hidden layers. Every layer is made of a set of neurons, and each layer is fully connected to all neurons in the layer before. Finally, there is a last fully-connected layer — the output layer — that represent the predictions.

Convolutional Neural Networks are a bit different. First of all, the layers are organised in 3 dimensions: width, height and depth. Further, the neurons in one layer do not connect to all the neurons in the next layer but only to a small region of it. Lastly, the final output will be reduced to a single vector of probability scores, organized along the depth dimension.

CNNs eliminate the need for manual feature extraction, so it is not needed to identify features used to classify images. The CNN works by extracting features directly from images. The relevant features are not pre-trained; they are learned while the network trains on a collection of images. This automated feature extraction makes deep learning models highly accurate for computer vision tasks such as object classification.

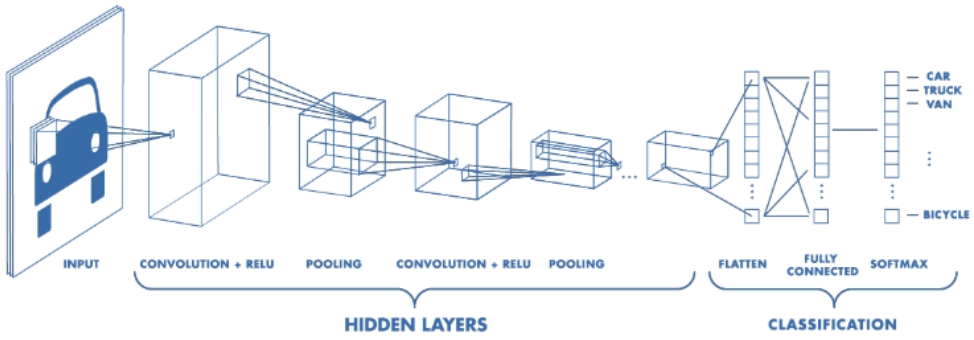


Figure 9 Example of architecture of a CNN

Convolutional and pooling (or “sub-sampling” in Fig. 4) layers act like the V1, V2 and V4 visual cortex units, responding to feature extraction. The object recognition reasoning happens in the later fully-connected layers which consume the extracted features.

CNNs learn to detect different features of an image using tens or hundreds of hidden layers. Every hidden layer increases the complexity of the learned image features. For example, the first hidden layer could learn how to detect edges, and the last learns how to detect more complex shapes specifically catered to the shape of the object we are trying to recognize.

3.1.2 PyTorch and Keras

With the increasing demand in the field of Data Science, there has been an enormous growth of Deep learning technology in the industry.

PyTorch and Keras are both very powerful open-source tools in Deep Learning framework. These are some of the favourites among Data Scientists as well as beginners in the field of Deep Learning and both frameworks have gained quite a lot of popularity.

- Keras is a high-level API capable of running on top of TensorFlow, CNTK, Theano, or MXNet. Since its initial release in March 2015, it has gained favour for its ease of use and syntactic simplicity, facilitating fast development. Keras has a simple architecture and it is designed to enable fast experimentation with deep neural networks. It's supported by Google.

- PyTorch, released in October 2016, is a lower-level API focused on direct work with array expressions. It has gained immense interest in the last year, becoming a preferred solution for academic research, and applications of deep learning requiring optimizing custom expressions. It's supported by Facebook.

There is no absolute answer to which one is better. The choice of the framework usually depends on: Technical background, Requirements and Ease of Use.

Keras is most suitable for: making readable code, Rapid Prototyping, Small Dataset, Multiple back-end support

PyTorch is most suitable for: Flexibility, Short Training Duration, Debugging capabilities, high performance models and large datasets that require fast execution.

Keras is without a doubt the easier option if you want a plug & play framework: to quickly build, train, and evaluate a model, without spending much time on mathematical implementation details.

Knowledge of the core concepts of deep learning is transferable. Once you master the basics in one environment, you can apply them elsewhere and hit the ground running as you transition to new deep learning libraries.

3.1.3 “Moments in Time” pre-trained models

When the dataset “Moments in Time” was published, the researchers included baseline results of several known models trained and tested on the dataset, addressing separately, and jointly, three modalities: spatial, temporal and auditory.

To do so they generate a training set of 802,264 videos with between 500 and 5,000 videos per class for 339 different classes and evaluate performance on a validation set of 33,900 videos with 100 videos for each class. They additionally withhold a test set of 67,800 videos consisting of 200 videos per class which will be used to evaluate submissions for an action recognition challenge held at CVPR'18 jointly with the ActivityNet Challenge 2018.

Top-1 and top-5 classification accuracy was used as the scoring metrics. Top-1 accuracy indicates the percentage of testing videos for which the top confident predicted label is correct. Top-5 accuracy indicates the percentage of the testing videos for which the ground-truth label is among the top 5 ranked predicted labels. This is appropriate for video classification as videos may contain multiple actions.

For the spatial modality, an experiment was made with a 50 layer resnet (Resnet50) trained on randomly selected RGB frames from each video with networks trained from scratch, initialized on a scene dataset called Places, and initialized on ImageNet. In the testing phase, an average of the predictions from 6 equi-distant frames was taken.

Among these models the best in terms of accuracy is the one trained after being initialized on ImageNet (dataset that contains about 1.4 million labeled images and 1000 different categories including animals and everyday objects), which reach a 27.16% top-1 accuracy and a 51.68% top-5 accuracy, by only considering the spatial component.

This pre-trained model based on CNNs has been published on GitHub with some sample Python scripts which are using the PyTorch framework because the model was trained using that framework. This model is used for the creation of one of the action recognition model presented in this thesis. [17]

3.1.4 Transfer Learning

To train a deep network from scratch, for example an Image classifier that will achieve near or above human level accuracy on Image classification, a massive amount of data is needed, large compute power, and lots of time on hands (days or weeks).

So due to time restrictions or computational restraints, it's not always possible to build a model from scratch which is why pre-trained models exist. Pre-trained model can be used as a benchmark to either improve the existing model, or test another model against it. Several researchers built models, trained on large image datasets like ImageNet, COCO, Open Images, and decided to share their models to the general public for reuse.

Transfer learning It is a popular approach in deep learning where a model developed for a task is reused as the starting point for a model on a second task. This has the advantage of needing much less data (e.g. processing thousands of images, rather than millions), so computation time drops to minutes or hours.

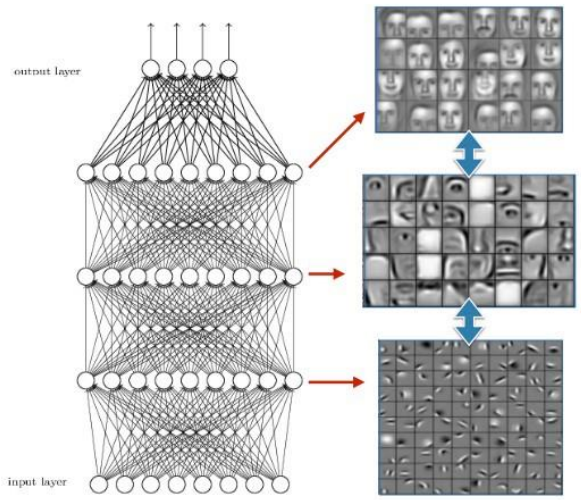


Figure 10 A neural network learns to detect objects in increasing level of complexity.

Transfer learning works for Image classification problems because Neural Networks learn in an increasingly complex way. In fact, going deeper in the network the more image specific features are learnt. For example, in a neural network trying to detect faces, it was noticed that the network learns to detect edges in the first layer, some basic shapes in the second and complex features as it goes deeper.

A typical CNN has two parts: a "Convolutional base" which is generating features from the image and a "Classifier", which classifies the image based on the detected features.

Applying the transfer learning approach starting from a pre-trained model leads to removing the original *classifier* and adding a new classifier that fits the purposes, then depending on data size and similarity between models, it is possible to adjust how much of CNN *convolutional base* to train by freezing certain layers, according to prevents its weights from being modified or not. There are three possibilities in order to fine-tune the model:

- Train the entire model: using the architecture of the pre-trained model and train it according to your dataset. You're learning the model from scratch, so you'll need a large dataset (and a lot of computational power).

- Train some layers and leave the others frozen: choosing how much to adjust the weights of the network. Usually, if there is a small dataset and a large number of parameters, it is advisable leave more layers frozen to avoid overfitting. By contrast, if the dataset is large and the number of parameters is small, it is advisable to train more layers to the new task since overfitting is not an issue.

- Freeze the convolutional base: so the pre-trained model is used as a fixed feature extraction mechanism, which can be useful if there is limited computational power, the dataset is small, and/or pre-trained model solves a problem very similar to the one to solve.

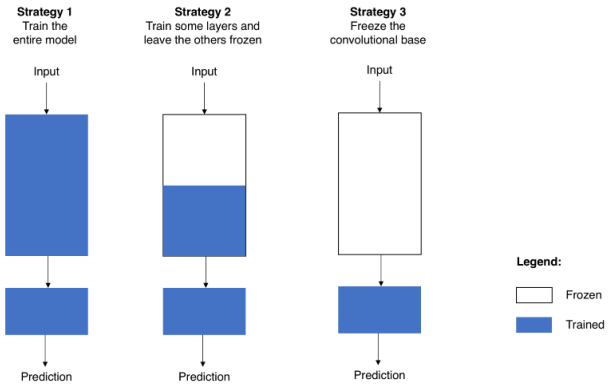


Figure 11 Fine-tuning strategies

In case of working on convolutional base it's required to be careful with the learning rate used, because is a hyper-parameter that controls how much to adjust the weights of the network. Using a pre-trained model based on CNN, it's smart to use a small learning rate because high learning rates increase the risk of losing previous knowledge. Assuming

that the pre-trained model has been well trained, which is a fair assumption, keeping a small learning rate will ensure that you don’t distort the CNN weights too soon and too much. [18]

3.1.5 GPU in Deep Learning

A GPGPU (General-Purpose computing on Graphics Processing Units) is a parallel programming setup involving GPUs & CPUs which can process and analyse data in a similar way to image or other graphic form. GPGPUs were created for better and more general graphic processing, but were later found to fit scientific computing well. This is because most of the graphic processing involves applying operations on large matrices.

The use of GPGPUs for scientific computing started back in 2001 with implementation of Matrix multiplication. At this time, researchers had to code every algorithm on a GPU and had to understand low level graphic processing, but in 2006, NVidia came out with a high level language CUDA, which helps writing programs from graphic processors in a high level language. This was probably one of the most significant change in the way researchers interacted with GPUs. [19]

Deep learning is a field with intense computational requirements: with no GPU this might look like months of waiting for an experiment to finish, or running an experiment for a day or more only to see that the chosen parameters were off and the model diverged. With a good, solid GPU, one can quickly iterate over designs and parameters of deep networks, and run experiments in days instead of months, hours instead of days, minutes instead of hours.

In order to train a deep learning model, two main operations have to be performed:

- Forward pass: in which the input is passed through the neural network and after processing the input, an output is generated.
- Backward pass: for updating the weights of neural network on the basis of error got in forward pass.

Both of these operations are essentially matrix multiplications.

For example, the architecture VGG16 (a convolutional neural network of 16 hidden layers which is frequently used in deep learning applications) has around 140 million parameters (weights and biases). So considering all the required matrix multiplications in order to pass just one input to this network, it would take years to train this kind of systems using traditional approaches. GPU allows to do all the operations at the same time instead of doing it one after the other. This is why is preferred instead of a CPU (central processing unit) for training a neural network.

3.2 GPU in Cloud – Google Colab and Kaggle Kernel

One of the biggest hurdles of democratizing deep learning has been the need of a GPU accelerated environment. If we want to spend our time efficiently testing and learning across problem statements, we need to have GPUs in our machine or in a GPU based server. But these GPU based environments across platforms (Google Cloud Platform, AWS, Azure, …) cost at least around 0.5 dollars per hour and often the training phase requires the processing of large amounts of data using high-end GPU clusters for many hours, so the costs can be really high.

Google has two products called *Colab* and *Kaggle kernel* which offer a free Jupyter notebook environment that requires no setup and runs entirely in the cloud, providing a free GPU usage. This services support Python and all major libraries are pre-installed and ready to be imported. It's also possible to share Jupyter notebook among each other without having to download, install, or run anything other than a browser.

Not only is this a great tool for improving your coding skills, but it also allows absolutely anyone to develop deep learning applications using popular libraries such as PyTorch, TensorFlow, Keras, and OpenCV. Those libraries are always updated, as well as the computational resources, in fact as of early March 2019, Kaggle has upgraded its GPU chip from a NVidia Tesla K80 to a NVidia Telsa P100, so for example image classification task can train definitely faster. Colab still gives a K80.

So working with an intensive PyTorch project, it could be worth developing on Kaggle to receive a speed boost. [20]

Both Colab and Kaggle are great resources to start deep learning in the cloud, other few differences are:

- Colab provides more flexibility to adjust your batch sizes
- Kaggle offer an execution time of maximum 6 hours, while Colab 12 hours. After this time the session restarts.
- Colab stores notebooks in Drive. Kaggle stores notebooks in its own data store. Sharing and commenting in Colab Drive notebooks work like Docs.
- Tensor Processing Unit (TPU) are available only in Colab
- Colab supports loading and saving notebooks in GitHub, instead Kaggle has its own versioning system.
- Kaggle supports preloaded data sets. In Colab, data sets need to be loaded from somewhere else like Drive or Google Cloud Storage.
- Colab supports local Jupyter backends running on your local machine or GCE / AWS Virtual Machines. Kaggle execution all happens on Kaggle service backends.

3.3 IBM Watson OpenScale – NeuNetS

NeuNetS is an automated Neural Network Synthesis engine for custom neural network design that is available as part of IBM's AI OpenScale's product since December 2018. NeuNetS is available for both Text and Image domains and can build neural networks for specific tasks in a fraction of the time it takes today with human effort, and with accuracy similar to that of human-designed AI models.

Every business has unique challenges to solve, and the range of use-cases for AI is constantly expanding. While pre-built AI models exist for certain scenarios, to try and meet the constraints that are unique to each application, AI teams will need to think about developing custom AI models of their own.

Artificial neural networks are arguably the most powerful tool currently available to data scientists and businesses. However, only a small proportion of data scientists have the skills and experience needed to create a neural network from scratch, and the demand far exceeds the supply. As a result, getting a new neural network that is architecturally custom designed to meet the needs of that application, even to the proof-of-concept stage, requires a level of investment that most enterprises struggle to afford.

NeuNetS helps reduce the complexity and skills required to build AI models by automatically designing the architecture of neural networks for a given data, making data scientists more productive and enabling them to scale AI across their workflows.

Overall, NeuNetS has two main stages:

- *Coarse-grained* synthesis automatically optimizes and determines the overall architecture of the network: How many layers there should be, how they are connected, different architectural features like convolution layers and so on.

- The unique and novel step of *fine-grained* synthesis enables NeuNetS to take a deeper dive into each layer and optimizes the individual neurons and connection. For example, what kind of convolution filter should be applied, and which neurons and edges should be optimized. One of the critical breakthroughs that have enabled this capability is a very high-fidelity approach to performance estimation, which allows to bypass real-time training and analysis and design neural networks automatically in a matter of hours compared to the weeks or months that it might take a data scientist to train and optimize the AI model.

The data science teams can then further fine tune the model, leading to greater productivity and cost-efficiency. NeuNetS is a novel tool for augmenting human expertise with powerful, AI-driven optimization capabilities.

NeuNetS runs on a fully containerized environment deployed on the IBM Cloud with Kubernetes. The architecture is designed to minimize human interaction, automate user workload, and improve over usage. Users do not need to write code or have experience with existing deep learning frameworks: Everything is automated, from the dataset ingestion and pre-processing, to the architecture search training and model deployment.

The NeuNetS algorithm portfolio includes enhanced versions of recently published works, for o addressing fundamental problems such as dataset generality and performance scalability. Such as:

- TAPAS is an extremely fast neural-network synthesizer, performing close to transfer-learning approaches by relying on pre-generated ground-truth and smart prediction mechanisms. [21]

- NCEvolve synthesizes top-performant networks, minimizing the amount of training time and resource needs. [22]

- HDMS combines an improved version of hyperband with reinforcement learning to synthesize networks tailored for the less common datasets.

- Fine-grain synthesis engine uses an evolutionary algorithm for building custom convolution filters, leading to low-level fine-tuning of the neural architecture.

So the user has only to upload the data in order to make image or text classification, then the whole process is automatic. NeuNetS leverages existing IBM assets, such as DLaaS, HPO, and Watson Machine Learning (WML). Neural Networks models are synthesized on the latest generation NVIDIA® Tesla® V100 GPUs.

NeuNetS algorithms are designed to create new neural network models without re-using pre-trained models. So a wide space of network architecture configurations is explored and at the same time the model is fine-tuned for the specific dataset provided by the user.

When the synthesis finish, the user can see statistics on the uploaded data and the confusion matrix. Furthermore, it’s possible to download a Keras pre-trained model or deploying it to Watson Machine Learning in order to interact with the model through an API REST.

3.4 Flask Web-Application with Python

For building dynamic websites, one of the important and the most frustrating tasks is choosing a web framework. There are more than thousands of web frameworks in several programming languages, usually based on different use cases.

A web framework (WF) or web application framework (WAF) is a software framework that is designed to support the development of web applications including web services, web resources, and web APIs.

Flask is a micro level web framework written in Python. It is classified as a microframework because it does not require particular tools or libraries. It has no database abstraction layer, form validation, or any other components where pre-existing third-party libraries provide common functions. However, Flask supports extensions that can add application features as if they were implemented in Flask itself. Extensions exist for object-relational mappers, form validation, upload handling, various open authentication technologies and several common framework related tools. Extensions are updated far more regularly than the core Flask program. In fact, it's built with a small core and easy-to-extend philosophy.

Flask is based on Werkzeug (Utilities and Requests), Jinja2 (Templating).

Pros — Flask provides extreme simplicity, flexibility and fine-grained control. It implements bare-bones and leaves the rest of the customization to add-ons or the developers. Routing URLs is simple. Required database connections can be synced explicitly. It's easier to learn and implement and hence could be a good starting point for all the people out there who're craving to start their web development journey.

Cons — Non-Asynchronous. Lack of database and ORM (Object Relational Mapping — It is a programming technique in which a metadata descriptor is used to connect object code to a relational database). But it does support MySQL, PostgreSQL, MongoDB and other few databases and based on the use case we need to choose the most suitable database.

Finally, Flask is one of the best choice in order to start from the scratch and for building minimalistic web applications. Using Flask is possible to integrate Python based machine learning algorithms within the functions with ease for making applications more intelligent and cognitive.

This web framework is widely used to build: blogs, chats, dashboards, REST applications and so more. As a matter of fact, applications that use the Flask framework include Pinterest, LinkedIn, and the community web page for Flask itself.

It makes the process of designing a web application simpler. Flask lets us focus on what the users are requesting and what sort of response to give back. [23]

4 Building Training & test dataset for the ADL recognition task

In order to build a good deep learning model that is able to recognize actions, it is very important to provide the right data during training. Otherwise, if the whole process starts with incorrect data, the model will also be incorrect.

For this thesis the dataset "Moments in Time" is used as a source of data to work on, because the actions to be identified are represented by many videos and above all the majority of these are very similar to videos that would have been filmed within a home.

Furthermore, since the model will take into account the spatial component, without considering the temporal and audio component, frames from the videos will be extracted. "Moments in Time" is well suited to this because the action is short and usually present during all 3 seconds, so it is quite easy to extract a meaningful frame with respect to what could be working with video of variable length.

During this phase the possibility of adding a class that was not present in the "Moments in Time" dataset was taken into consideration, for example the action of “setting the table”. This would have involved the need to recover a few hundred videos showing the action "set the table", so it was chosen not to invest time in generating the data source for a class but to use only the classes within "Moments in Time" because each one contains at least a few thousand videos.

4.1 Selection of classes

The "Moments in Time" dataset contains 339 classes, some of which describe very generic actions such as working and pulling, while others are fairly specific such as surfing and hammering.

There are actions within "Moments in time" such as opening, closing, jumping that make sense only if the frames are also evaluated on the time component: for example, by analysing a single frame it is not possible to understand if a door is opening or by closing, or if a man is walking or standing. So these classes were discarded a priori during this project because they would need another type of model.

The same kind of reasoning was applied to the classes of action for which the audio component is fundamental, such as those that identify whether a man or a woman is talking, shooting or knocking.

4.1.1 Matching the Multidimensional Prognostic Index (MPI) with the "Moments in time" classes

Since in this project we want to estimate action in order to create a support system for the compilation of the MPI, a first analysis was carried out to try to identify which are the relevant classes for the domains of MPI and then generate the training and test dataset starting from those classes.

In order to do that, all the classes of "Moments in Time" were listed in a simple excel file and then for each element were provided:

- the Italian translation, generated using Google Translate and considering the first translation, to facilitate a quick reading at the time of consulting the spreadsheet.

- if the action falls within a residential context, both in an indoor or outdoor environment.

- if the action could trigger an alert because it is a dangerous situation. This was done only to identify actions that future developments could also integrate for the alert.

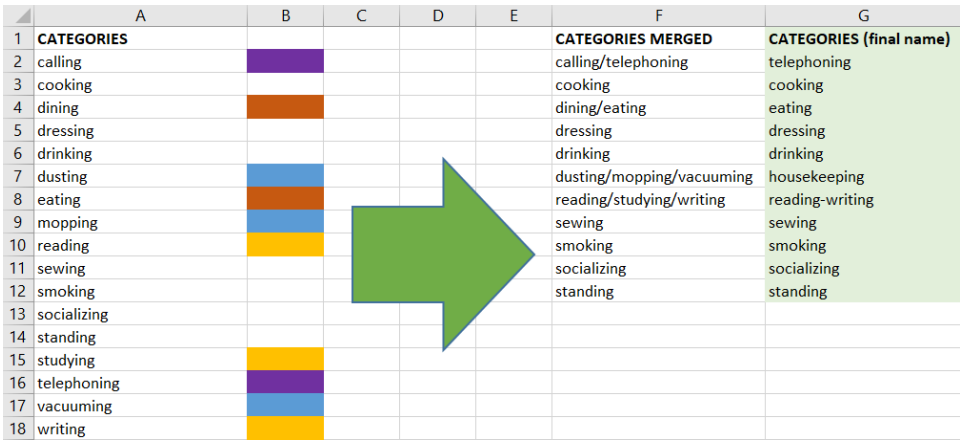
- Finally, if that the action must be taken into consideration because it is useful for compiling the MPI.

	A	B	C	D	E	F	G	H
1	CATEGORY EN	CATEGORY IT		HOME CONTEXT		DANGER		MULTIDIMENSIONAL PROGNOSTIC INDEX
2	clapping	applaudire						
3	praying	pregare						
4	dropping	lancio						
5	burying	seppellendo						
6	covering	copertura						
7	flooding	allagamento						
8	leaping	saltare						
9	drinking	bere						OK
10	slapping	schiaffeggiare						
11	cuddling	coccole						
12	sleeping	addormentato						
13	preaching	predicazione						
14	raining	piovere						
15	stitching	Cucitura						
16	spraying	spruzzatura						
17	twisting	torsione						
18	coaching	istruire						
19	submerging	sommersione						
20	breaking	rottura						
21	tuning	messa a punto						
22	boarding	imbarco						
23	running	in esecuzione						
24	destroying	distruttivo						
25	competing	competere						
26	giggling	ridacchiare						

Figure 12 A screenshot of the created excel.

Starting from this overall vision, a filter has been applied over the classes that are relevant for estimate items of the MPI. Subsequently, it emerged that some of these classes can be merged in order to generate a larger set of data and avoid errors given by the wrong prediction got by the similarity of some actions.

Consequently, it was decided to represent the categories resulting from the union of several categories with a simple and explanatory name. For example, the union of dusting, mopping and vacuuming has been called "housekeeping", in fact the section “E” of the IADL has this name.



	A	B	C	D	E	F	G
1	**CATEGORIES**					**CATEGORIES MERGED**	**CATEGORIES (final name)**
2	calling					calling/telephoning	telephoning
3	cooking					cooking	cooking
4	dining					dining/eating	eating
5	dressing					dressing	dressing
6	drinking					drinking	drinking
7	dusting					dusting/mopping/vacuuming	housekeeping
8	eating					reading/studying/writing	reading-writing
9	mopping					sewing	sewing
10	reading					smoking	smoking
11	sewing					socializing	socializing
12	smoking					standing	standing
13	socializing						
14	standing						
15	studying						
16	telephoning						
17	vacuuming						
18	writing						

Figure 13 The filtered categories were merged if the action was similar.

As it can be notice all these actions are related to only two of the eight domains, the Activities of Daily Living (ADL) and the Instrumental Activities of Daily Living (IADL).

This because the remaining domains have elements for which the recognition of actions through video fails to be supportive. Such as body measurements, questions about how the patient health or the ability to answer or not to personal questions.

At the end 11 categories were selected on which to build the model: calling, cooking, eating, dressing, drinking, housekeeping, reading-writing, sewing, smoking, socializing, standing.

4.2 Pre-processing

Once the classes have been chosen it is necessary to retrieve the data. To do this, different Python scripts were needed in order to deal with: interaction with the zip file that contains the data, interaction with the filesystem (moving and saving files), interaction with the video and interaction with the images.

Several scripts have been created, each with a specific function, which together form a pre-processing pipeline. In this way it is possible to vary parameters (such as the list of categories to be extracted) without affecting the operation of the others and ensuring modularity between the various scripts.

4.2.1 Video extraction

After filling out a form relating to the "Terms of use" on the website where is possible to download the dataset "Moments in Time", what is obtained is a zip file of around 75.5GB which contains within the training and validation folders, both containing 339 folders that correspond to the folders that contain the videos for that class.

Working with a zip file of that size on a laptop is not comfortable because of the size that would become more than double if everything was extracted, so the first script performs the extraction only of the classes listed in a file, using the Python's zipfile module for performing the extraction.

In fact, the list generated in column A of the excel in *Figure 13* is fundamental in this phase, because it is copied and inserted in the file to which the script refers to know which classes to extract.

Then, the folders extracted from the training and test of the zip file are merged into a single folder because the split will be made directly inside the code later.

4.2.2 Video resizing and splitting

All the videos extracted from the "Moments in Time" dataset have the same specifications, they are: in .mp4 format, large 256x256, at 30fps (frames per second) and 3 seconds long.

As was anticipated at the beginning of chapter 4, the possibility of adding a new class was taken into consideration, by downloading the videos from external sources. So it was necessary to pre-process videos to get them to have the same specifics of the video taken from moments.

videos_2_resize_pad.py

```
import os
from os import listdir
from os.path import isfile, join

folder = "C:\\Users\\Matteo\\Desktop\\lay+table\\"

for file in listdir(folder):
    src_file_path = join(folder, file)
    dest_file_path = join(folder, "res_" + file)
    if not isfile(src_file_path):
        continue
    print(file)

    os.system(
        'ffmpeg -y -r 30 -i "{0}" -vf scale=256:256:force_original_aspect_ratio=decrease,pad=256:256:(ow-iw)/2:(oh-ih)/2 "{1}"'
            .format(src_file_path, dest_file_path)
    )
```

Figure 14 Script that execute the ffmpeg command over all files in a folder

The script in *Figure 14*, using ffmpeg to scale the video to a size of 256x256 keeping the proportions, placing the video in the centre of the dial and putting a black band padding in case it is necessary.

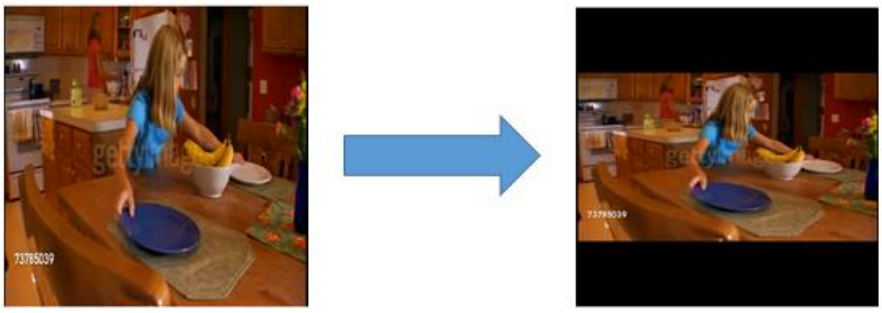


Figure 15 Avoiding image distortion by maintaining the original ratio

Videos obtained from external sources can have variable length, usually greater than 3 seconds. In trying to generate data for a new class, it was necessary to keep only 3-seconds videos that correctly express that action. For example, in a video of a few tens of seconds it is possible that during a scene change a few seconds pass without that action having occurred. The following script took care of reading the duration of the video and dividing it into 3-seconds chunks, so after it is possible to manually intervening to decide which chunk to discard.

videos_3_split_3sec.py

```python
import moviepy.editor as mp
import os
from os import listdir
from os.path import isfile, join

folder = "C:\\Users\\Matteo\\Desktop\\lay+table\\"

for file in listdir(folder):
    src_file_path = join(folder, file)
    if not isfile(src_file_path):
        continue

    video_clip = mp.VideoFileClip(src_file_path)
    duration = video_clip.duration
    video_clip.close()

    if duration <= 3.0:
        continue

    chunks = int(duration / 3)
    print("{} \t has {} '3 seconds chunks' inside {} seconds".format(file, chunks, duration))

    for n_chunk in range(0, chunks):
        new_filename = "{}_{}".format(n_chunk, file)
        dest_file_path = join(folder, new_filename)
        sec_start = n_chunk * 3
        print(dest_file_path, sec_start)
        os.system('ffmpeg -i "{0}" -ss {1} -t 3 {2}'.format(src_file_path, sec_start, dest_file_path))
```

Figure 16 Split videos into 3-seconds chunks

4.2.3 Frame extraction

Now that the videos have all been standardized with the same specifications, frames can be extracted more easily from them. At this point, a set of 18 folders corresponding to the classes selected in column A of *Figure 13*, each of which contains all the videos of the actions belonging to that category.

The purpose of the following script is to replicate the structure of the folders and insert within these the frames extracted from each video. Inside the code we wanted to leave parameterized the possibility of choosing which frames to extract from the video.

We initially evaluated the possibility of extracting the frames evenly spaced at 25%, 50%, and 75% of the video duration, following the approach described in the" Moments in Time" papers in chapter 3.4 [12].

But since the extracted frames became too many if all 3 were kept and above all they were very similar between themselves, it was decided to continue by extracting only the central frame, the one at 50% of the duration of the video. The choice to take only the central frame comes from an observation of when the frames were extracted at 25%, 50%, 75%. In fact, in most cases the middle frame was generally the most representative of the action.

It will be explained in the following chapter how it was assessed whether the frame was representative of the action and therefore could be considered, or vice versa was a useless data.

videos_5_extract_frames.py

```python
import os
from os import listdir
from os.path import isfile, join, isdir, exists, splitext
import cv2
import sys

src_folder = "C:\\Users\\Matteo\\Desktop\\tesi-moments\\extracted_videos_v0.2\\Moments_in_Time_256x256_30fps\\"
dest_folder = "C:\\Users\\Matteo\\Desktop\\tesi-moments\\extracted_frames"

n_folders = len(listdir(src_folder))

# src_folder is supposed to have folders (one for each class) containing videos
for f_index, dir in enumerate(listdir(src_folder)):
    src_dir_path = join(src_folder, dir)
    if not isdir(src_dir_path):
        continue
    print("{}/{} {}".format(f_index+1, n_folders, src_dir_path))

    # create the folders structure
    dest_dir_path = join(dest_folder, dir)
    if not isdir(dest_dir_path) and not exists(dest_dir_path):
        os.makedirs(dest_dir_path)

    for file in listdir(src_dir_path):
        src_file_path = join(src_dir_path, file)

        cap = cv2.VideoCapture(src_file_path)
        if not cap.isOpened():
            print("could not open :", src_file_path)
            sys.exit()

        n_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT)) # cap.get(7)
        width = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
        height = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
        fps = cap.get(cv2.CAP_PROP_FPS)
        print("frames: {}\t W: {}\t H: {}\t fps: {}".format(n_frames, width, height, fps))

        duration = float(n_frames) / float(fps)  # in seconds
        #print("Duration:", duration)

        # # another way to get the duration
        # cap.set(cv2.CAP_PROP_POS_AVI_RATIO,1)
        # print(cap.get(cv2.CAP_PROP_POS_MSEC))

        if duration > 5:
            print("ERROR: video with duration > 5sec.  The script is supposing to have video of 3sec")
            sys.exit()

        temp_dest_file_path = join(dest_dir_path, splitext(file)[0])

        frames_to_extract = [0.5] ## [0.25, 0.5, 0.75] # extracting 3 frames at 25%, 50% and 75%
        for index, x in enumerate(frames_to_extract):
            dest_file_path = "{}_{}.jpg".format(temp_dest_file_path, index)

            frame_to_extract = int(n_frames * x)
            # print("Extracting frame", frame_to_extract)
            cap.set(1, frame_to_extract)  # e.g. cap.set(1, 100) extract 100th frame
            ret, frame = cap.read()
            cv2.imwrite(dest_file_path, frame)
```

Figure 17 Frames extraction using OpenCV Python library

The position in which to extract the frame is determined considering the number of frames in the video, ensuring to be as precise as possible. The solution accepts a variable number of percentages in the frames_to_extract variable.

4.2.4 Manual data cleaning

After extracting frames from the videos and merging some classes as foretold in chapter 4.1.1, it is necessary to get an idea of how the images are distributed inside folders.

Using a simple script, taking in input the root folder that contains all the class-folders with the related videos, the list of classes is shown with the number of videos present and some statistics related to folders.

```
calling-telephoning      4819
cooking      4341
dining-eating    5542
dressing     654
drinking     3385
dusting-mopping-vacuuming    4464
reading-studying-writing     6181
sewing   1427
smoking      4047
socializing      2984
standing     5031

Number of classes: 11
AVERAGE number of images each class: 3897.7
TOTAL number of images: 42875
Class with MINIMUM number of images: 654
Class with MAXIMUM number of images: 6181
```

Figure 18 A brief overview of stats in folders

As can be seen in *Figure 18*, the number of videos is very varied among the classes formed, the class with the highest number of videos is about 9 times the one with the least number of elements.

It is possible to say that dataset is currently unbalanced, so the classic approaches can be used to bridge this gap, such as:

- collecting more data for the classes that have less
- doing oversampling, by randomly increase the number of observations which are just copies of existing samples. This ideally gives us sufficient number of samples to play with, but may lead to overfitting to the training data
- doing undersampling, by randomly delete the class which has sufficient observations so that the comparative ratio of two classes is significant in our data. Although this approach is really simple to follow but there is a high

possibility that the data that we are deleting may contain important information about the predictive class.

- Doing data augmentation, so generating new data by flipping, rotating, scaling, cropping, translating, adding Gaussian noise the original data, or even using Advanced Augmentation Techniques as Conditional GANs [24]

But there is another problem, probably more relevant than that of the number of data, that is the one related to data quality.

In fact, many of the videos that were extracted are not representative of the action within the defined type of problem, that is the action recognition to estimate the ADL of an elder person inside a home.

For example:

- among the videos of the "smoking" action there were also erupting volcanoes, smoke coming out of pans during cooking or fires that produced a lot of smoke.
- the “feeding” videos also showed an animal eating grass, or a bird feeding small birds.
- some extracted frames are not representative of the action because the extracted instant in the middle of the video showed an object but not the action, or just a scene change or even a black screen

In order to deal with those problems, it was decided to create a script that is supportive in the data cleaning phase, it consists in the creation of a set of images for each action that is representative of the action, manually avoiding incorrect data.

it was chosen to hold 500 images for each class, in fact this number is less than the minimum number of images between classes, so every class can afford to reach this quota, in fact taking a look at the complete set of images it was noticed that no more than a 10% can be considered “not representative”.

Then the script creates, by replicating the folder structure, a random selection of 500 images, afterwards by manually exploring the folders and looking at the retrieved frames it is possible to delete those that are considered invalid.

Later re-executing the script, a number of missing images are randomly inserted into the folder to reach the 500 quota. Obviously it is an iterative process that requires more steps to reach the goal.

videos_6_extract_data_for_model.py

```python
import os
from os import listdir
from os.path import join, isdir, exists
import shutil
import random
"""
-create a folder 'extracted_data_for_model'
-reproduce folders tree from 'extracted_frames'
-see if there is "n_data_each_class" elements in the folder, if not take the missing randomly.
-if choosed file already exists in folder, take another

IMPROVEMENT:
-remember deleted elements so it will not re-take those.. but there's more then 2K videos so it's ok
"""

src_folder = "C:\\Users\\Matteo\\Desktop\\tesi-moments\\extracted_frames_11"
dest_folder = "C:\\Users\\Matteo\\Desktop\\tesi-moments\\extracted_data_for_model_11_500"
n_data_each_class = 500
n_folders = len(listdir(src_folder))

for f_index, dir in enumerate(listdir(src_folder)):
    src_dir_path = join(src_folder, dir)
    if not isdir(src_dir_path):
        continue
    print("{}/{} {}".format(f_index+1, n_folders, src_dir_path))

    # create the folders structure
    dest_dir_path = join(dest_folder, dir)
    if not isdir(dest_dir_path) and not exists(dest_dir_path):
        os.makedirs(dest_dir_path)

    n_data_from_each_class = n_data_each_class
    listdir_src_dir_path = listdir(src_dir_path)

    if len(listdir_src_dir_path) < n_data_from_each_class:
        print("WARNING: this class has < {} elements. Has {} elements.".format(n_data_from_each_class, len(listdir_src_dir_path)))
        n_data_from_each_class = len(listdir_src_dir_path)

    listdir_dest_dir_path = listdir(dest_dir_path)
    if len(listdir_dest_dir_path) == n_data_from_each_class:
        print("This folder already has all elements")
        continue
    elif len(listdir_dest_dir_path) > n_data_from_each_class:
        print("WARNING: This folder already too much elements, you should delete some of those in order to have {} elements"
              .format(n_data_from_each_class))
        continue
    else: # < n_data_from_each_class
        if len(listdir_dest_dir_path) == 0:
            print("Destination folder is empty, so I won't worry about choosing {} elements".format(n_data_from_each_class))
            choosed_files = random.sample(listdir_src_dir_path, n_data_from_each_class)
        else:
            n_data_to_choose = n_data_from_each_class - len(listdir_dest_dir_path)
            print("{} elements are missing (maybe due to the removing of some wrong data)".format(n_data_to_choose))
            print("so I'll take some data that is not already in the folder")
            # retrieve the elements in listdir_src_dir_path that aren't in listdir_dest_dir_path
            non_present_elements = [x for x in listdir_src_dir_path if x not in listdir_dest_dir_path]
            choosed_files = random.sample(non_present_elements, n_data_to_choose)

        for file_name in choosed_files:
            full_file_name = join(src_dir_path, file_name)
            if (os.path.isfile(full_file_name)):
                shutil.copy(full_file_name, dest_dir_path)

```

Figure 19 A script which is supporting during the data cleaning phase

5 Building Deep learning models

In this chapter an analysis is carried out on how to build deep learning models that perform action recognition starting from images, in order to estimate the Activities of Daily Living of an elderly person in a residential context.

The previous chapter described how the dataset of images is created, now it will be reported how to build the neural networks that will take in input this data.

By analysing and studying state-of-the-art solutions to create a deep learning model, several questions emerged, regarding:

- first of all, if it is necessary to train a model from scratch or a transfer learning approach could be taken into account
- then, where to find computational resources and how much time it is needed to create a model
- consequently, what should be the total number of images to be taken into consideration, in order to guarantee the right trade-off between accuracy, computational resources and timing
- finally, it was wondered about what could be the accuracy that the model could achieve in order to know if it's performing well

Starting from these questions it was studied how to deal with the various problems. What emerged was that the training of the model from scratch would require: a much larger dataset of images than the one we created in the previous chapter and a lot of computational resources, as described in *Section 3.1.4*.

Furthermore, as described in *Section 3.1.3*, there is a pre-trained model on "Moments in Time", which is already able to recognize those actions, so it was decided to use Transfer Learning starting from that model, maintaining the "Convolutional Base" and going to train only on the "Classifier", so it can specialize in our 11 classes.

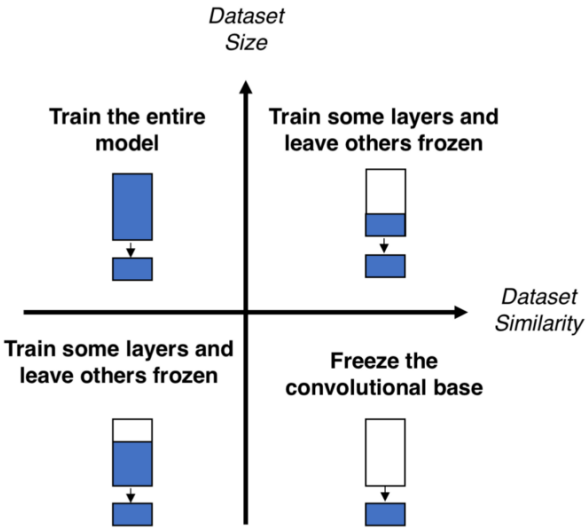


Figure 20 Decision map for fine-tuning pre-trained models [18]

This solution is applicable because there is a high dataset similarity (the data is actually coming from the same source) and the dataset size is small, also if all the pre-processed images are taken into account. This is the cheapest solutions in terms of computational costs, so it was decided to carry out the transfer learning using Kaggle Kernel, because it offers a free GPU usage inside the notebook, accessible in a really simple way. For example, enabling internet access at the notebook and activating the GPU usage is a matter of clicks.

The last question to which an answer has not yet been given, it's the one asking what accuracy it is possible to achieve, in order to set an objective. So what is an accuracy value for which it is possible to say that a good job has been done?

To try to get an idea of the attainable accuracy an Auto ML tool from IBM called NeuNetS was used.

5.1 Building three different ADL action recognition models

At first, NeuNetS has been used to create a model following the auto ML approach, because by only supplying the input data, it is received in output a ready-to-use model in Keras, which was automatically synthesize and trained, so there's no need of worrying about the processing times because it takes maximum a few hours, and about the resources because the service is hosted for free in the IBM Cloud.

In this way it is possible to examine the results returned by NeuNetS, then consider the model and its relative accuracy as a reference point for the performance of those models that will be constructed through transfer learning, so then having all models it can be selected the one with best performance.

During the creation of the models built through the use of transfer learning, it will be analysed how the performances vary by comparing the model built starting from the ResNet50 network trained on "Moments in Time" (which started itself from a pre-trained model on ImageNet) and the one trained only on the ImageNet dataset.

The second starting model has never seen the data inside the "Moments in Time" dataset nor even actions in general, since ImageNet is not a dataset designated for action recognition. It is reasonable to suppose that this model will have worse performance than the first one.

To compare the models, a set of images has been created retrieving 10 extra images for each class, different from those used during training, which have been taken from various sites and are all related to the elderly in a domestic environment, in this way the evaluation is done on images related to what the system will see during its use.

In addition to general accuracy, we also want to assess the accuracy in relation to each class, to see if there are differences between the models on performances over classes.

5.2 Auto ML approach – NeuNetS

To start using NeuNetS it is sufficient to create a free account on the IBM cloud [25] and create an instance of Watson OpenScale, which provides a company-wide environment for applications based on artificial intelligence (AI) that offers companies the opportunity to learn how to create and use AI to increase ROI, in line with your business. [26]

Within this environment there is the NeuNetS tool, which offers a graphical interface, in which no line of code is required, which guides the user (even if it's not an expert) in a few steps to create a neural network for a specific problem starting from his data.

5.2.1 Creating synthetize models

Entering Watson OpenScale, in the NeuNetS section the first screen, instructions for using this service are shown. The service has reduced interaction with the user to a minimum, in fact it is summarized in two steps: uploading the data and letting NeuNetS do the rest.

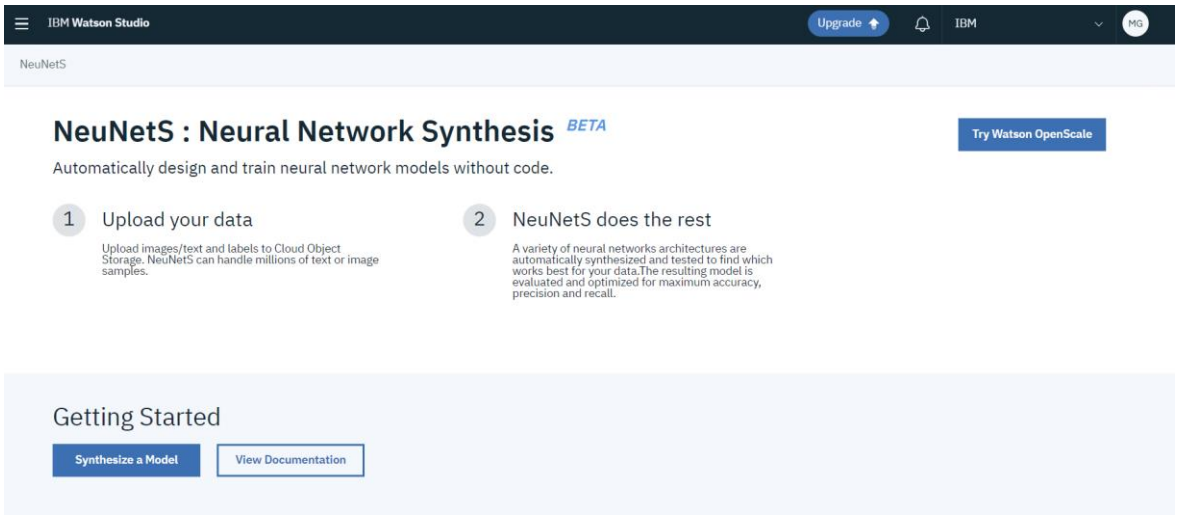


Figure 21 Landing page NeuNetS

Continuing by clicking on "synthesize a model" a screen is shown in which it must be supplied:

- a name and an optional description related to the project
- the data type, which can be images or text, in fact at the moment NeuNetS can automatically generate systems of neural models for the classification of images or text
- connection to an instance of the Watson Machine Learning [27] service which allows the deployment of the created model, in order to being able to invoke and request predictions through REST APIs, thus providing the possibility to keep the model in the cloud and not having to download and load it locally. This can be very convenient if the model is used on several projects at the same time and therefore the option to keep the "local" version is impractical if changes are made as they should be replicated on each model within the various projects.
- connection to the IBM Cloud Object Storage [28], which provides a space in which the data that will be supplied to NeuNetS must be loaded and where the resulting model with the related metadata will be saved after the model generation.

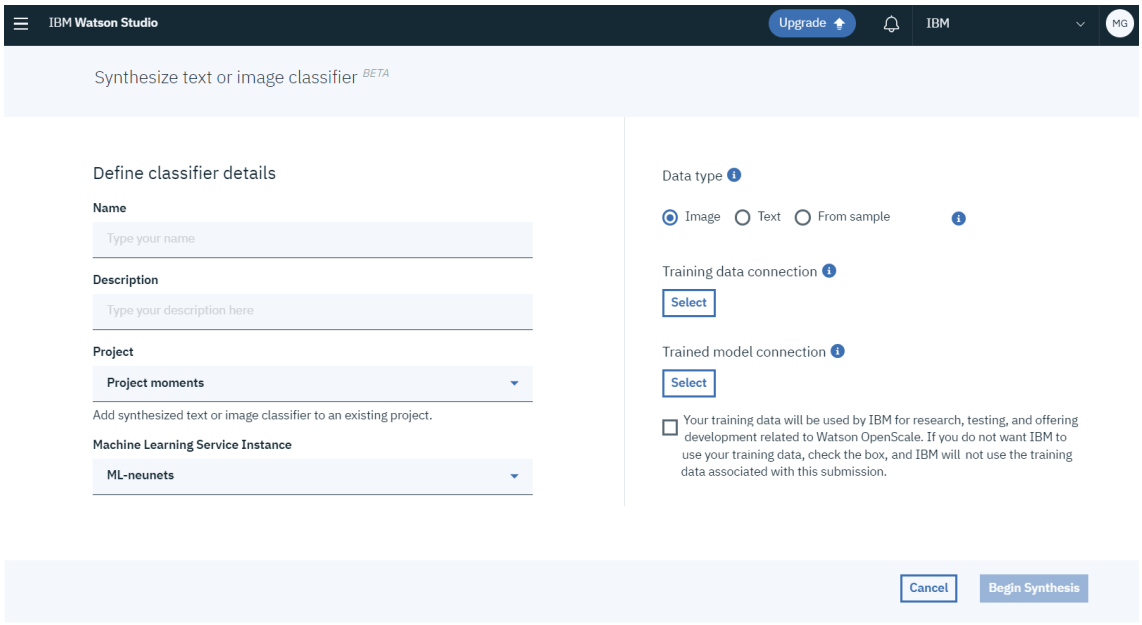


Figure 22 NeuNetS project details and connections

The data on the basis of which NeuNetS synthesize the model must be contained in a zip file called "train.zip" and in support of this a file "labels.csv" must be provided that contains the relative path of the files inside of the zip and the class that the image is expressing, separated by a comma. To generate the "labels.csv" file, a script has been created which taking in input the folder generated in the previous chapter fill the file with that information.

Two different synthesis have been launched in order to compare performances in relation on the data used:

- in the first case, it was provided a dataset containing 500 images pre-processed and filtered thanks to the manual data-cleaning phase described in section 4.2.4, so in total there were in total 5500 images divided equally into the 11 selected classes. This NeuNetS project was called “moments-11-500”.
- the other synthesis was launched without performing data cleaning on the data, so all the available frames were provided for each class. The total number of images is 42875, divided as shown in *Figure 18.* This NeuNetS project was called “moments-11-all”.

5.2.2 Analysis of synthesis results

Both synthesis took about 2 hours and the results page shows: model evaluation metrics of the model, the data distribution of the various classes, the confusion matrix, the possibility to download the importable Keras model and the possibility of deploy the model on Watson Machine Learning [27].

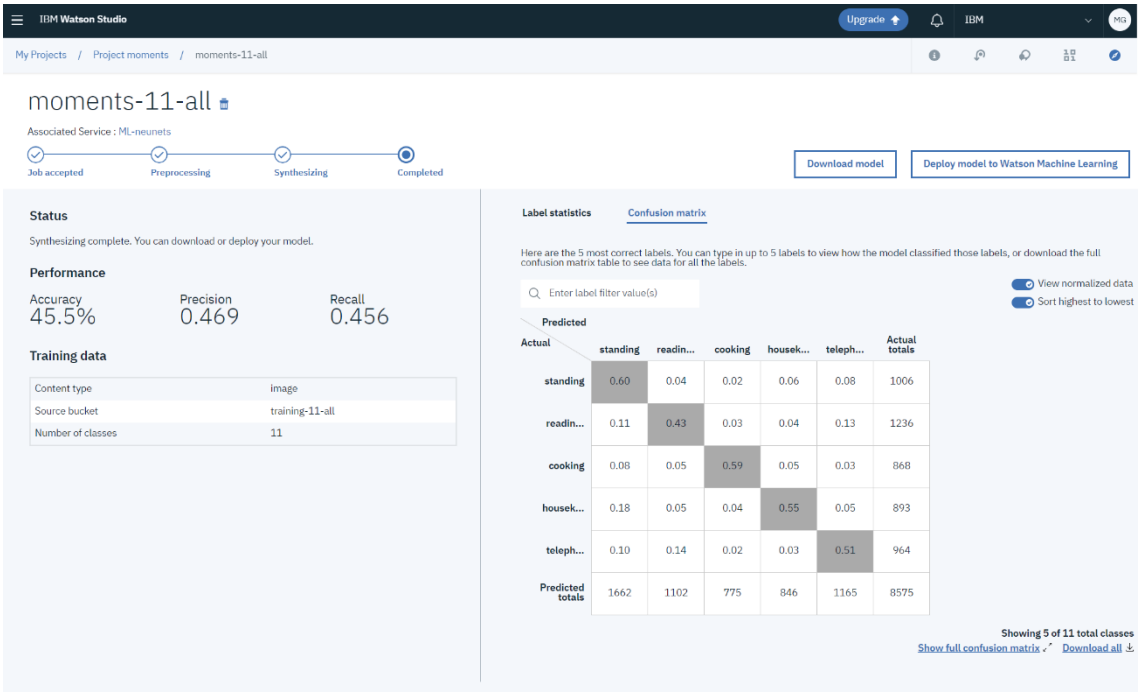


Figure 23 Result page after the synthesis

NeuNetS randomly divides the dataset into 80% for training and the rest 20% is set aside to be used as validation data (for measuring performance). Based on the score of the model on validation data, the metrics shown on the results page are calculated.

For the “moments-11-500” model an accuracy of 27.8% was reached, while for “moments-11-all” the accuracy was 45.5%.

The data that have been used for validation are not shown and above all they will certainly be different between the two synthesis created, so it is not correct to compare this two value because those are relying on different data.

For this reason, it is necessary to check the performance of the models on the test dataset that has been created, containing for each class 10 images, which were created from external sources and which were therefore not used during the model’s training.

In this way the accuracy of the two models is calculated using the same images.

5.2.3 Models score on test dataset

To evaluate the accuracy of the models, a script has been developed in order to provide those chosen metrics:

- Top-1 accuracy which evaluates the percentage of successes in identifying the action through the prediction that generated a higher score

- Top-2 accuracy which evaluates the percentage of successes in identifying the action through the first or second class predicted (assuming to have scores in a descending-ordered)

- The accuracy related to each class, so with what percentage the class has been correctly predicted

To do this, the Keras (Tensorflow) model which was designed and trained using NeuNetS, was downloaded. This download produces the file neunets_output.tar.gz. This compressed file contains two objects, keras_model.hdf5, and metadata.json.

The metadata file contains information necessary for pre- and post-processing input images to be compatible with the Keras model. Since Keras requires that classes have integer labels, this file contains the mapping necessary to find the actual label corresponding to the predicted class.

So by importing into the script the model and a library composed by some module provided in the NeuNetS documentation (in order to deal with pre-processing, post-processing and predictions of input images), all the images in the test dataset were scored and a better result was certainly obtained for the "moments-11-all" model.

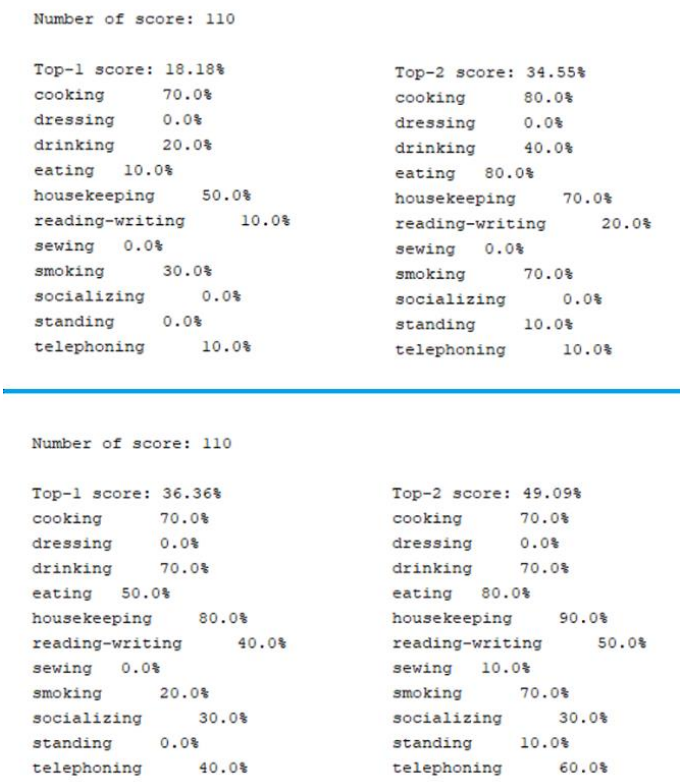


Figure 24 Performance on "moments-11-500" (top) and "moments-11-all" (bottom)

What can be noticed is that the model that has been synthesized with having all the images in input has a Top-1 accuracy of the double of the other which had 500 images per class. Furthermore, they have in common that the "dressing" and "sewing" classes are those that have the most difficulty in being identified.

This is very likely the consequence of the fact that having only 500 elements per class is too little in the case of a model synthesized using NeuNetS.

An interesting thing to note is that during the execution NeuNetS build the architecture of the neural network, in this case the generated architectures have 86 layers in the case of "moments-11-500a" and 44 layers in the case of "moments-11-all ".

The model “moments-11-all” is already a good starting point with an accuracy of 36.36% and can be improved by continuing to work on it, but it is not the subject of this thesis project, in fact what we wanted to achieve is a quick result to understand what is the reachable accuracy by using an Auto ML approach.

5.3 “Moments in Time” pre-trained model usage with PyTorch

As anticipated in section 3.1.3, there is a pre-trained model (ResNet50 pre-trained on ImageNet) on the "Moments in Time" dataset released in a GitHub repository [17] together with sample code successfully tested in PyTorch 1.0 + Python 3.6.

Since the model has been trained using PyTorch and therefore the code also uses it, this framework will be used during this part of project.

In fact, it is not possible to import the model directly into other frameworks, for example Keras, but to do so you should perform a conversion of the model using Python libraries as pytorch2keras to do so.

The pre-trained model doesn’t deal with the spatial components, but since the video is formed by a sequence of frames, an example script was provided in which an input video is accepted and the prediction is nothing but the average of some equally spaced frames (extracted using ffmpeg).

It will be shown in more detail how the prediction of a video occurs in the next chapter, when the interaction dashboard with the model will be shown.

Following, will be shown the steps required to predict an image using the pre-trained model:

1. First of all, the libraries used in the script are imported and categories are loaded from a file that list all the 339 classes, because the output of the model will be de index that has to be mapped to the corresponding row in the file.

```
1   import os
2   from PIL import Image
3   import torch
4   from torchvision import models, transforms as trn
5
6
7   # load categories
8   with open('category_momentsv1.txt') as f:
9       categories = [line.rstrip() for line in f.readlines()]
10
```

Figure 25 Step 1 - Importing libraries and loading categories

2. After that, if the file which contains weights does not exist, the curl command invoked by os.system will download it.
   Then it is loaded in the variable *model* the definition for the ResNet50 model architectures. If the “pre-trained” parameter is set to True, it returns a model

pre-trained on ImageNet, but it will not be used in this case, because the weights in the file will be loaded.

```
# load the model
weight_file = 'moments_RGB_resnet50_imagenetpretrained.pth.tar'
if not os.access(weight_file, os.W_OK):
    weight_file = 'moments_RGB_resnet50_imagenetpretrained.pth.tar'
    weight_url = 'http://moments.csail.mit.edu/moments_models/' + weight_file
    os.system('curl ' + weight_url)

model = models.resnet50(pretrained=False, num_classes=len(categories))

useGPU = 0
if useGPU == 1:
    checkpoint = torch.load(weight_file)
else:
    checkpoint = torch.load(weight_file, map_location=lambda storage, loc: storage)  # allow cpu

state_dict = {str.replace(k, 'module.', ''): v for k, v in checkpoint['state_dict'].items()}
model.load_state_dict(state_dict)
model.eval()

```

Figure 26 Step 2 - Loading model

3. Then, the input image as to be resized and normalized to make them as ResNet50 likes them (224 x 224 px, with scaled colour channels).
   Unsqueeze(0) change the size of the tensor, from torch.Size([3, 224, 224]) to torch.Size([1, 3, 224, 224]), because in the next step they will expect a 4-dimensional input

```
# load the transformer
tf = trn.Compose([
    trn.Resize((224, 224)),
    trn.ToTensor(),
    trn.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
])

# load the test image
img_filename = 'images/test5.jpg'
img = Image.open(img_filename)
with torch.no_grad():
    input_img = tf(img).unsqueeze(0)

```

Figure 27 Step 3 – Resizing and normalizing the input image

4. Finally, a single feed-forward operation is performed, then the Softmax function is applied to the tensor so that the elements lie in the range (0,1) and sum to 1.

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Figure 28 Softmax formula

The scores are sorted in a descending order and the Top-5 are showed.

```
# forward pass
logit = model.forward(input_img)
h_x = torch.nn.functional.softmax(logit, 1).data.squeeze()
probs, idx = h_x.sort(0, True)

# output the prediction of action category
print('--Top Actions:')
for i in range(0, 5):
    print('{:.3f} -> {}'.format(probs[i], categories[idx[i]]))

```

Figure 29 Step 4 - scoring the image

Below is an example of the predictions received by supplying an input image.

```
--Top Actions:
0.328 -> dining
0.310 -> socializing
0.100 -> eating
0.052 -> giggling
0.015 -> smiling
0.012 -> discussing
0.012 -> serving
0.011 -> giving
0.011 -> cooking
0.009 -> celebrating
...
```

Figure 30 Prediction on an image using the model pre-trained on "Moments in Time"

5.4 Transfer learning approach on Kaggle Kernel

Now using Kaggle Kernel and starting from pre-trained models transfer learning will be performed.

In fact, as Andrej Karpathy said: “In practice, very few people train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a ConvNet on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the ConvNet either as an initialization or a fixed feature extractor for the task of interest.” [29]

Transfer learning is the process of making tiny adjustments to a network trained on a given task to perform another, similar task.

Initially, transfer learning will be performed starting from the "Moments in Time" pre-trained model, then in order to make a comparison with a model created starting from a dataset not specialized in recognizing actions, it will be used a model trained on ImageNet.

These starting models are enough to learn a lot of textures and patterns that may be useful in other visual tasks, even as actions. That way, we use much less computing power to achieve much better result.

So, a folder was prepared with the images of each of the 11 classes splitted into training (80% of the dataset) and validation (remaining 20%) and loaded into Kaggle Kernel. So being 500 images per class, in total 4400 images are dedicated to training and 1100 to validation.
It is indicated as “validation” to differentiate it from the test dataset intended as one formed by 10 images (obtained from external sources, without taking those inside the "Moments in Time" dataset) for each class.

In Kaggle Kernel both the data and the pre-trained model on "Moments in Time" were loaded.

In the following section it will be described the code, inserted in the Jupyter Notebook hosted in Kaggle Kernel, that will freeze the weights for all of the network except that of the final fully connected layer. This last fully connected layer is replaced with a new one with random weights and only this layer is trained.

5.4.1 Transfer learning from ResNet50 pre-trained on “Moments in Time”

The model taken into account here, is a 50 layer resnet (Resnet50) trained on randomly selected RGB frames from each video in “Moments in Time” with networks initialized on ImageNet (ResNet50-ImageNet) [12].

ResNet-50 is a popular model for ImageNet image classification (AlexNet, VGG, GoogLeNet, Inception, Xception are other popular models). It is a 50-layer deep neural network architecture based on residual connections, which are connections that add modifications with each layer, rather than completely changing the signal.

The steps performed are summarized as follows:

1. Import dependencies and define variable that will be useful in the script as: the name of the directory containing the data, the names of classes and the number of classes.
   This script used the PyTorch 1.0 version and Python 3.6

```
directory_name = 'extracted_data_for_model_11_500'

import os
print(os.listdir("../input"))

directory_path = os.path.join("../input/", directory_name)
print(os.listdir(directory_path))

classes = os.listdir(os.path.join(directory_path, 'test'))
n_classes = len(classes)
print(n_classes)
```

Figure 31 Defining useful variables

```
import numpy as np
import time
import copy
%matplotlib inline
import matplotlib.pyplot as plt
import PIL.Image

import torch
from torchvision import datasets, models, transforms
import torch.nn as nn
import torch.optim as optim
```

Figure 32 Importing libraries

2. Create PyTorch data generators, because usually the images can't all be loaded at once, as doing so would be too much for the memory to handle. At the same time, there is a benefit from the GPU's performance boost by processing a few images at once. So images are loaded in batches (in this case 4 images at once) using data generators. Each pass through the whole dataset is called an epoch.

   Here, data generators are used also for pre-processing: resizing and normalizing images to make them as ResNet-50 likes them (224 x 224 px, with scaled colour channels) and to randomly perturb images on the fly during training in order to do data augmentation.
   Data augmentation is useful to show a neural network which kinds of transformations don't matter and for training on a potentially infinite dataset by generating new images based on the original dataset.

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225])

data_transforms = {
    'train':
    transforms.Compose([
        transforms.Resize((224,224)),
        transforms.RandomAffine(0, shear=10, scale=(0.8,1.2)),
        transforms.RandomHorizontalFlip(),
        transforms.ToTensor(),
        normalize
    ]),
    'validation':
    transforms.Compose([
        transforms.Resize((224,224)),
        transforms.ToTensor(),
        normalize
    ]),
}

image_datasets = {
    'train':
    datasets.ImageFolder(os.path.join(directory_path, 'train'), data_transforms['train']),
    'validation':
    datasets.ImageFolder(os.path.join(directory_path, 'test'), data_transforms['validation'])
}

dataloaders = {
    'train':
    torch.utils.data.DataLoader(image_datasets['train'],
                                batch_size=4,
                                shuffle=True, num_workers=4),
    'validation':
    torch.utils.data.DataLoader(image_datasets['validation'],
                                batch_size=4,
                                shuffle=False, num_workers=4)
}

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'validation']}
```

Figure 33 Data generators in PyTorch

3. Create the network, by importing a ResNet-50 model and loading the weights provided after the training on “Moments in Time”.
   Then by iterating on model’s parameter, all the ResNet-50’s convolutional layers are freezed, and only train the last two fully connected (dense) layers. As our classification task has only 11 classes (compared to 339 classes of the “Moments in Time” pre-trained), we need to adjust the last layer.

   In PyTorch, it should be explicitly specified if the GPU has to be load using .to(device) method. It is necessary have to write it each time we intend to put an object on the GPU, if available.

```
In [4]:
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device

Out[4]:
device(type='cuda', index=0)
```

Figure 34 Checking the available device: CPU or GPU

```
## LOAD THE MOMENTS MODEL (pretrained on ImageNet)
model = models.resnet50(pretrained= False,num_classes=339).to(device)

checkpoint = torch.load(os.path.join(directory_path, 'moments_RGB_resnet50_imagenetpretrained.pth.tar'))
state_dict = {str.replace(k,'module.',''): v for k,v in checkpoint['state_dict'].items()}
model.load_state_dict(state_dict)
```

Figure 35 Loading the “Moments in Time” pre-trained model

```
for param in model.parameters():
    param.requires_grad = False

model.fc = nn.Sequential(
               nn.Linear(2048, 128),
               nn.ReLU(inplace=True),
               nn.Linear(128, n_classes)).to(device)
```

Figure 36 Freezing all convolutional layers and add then modify the last layer

4. Train the model, so we need to pass data, calculate the loss function and modify network weights accordingly. The loss function (log-loss) and accuracy are measured for both training and validation sets.

   In PyTorch everything is explicit. In the defined training function there are nested loops, iterating over: epochs, training and validation phases and batches.
   The epoch loop does nothing but repeat the code inside, while the training and validation phases are done for three reasons:

   - Some special layers, like batch normalization (present in ResNet-50) and dropout (absent in ResNet-50), work differently during training and validation. We set their behaviour by model.train() and model.eval(), respectively.

   - There are different images for training and for validation, of course.

   - we train the network during training only. The magic commands optimizer.zero_grad(), loss.backward() and optimizer.step() (in this order) do the job.

   During the epochs iteration the model that performs better is saved.
   The training took about 25 minutes, so on average one epoch per minute

```
#model_trained = train_model(model, criterion, optimizer, num_epochs=20)

model_ft = model.to(device)

criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = optim.lr_scheduler.StepLR(optimizer_ft, step_size=7, gamma=0.1)

model_trained = train_model_1(model_ft, criterion, optimizer_ft, exp_lr_scheduler, num_epochs=2
5)
```

```
Epoch 0/24
----------
train Loss: 2.1858 Acc: 0.2489
validation Loss: 1.6252 Acc: 0.4945

Epoch 1/24
----------
train Loss: 1.8503 Acc: 0.3802
validation Loss: 1.3750 Acc: 0.5273
```

.....

```
Epoch 24/24
----------
train Loss: 1.4891 Acc: 0.5086
validation Loss: 1.2494 Acc: 0.5973

Training complete in 24m 21s
Best val Acc: 0.625455
```

Figure 37 Training the model

5. Once the network is trained, often with high computational and time costs, it has to be saved for later reuse. There are two types of savings:
    - saving the whole model architecture and trained weights (and the optimizer state) to a file
    - saving the trained weights to a file (keeping the model architecture in the code)

    PyTorch creators recommend saving the weights only. They discourage saving the whole model because the API is still evolving.

    Loading models is as simple as saving, the only thing to remember is which saving method has been chosen and the file paths.

    In Kaggle Kernel by launching an execution of the whole notebook, you get a "Output" section in which there are all the files saved inside the notebook

```
torch.save(model_trained.state_dict(), 'my_pytorch_weights.h5')
```

```
model = model_trained # !!!!
```

```
model = models.__dict__['resnet50'](pretrained=False).to(device)
#model = models.resnet50(pretrained=False).to(device)
model.fc = nn.Sequential(
               nn.Linear(2048, 128),
               nn.ReLU(inplace=True),
               nn.Linear(128, n_classes)).to(device)
model.load_state_dict(torch.load('my_pytorch_weights.h5'))
model.eval()
#print(model)
```

Figure 38 Saving and loading model

6. Make some predictions on sample test images, in order to rapidly checking if the model is working correctly and how the model is performing. In this case

   Since we are using PyTorch the code that makes the prediction is practically identical to the one described in section 5.3, even the input pre-processing part is the same.
   The only difference is that in this case there are 11 classes instead of 339.

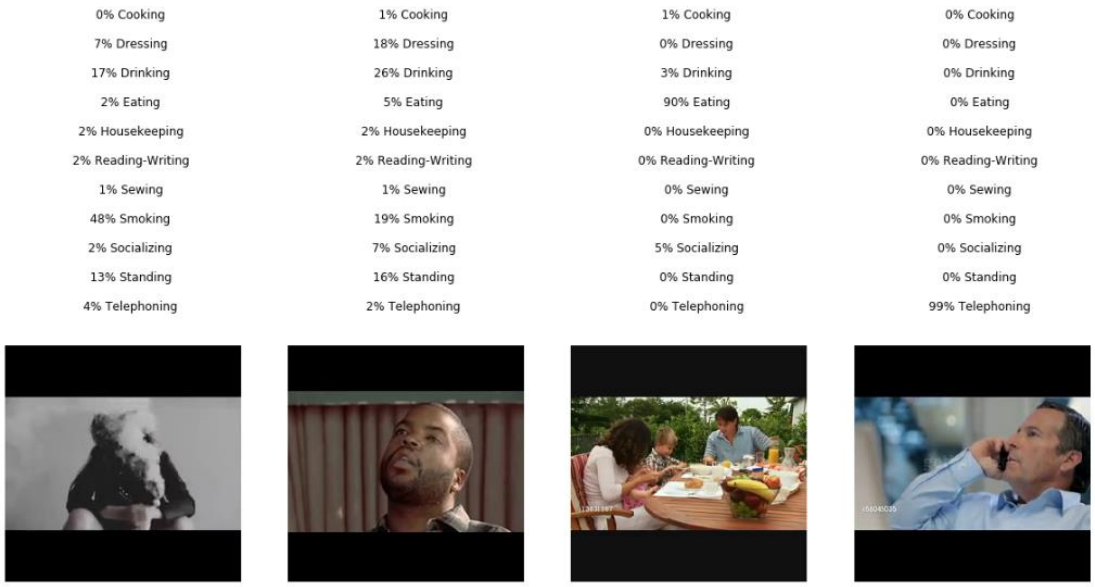


Figure 39 Predictions on sample test images

7. Analyse performance on the test dataset.
   As can be seen from the screenshot in *Figure 37*, the saved model (the one with the best accuracy during the various eras) has an accuracy of 62.6%.

   But to be able to compare the performances of the various models, as was done in section 5.2.3, the model was used to calculate the predictions on the test dataset created. And the result is very similar to the one calculated in Kaggle Kernel on validation, in fact we have a Top-1 accuracy of 64.6%.

```
Number of score: 110

Top-1 score: 64.55%
cooking      90.0%
dressing     60.0%
drinking     100.0%
eating   80.0%
housekeeping     90.0%
reading-writing      60.0%
sewing   50.0%
smoking      30.0%
socializing      50.0%
standing     0.0%
telephoning      100.0%

Top-2 score: 80.0%
cooking      100.0%
dressing     70.0%
drinking     100.0%
eating   90.0%
housekeeping     100.0%
reading-writing      90.0%
sewing   80.0%
smoking      50.0%
socializing      80.0%
standing     20.0%
telephoning      100.0%
```

Figure 40 Performance on model created using Transfer Learning, starting from "Moments in Time" pre-trained model

5.4.2 Transfer learning from ResNet50 pre-trained on ImageNet

In this case, it will be used a ResNet-50 model trained to classify images from the ImageNet dataset.

ResNet-50 is a popular model for ImageNet image classification (AlexNet, VGG, GoogLeNet, Inception, Xception are other popular models). It is a 50-layer deep neural network architecture based on residual connections, which are connections that add modifications with each layer, rather than completely changing the signal.

To do this, exactly the same code as in the previous section was used, with the only difference in point 3 when loading the model, because in this case the parameter “pre-trained” is set to True, so a pre-trained model on ImageNet is returned.

```
## LOAD THE ResNet50 MODEL (trained on ImageNet)
model = models.resnet50(pretrained=True).to(device)
```

Figure 41 Loading the ResNet-50 Model trained on ImageNet

Execution times continue to be around one minute per era, but this time the accuracy is lower, being 48.6% on validation.

It has also been tested on the test dataset and the accuracy remains in line with that calculated on the validation, in fact it is 48.18%.

```
Number of score: 110

Top-1 score: 48.18%                 Top-2 score: 69.09%
cooking      90.0%                  cooking      100.0%
dressing     50.0%                  dressing     80.0%
drinking     70.0%                  drinking     90.0%
eating   50.0%                      eating   90.0%
housekeeping     100.0%             housekeeping     100.0%
reading-writing      50.0%          reading-writing      100.0%
sewing   20.0%                      sewing   40.0%
smoking      10.0%                  smoking      30.0%
socializing      20.0%              socializing      40.0%
standing     0.0%                   standing     0.0%
telephoning      70.0%              telephoning      90.0%
```

Figure 42 Performance on model created using Transfer Learning, starting from ResNet-50 model trained on ImageNet

5.5 Model comparison

By looking at results obtained by comparing all the models accuracy (*Figure 24*, *Figure 40*, *Figure 42*) on the test dataset created, it can be noted that there are classes that are difficult to recognize in all models, for example "standing".

This problem is probably given by the fact that in this model the contemporary execution of actions is not considered, in fact there could be many images that show a person "cooking" while "standing".

The result of the model created through transfer learning starting from the pre-trained dataset on "Moments in Time" is definitely the one that performs best.

An interesting fact is that the model built starting from ResNet-50 trained on ImageNet (on which Transfer Learning has been made using a dataset of 5500 images) has obtained better performances than the one built with NeuNetS, which was supplied by 42875 images that correspond to all the data related to a class, without having been filtered.

6 Interaction with models in a web-app

This chapter describes the design and implementation choices for the development of a simple web-application that could show the behaviour of the action recognition models created in this thesis project.

The application wants to provide the possibility to upload images or videos and consequently offer mainly 2 features related to predictions for a given input:

- have an overview of the scores of the 3 models in order to compare the scores in output
- use the best model to receive the prediction of the action

A scene usually lasts a few seconds and, unless the video is short, it is very likely that there is a transaction from one action to another.

For this reason, a time chart of the video's action has been introduced, which splits the video into scenes and carries out the prediction for each of them, then shows on a graph how the actions have followed.

For simplicity, the video is subdivided into scenes of 3 seconds, based on the same principle on which the "Moments in Time" [12] dataset was built, which considers a window of 3 seconds of reasonable time for the correct expression of the action.

This type of approach may seem a bit forced, because a straight cut does not make evaluations, so there could be scenes with unclear scores. But the real use case on which the thesis project focuses is aimed at recognizing the actions of an elder person in a residential context and estimate the ADL, therefore during very long times, even 24/7 if desired. So a 3 second scene is irrelevant, in fact the actions taken into consideration are actions that usually last for minutes.

Afterwards, it was proposed by IBM to exhibit this web-application in a form of demo, inside a stand in an exhibition space.

The exhibition space is inside the Phygital HUB which is a technological showcase with the aim of stimulate and facilitate the discovery of new emerging technologies, built together by large tech companies and innovative start-ups [30].

This exhibition space is located in Casalecchio di Reno (Bologna) and is curated by Gellify, which presents itself as an innovation platform that connects high-tech B2B start-ups with traditional companies to innovate processes, products and business models through investments and thanks to the expertise of enterprise software and SaaS product experts. [31]

6.1 Building the Flask app

In order to develop the web-application, a Python web-framework called Flask has been chosen because is a Python web framework that allows to create very good solutions quickly and it is also possible to deploy this application using a service in the IBM Cloud, but this will be deepened later.

A framework is a software that abstracts complicated ideas away, and basically handles all of the business logic of what needs to be in a website, so it remains only to think about the code for the bits unique to your specific website. Flask likes to call itself a microframework, meaning that it tries to be as minimalist as possible.

The application developed, exposes these endpoints:

- the homepage
- the upload form and the list of uploaded files
- the page where the score is displayed

Now we will describe in more detail what each of these pages does.

6.1.1 Homepage and video recording

The homepage of the application has a purely informative function and in particular it shows: a quick description of the project, the figures involved in the project, links to the page containing the uploaded files and a list of references to resources used in the project.

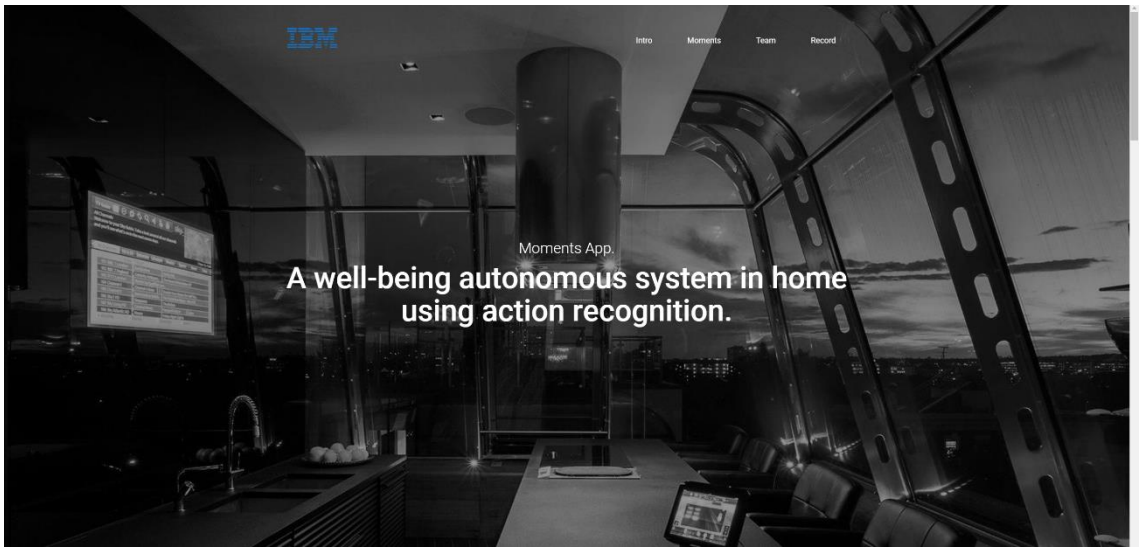


Figure 43 Home page header of the web-application

On the homepage a section has also been inserted in which you can record a video directly from the web page, using the Camera of the device that is in use.

The video is acquired in .mp4 format at 30fps with a size of 640x480, and it's a feature designed to allow a user to record and download a video in a simple way during the demo in order to testing the models by performing actions.

In order to do that HTML and JavaScript were used, in particular RecordRTC.js that is a WebRTC JavaScript library for audio/video as well as screen activity recording [32].

6.1.2 Upload Page

On the upload page there is a form through which a file can be selected and uploaded to the application.

The upload is password protected so that only those who are aware of it can upload files, since allowing users to upload videos without any control it can generate problems in terms of security.

The file extensions accepted in phase are: png, jpg, jpeg, mp4. A further check is performed on the size, allowing the maximum 16MB file.

Initially the possibility of saving the uploaded files within the IBM Cloud Object Storage service was evaluated, but since it is a demo that will show only a few dozen files, it was chosen to save these files within the application for practicality.

The management of the requests is shown, in fact if a GET request is made the list of uploaded files and the form to upload the files will be returned. While if a POST request is made by sending a file (this request is implemented in the file upload form), the application prepares to receive and save it.

```
#### UPLOAD_FILES

@app.route('/upload', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        # check if the post request has the file part
        if 'file' not in request.files or request.form['pw-upload'] != 'aaa':
            print('No file part')
            return redirect(request.url)
        file = request.files['file']
        if file.filename == '':
            print('No selected file')
            return redirect(request.url)
        if file and allowed_file(file.filename):
            filename = secure_filename(file.filename)
            file.save(os.path.join(app.config['UPLOAD_FOLDER'], filename))
            # return redirect(url_for('uploaded_file', filename=filename))

    keys_list = []
    for f in os.listdir(UPLOAD_FOLDER):
        f_path = os.path.join(UPLOAD_FOLDER, f)
        stat = os.stat(f_path)
        last_modified = os.path.getmtime(f_path)
        keys_list.append({
            'name':f,
            'size':os.path.getsize(f_path),
            'last_modified':"{}".format(datetime.datetime.utcfromtimestamp(last_modified).strftime('%Y-%m-%d %H:%M')),
            })
    keys_list = sorted(keys_list, key=itemgetter('last_modified'), reverse=True)

    remove_file_over_limit()
    return render_template("upload.html", keys_list=keys_list)
```

Figure 44 Flask endpoint for upload files and see the list of files

In order to return a web page, and HTML has to be generated, and the best way to do it using Flask is using the Jinja2 [33] template engine. To render a template the render_template() method has to be used, by providing the name of the template and the variables to pass to the template engine as keyword arguments.

At line 91 of *Figure 44*, it's possible to see how a list of information about the files is passed to the template.

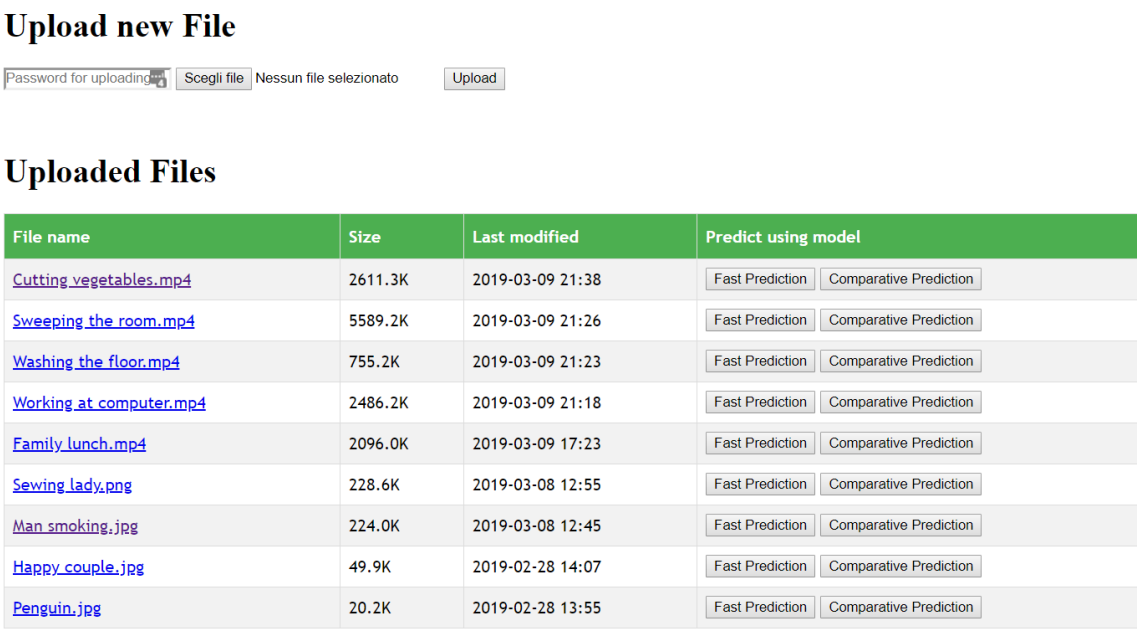


Figure 45 The upload page

6.1.3 Score of images & videos

Starting from the list of loaded files, as shown in *Figure 45*, there are two clickable buttons provided to make predictions.

When a "Comparative prediction" is performed, all 3 models are invoked and their scores are returned, while "Fast prediction" uses only the model that performs best, the one built using transfer learning starting from the pre-trained model on the "Moments in Time" dataset which generated an accuracy on the test dataset of 64.6%.

What it's showed on the score page of an image or video is a doughnut-chart for each invoked model that shows the scores and in the case of a video a temporal- chart of the shares, dividing the video into scenes of 3 seconds.

The temporal-chart predictions are always made using the model that perform the best accuracy on test dataset.

In case of a video scoring, in order to build the temporal-chart it was decided to extract 3 frames per second, so for example from a video 13 seconds, 39 frames will be extracted. After that, since the video is divided into scenes of 3 seconds each, the first 4 scenes will provide the prediction by averaging the predictions of the 9 frames related to each scene, while for the last scene (1 second long) it will be the average of the predictions of the remaining 3 frames.

Donut-charts, on the other hand, always show information about the entire video uploaded, thus averaging all the extracted frames, considering 3 frames per second. This information does not make much sense for long videos, but since in the demo there will also be short videos and describing only one scene, it was decided to show this information.

Since Flask uses Python as a language, the code within the web-app is composed of the same functions described in the previous chapters related to pre-processing (for example to extract the duration of a video with moviepy, extract the frames using ffmpeg, etc.) and making predictions in PyTorch and Keras.

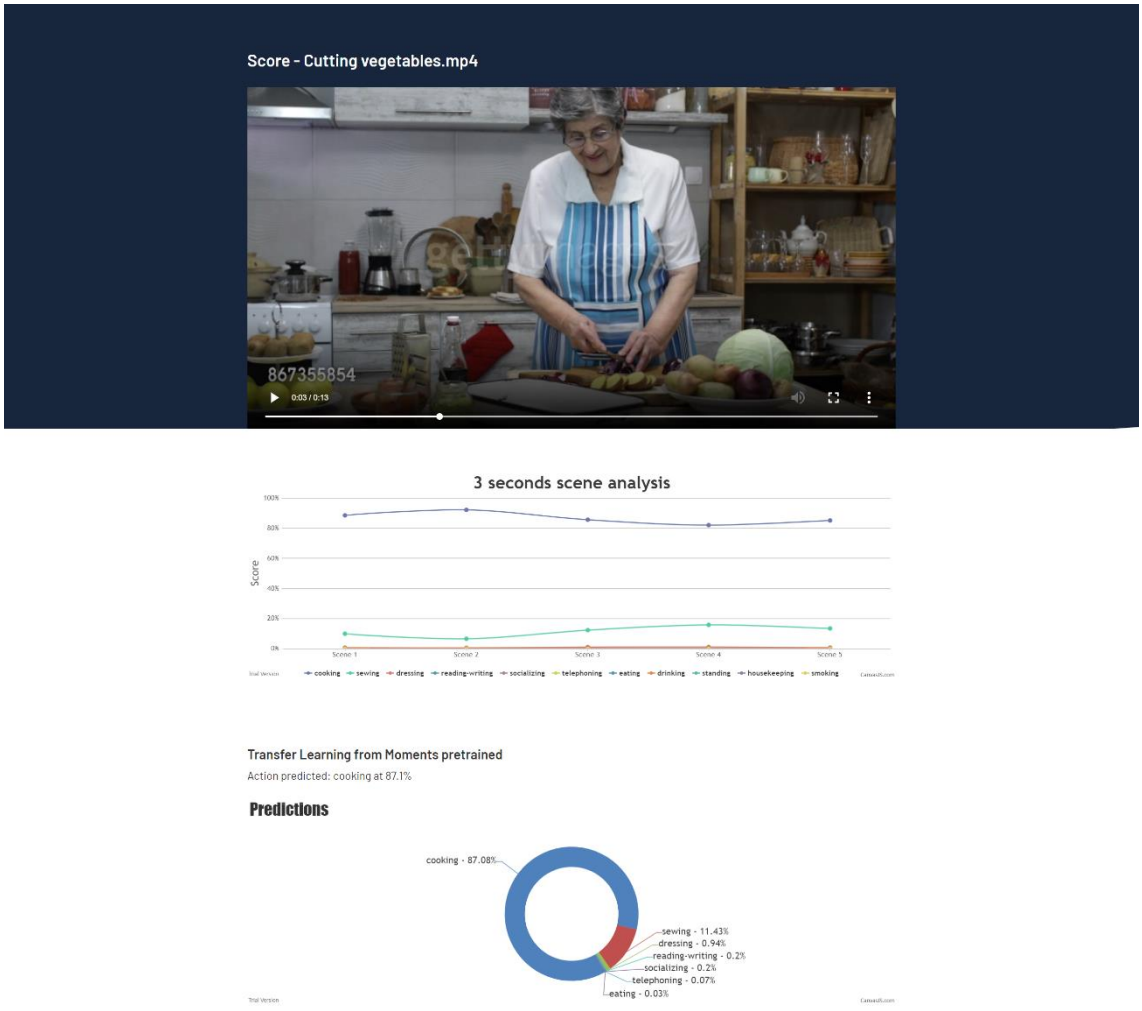


Figure 46 Score page in the web-app showing a "fast prediction" on a video

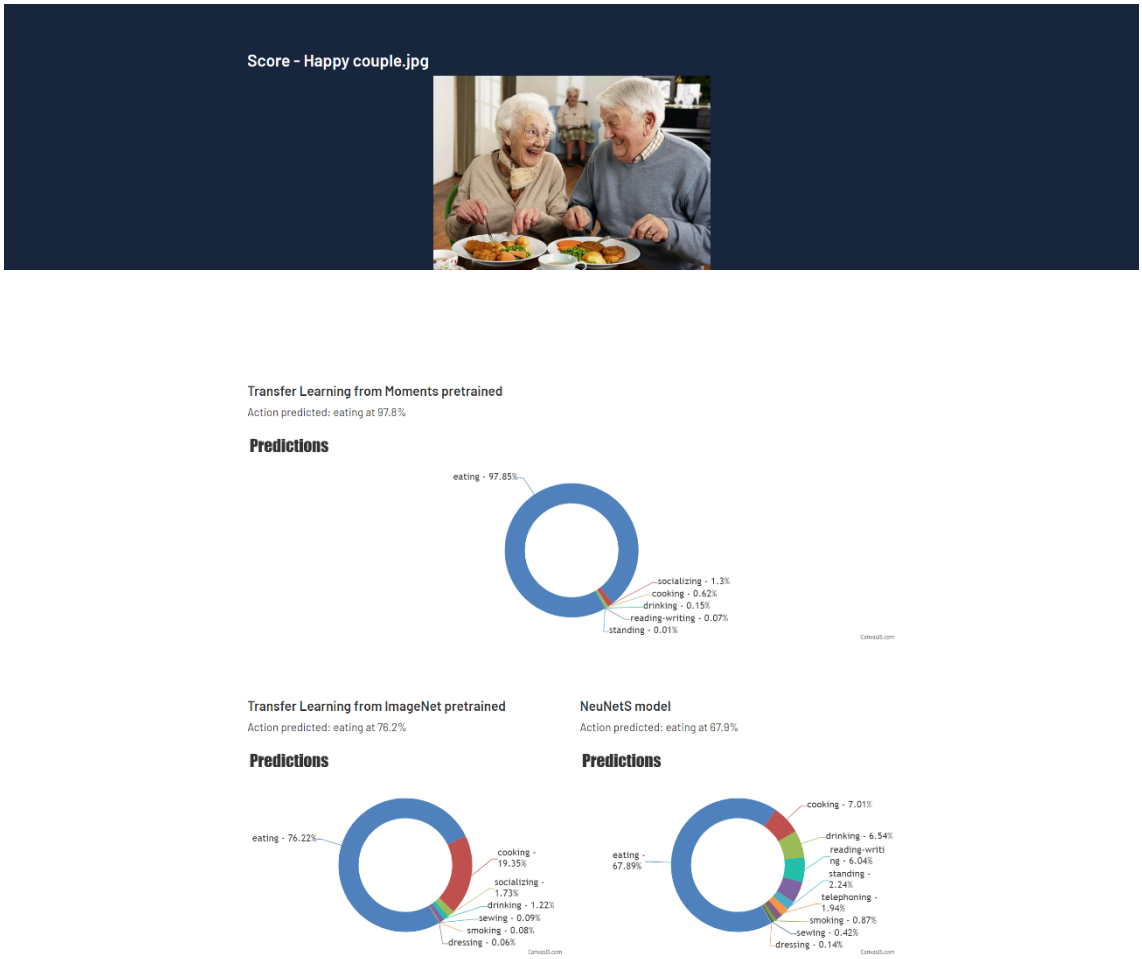


Figure 47 Score page in the web-app showing a "comparative prediction" on an image

6.2 Deploy of the web-app as a Cloud Foundry application

To make the application accessible at any time it is necessary to deploy the application on which you are working locally on a server. This way you can easily make it accessible through a link.

To do this, it was decided to use a service within the IBM cloud called Cloud Foundry [34] which allows the application to be implemented without having to configure and manage the servers manually.

It is necessary to have an IBM Cloud account, the Flask application and insert in the project folder the files that are required by Cloud Foundry to deploy, these are:

- runtime.txt which contains the Python version used, in this case python-3.6.4
- Procfile specifying the command to launch the application
- manifest.yml which contains attributes such as number of instances, disk space limit, and memory limit.
- environment.yml in which channels and dependencies to be installed are specified

Once the app is ready for deployment, all that remains is to use the IBM Cloud Command Line Interface directly from the terminal to authenticate and push the project.

After these simple steps our app is online, reachable by a link in the form appname.eu-gb.mybluemix.net.

During the application “push” phase, we came across problems related to the use of resources. In fact, loading the project with PyTorch and Keras, which are very expensive in terms of space and therefore overrun the limit of the resources available in a Lite plan.

The problem was solved by excluding PyTorch and Keras from the deployed online application, in fact since it is a demo contains dozens of files, it is convenient to pre-calculate the scores. In fact, the upload will not be allowed for security reasons, so it would only be a waste of resources to re-compute the score every time.

7 Conclusions and future works

This thesis project aimed to design and develop an Artificial Intelligence (AI) system able to recognize the development of certain actions by analysing video about residential contexts carried out by elderly people.

This system allows to estimate several parameters of two domains, specifically Activities of Daily Living (ADL) and the Instrumental Activities of Daily Living (IADL), that form the Multidimensional Prognostic Index (MPI), a predictive index of mortality not linked to a specific disease, widely accepted and validated at international level and built on data obtained through a Multidimensional Valuation (VMD) of the elderly subject.

To build the system multiple goals were achieved such as:

- Formalization of a specific business problem in the ADL recognition domain and the definition of a set of practical use cases that can be solved using AI techniques.

- Review of the literature about the MPI Index, Action Recognition dataset and the state-of-the-art of deep learning technologies to build a solution.

- Creation of a task-specific training data set deriving it from the "Moments in Time" database produced by MIT-IBM Watson AI Lab, which includes a collection of one million labeled 3 second videos, to extract the data that were needed in this project.

- Design and development of multiple Deep learning models to support the ADL action recogniton task.

- Evaluation of the obtained results by testing the models.

- Development of a web-based dashboard to easily interact and test the action recognition models.

The project produced preliminary satisfactory results regarding the creation of an ADL action recognition model from video analysis.

These results can be further improved by focusing on a number of new research activities such as:

- Extend the computer vision analysis models by considering a larger number of classes to have a wider coverage of the actions that can be recognised.

- Improve the performance of the model by expanding the training dataset collecting more data as well as improving the model accuracy by tuning parameters.

- Integrate in the model the ability to manage also the audio and temporal components, integrating it with the spatial one.

- Make the model able to recognize multiple overlapping actions that are taking the same place and time.

- Integrate object detection solutions that can provide important information about the interaction with an object during the course of the action.

- Create a multimodal action recognition environment by integrating vision-based systems with IoT systems including for examples wearable sensors and systems that allow voice interaction with the elderly person.

- Finally, starting a larger experimental phase in order to collect feedback in the field, in this case it will be useful to generate real data to improve models and also build new categories.

Bibliography and sitography

[1] L. Chen, C. D. Nugent e H. Wang, «A Knowledge-Driven Approach to Activity Recognition in Smart Homes,» *IEEE Transactions on Knowledge and Data Engineering,* February 2011.

[2] S. Volpato, S. Bazzano, A. Fontana, L. Ferrucci e A. Pilotto, «Multidimensional Prognostic Index Predicts Mortality and Length of Stay During Hospitalization in the Older Patients: A Multicenter Prospective Study,» *J Gerontol A Biol Sci Med Sci.,* September 2014.

[3] http://www.mpiage.eu/home/about-mpi, «MPI Age».

[4] J. M. Chaqueta, E. J. Carmonaa and A. Fernandez-Caballero, "A Survey of Video Datasets for Human Action and Activity Recognition," *Research Gate,* June 2013.

[5] https://neurohive.io/en/datasets/new-datasets-for-action-recognition/, «New Datasets for Action Recognition,» October 2018.

[6] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li e L. Fei-Fei, «ImageNet: A large-scale hierarchical image database,» *CVPR 2009,* June 2009.

[7] D. F. Fouhey, W.-c. Kuo, A. A. Efros e J. Malik, «From Lifestyle Vlogs to Everyday Interactions,» *CVPR 2018,* 2018.

[8] B. Bruno, F. Mastrogiovanni, A. Sgorbissa, T. Vernazza e R. Zaccaria, «Analysis of human behavior recognition algorithms based on acceleration data,» *IEEE International Conference on Robotics and Automation,* 2013.

[9] B. Bruno, F. Mastrogiovanni, A. Sgorbissa, T. Vernazza e R. Zaccaria, «Human motion modelling and recognition: A computational approach,» *IEEE International Conference on Automation Science and Engineering (CASE),* 2012.

[10] H. Pirsiavash e D. Ramanan, «Detecting Activities of Daily Living in First-person Camera Views,» *CVPR 2012,* 2012.

[11] R. Messing, C. Pal e H. Kautz, «Activity recognition using the velocity histories of tracked keypoints,» *ICCV 2009,* 2009.

[12] M. Monfort, A. Andonian, B. Zhou, K. Ramakrishnan, S. A. Bargal, T. Yan, L. Brown, Q. Fan, D. Gutfruend, C. Vondrick e A. Oliva, «Moments in Time Dataset: one million videos for event understanding,» January 2018.

[13] https://towardsdatascience.com/data-preprocessing-in-python-6f04e6c2cb70, «Data Preprocessing in Python».

[14] https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/, «What's the Difference Between Artificial Intelligence, Machine Learning, and Deep Learning?,» July 2016.

[15] https://www.mathworks.com/discovery/deep-learning.html, «What Is Deep Learning?».

[16] H. Wang e B. Raj, «On the Origin of Deep Learning,» *arXiv,* February 2017.

[17] https://github.com/metalbubble/moments_models, «Pretrained models for Moments in Time Dataset».

[18] https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751, «Transfer learning from pre-trained models».

[19] https://www.analyticsvidhya.com/blog/2017/05/gpus-necessary-for-deep-learning/, «Why are GPUs necessary for training Deep Learning models?».

[20] https://towardsdatascience.com/kaggle-vs-colab-faceoff-which-free-gpu-provider-is-tops-d4f0cd625029, «Kaggle vs. Colab Faceoff».

[21] R. Istrate, F. Scheidegger, G. Mariani, D. Nikolopoulos, C. Bekas e A. C. I. Malossi, «TAPAS: Train-less Accuracy Predictor for Architecture Search,» *arXiv,* June 2018.

[22] M. Wistuba, «Deep Learning Architecture Search by Neuro-Cell-based Evolution with Function-Preserving Mutations,» *ECML PKDD 2018,* September 2018.

[23] https://hackernoon.com/flask-web-programming-from-scratch-9ada8088fde1, «Flask Web Programming from Scratch,» May 2018.

[24] https://medium.com/nanonets/how-to-use-deep-learning-when-you-have-limited-data-part-2-data-augmentation-c26971dc8ced, «Data Augmentation | How to use Deep Learning when you have Limited Data — Part 2».

[25] https://www.ibm.com/it-it/cloud, «IBM Cloud».

[26] https://www.ibm.com/cloud/watson-openscale/, «Watson Openscale».

[27] https://www.ibm.com/it-it/cloud/machine-learning, «Watson Machine Learning».

[28] https://www.ibm.com/cloud/object-storage, «IBM Cloud Object Storage».

[29] A. Karpathy, «Transfer Learning,» *CS231n Convolutional Neural Networks for Visual Recognition.*

[30] http://www.gellify.com/phygital-hub/, «Phygital HUB».

[31] http://www.gellify.com/, «Gellify».

[32] https://github.com/muaz-khan/RecordRTC, «RecordRTC».

[33] http://jinja.pocoo.org/docs/2.10/, «Jinja2».

[34] https://www.ibm.com/it-it/cloud/cloud-foundry, «Cloud Foundry».

[35] K. Avgerinakis, A. Briassouli e I. Kompatsiaris, «Recognition of Activities of Daily Living for Smart Home Environments,» *2013 9th International Conference on Intelligent Environments,* July 2013.

[36] K. Ohnishi, A. Kanehira, A. Kanezaki e T. Harada, «Recognizing Activities of Daily Living with a Wrist-mounted Camera,» *CVPR2016,* November 2015.

[37] S. Volpato, S. Bazzano, A. Fontana, L. Ferrucci e A. Pilotto, «Multidimensional Prognostic Index Predicts Mortality,» *J Gerontol A Biol Sci Med Sci.,* March 2015.

[38] http://timdettmers.com/2018/11/05/which-gpu-for-deep-learning/, «Which GPU(s) to get for Deep Learning».

[39] A. Sood, B. Elder, B. Herta, C. Xue, C. Bekas, A. C. I. Malossi, D. Saha, F. Scheidegger, G. Venkataraman, G. Thomas, G. Mariani, H. Strobelt, H. Samulowitz, M. Wistuba e M. Manica, «NeuNetS: An Automated Synthesis Engine for Neural Network Design,» *arXiv,* January 2019.

[40] http://blog.qure.ai/notes/deep-learning-for-videos-action-recognition-review, «Deep Learning for Videos: A 2018 Guide to Action Recognition,» June 2018.

[41] https://towardsdatascience.com/deep-learning-unbalanced-training-data-solve-it-like-this-6c528e9efea6, «Deep learning unbalanced training data? Solve it like this.».

Appendix A - Multidimensional Prognostic Index

"CASA SOLLIEVO DELLA SOFFERENZA"
ISTITUTO DI RICOVERO E CURA A CARATTERE SCIENTIFICO
71013 SAN GIOVANNI ROTONDO (FG)
Dipartimento di Scienze Mediche
UNITA' OPERATIVA DI GERIATRIA

MULTIDIMENSIONAL PROGNOSTIC INDEX (MPI) *

CO-HABITATION STATUS

Does the patient live:	
Alone	
With relatives/nourse	
In institution	

MEDICATION USE

Number of drugs used	

** Pilotto A, Ferrucci L, Franceschi M et al. Development and validation of a Multidimensional Prognostic Index for 1-Year Mortality from a Comprehensive Geriatric Assessment in Hospitalized Older Patients. Rejuvenation Res 2007*

ACTIVITIES OF DAILY LIVING (ADL) *

A) **BATHING** (either sponge bath, tub bath, or shower)	
- Receives no assistance (gets in and out of tub by self if tub is usual means of bathing)	1
- Receives assistance in bathing only one part of the body (such as back or a leg)	1
- Receives assistance in bathing more than one part of the body (or not bathed)	0
B) **DRESSING** (gets clothes from closets and drawers – including underclothes, outer garments, and using fasteners including braces, if worn)	
- Gets clothes and gets completely dressed without assistance	1
- Gets clothes and gets dressed without assistance except for assistance in tying shoes	1
- Receives assistance in getting clothes or in getting dressed, or stays partly or completely undressed	0
C) **TOILETING** (going to the "toilet room" for bowel and urine elimination, cleaning self after elimination, and arranging clothes)	
- Goes to "toilet room," cleans self, and arranges clothes without assistance (may use object for support such as cane, walker, or wheelchair and may manage night bedpan or commode, emptying same in morning)	1
- Receives assistance in going to "toilet room" or in cleaning self or in arranging clothes after elimination or in use of night bedpan or commode	0
- Doesn't go to room termed "toilet" for the elimination process	0
D) **TRANSFER**	
- Moves in and out of bed as well as in and out of chair without assistance (may be using object for support such as cane or walker)	1
- Moves in and out of bed or chair with assistance	0
- Doesn't get out of bed	0
E) **CONTINENCE**	
- Controls urination and bowel movement completely by self	1
- Has occasional "accidents"	0
- Supervision helps keep urine or bowel control, catheter is used, or is incontinent	0
F) **FEEDING**	
- Feeds self without assistance	1
- Feeds self except for getting assistance in cutting meat or buttering bread	1
- Receives assistance in feeding or is fed partly or completely by using tubes or intravenous fluids	0

TOTAL _____

** Katz S, Ford AB, Moskowitz RW et al. Studies of illness in the aged. The index of ADL: A standardized measure of biological and psychological function. JAMA 1963; 185: 914-19.*

INSTRUMENTAL ACTIVITIES OF DAILY LIVING SCALE (IADL)*

A) **ABILITY TO USE TELEPHONE**	
- Operates telephone on own initiative: looks up and dials numbers, etc.	1
- Dials a few well-known numbers	1
- Answers telephone but does not dial	1
- Does not use telephone at all	0
B) **SHOPPING**	
- Takes care of all shopping needs independently	1
- Shops independently for small purchases	0
- Needs to be accompanied on any shopping trip	0
- Completely unable to shop	0
C) **FOOD PREPARATION**	
- Plans, prepares and serves adequate meals independently	1
- Prepares adequate meals if supplied with ingredients	0
- Heats, serves and prepares meals or prepares meals but does not maintain adequate diet	0
- Needs to have meals prepared and served	0
D) **HOUSEKEEPING**	
- Maintains house alone or with occasional assistance (e.g. "heavy work domestic help")	1
- Performs light daily tasks such as dishwashing, bed making, etc.	1
- Performs light daily tasks but cannot maintain acceptable level of cleanliness	1
- Needs help with all home maintenance tasks	0
- Does not participate in any housekeeping tasks	0
E) **LAUNDRY**	
- Does personal laundry completely	1
- Launders small items; rinses stockings, etc.	1
- All laundry must be done by others	0
F) **MODE OF TRANSPORTATION**	
- Travels independently on public transportation or drives own car	1
- Arranges own travel via taxi, but does not otherwise use public transportation	1
- Travels on public transportation when accompanied by another	1
- Travel limited to taxi or automobile with assistance of another	0
- Does not travel at all	0
G) **RESPOSIBILITY FOR OWN MEDICTIONS**	
- Is responsible for taking medication in correct dosages at correct time	1
- Takes responsibility if medication is prepared in advance in separate dosage	0
- Is not capable of dispensing own medication	0
H) **ABILITY TO HANDLE FINANCES**	
- Manages financial matters independently (budgets, writes checks, pays rent, bills goes to bank), collects and keeps track of income	1
- Manages day-to-day purchases, but needs help with banking, major purchases, etc.	1
- Incapable if handling money	0

TOTAL _____

* *Lawton MP, Brody EM. Assessment of older people:self-maintaining and instrumental activities of daily living. Gerontologist 1969;9:179-86.*

SHORT PORTABLE MENTAL STATUS QUESTIONNAIRE (SPMSQ) *

(Record the errors)

What is the date today? (Correct only when the month, date, and year are all correct)	1
What day of the week is it?	1
What is the name of this place? (Correct if any of the description of the location is given)	1
What is your street address?	1
How old are you?	1
When were you born?	1
Who is the president (or the Pope) now? (Requires only the correct last name)	1
Who was president (or the Pope) just before him?	1
What was your mother's maiden name?	1
Subtract 3 from 20 and keep subtracting 3 from each new number at least for 3 times (The entire series must be performed correctly to be scored as correct)	1

TOTAL _____

** Pfeiffer E. A short portable mental status questionnaire for the assessment of organic brain deficit in elderly patients. J Am Geriatr Soc. 1975; 23:433-441.*

EXTON-SMITH SCALE (ESS) *

(evaluation of pressure sores risk)

General Condition		**Incontinence**	
Bad	1	Doubly incontinent	1
Poor	2	Usually of urine	2
Fair	3	Occasional	3
Good	4	Not	4
Mental State		**Mobility in Bed**	
Stuporosous	1	Immobile	1
Confused	2	Very limited	2
Apathetic	3	Slightly limited	3
Alert	4	Full	4
Activity			
In bed all day	1	**TOTAL** _____	
Chairfast	2		
Walks with help	3	*Score 16-20: minimum risk*	
Ambulant	4	*Score 10-15: medium risk*	
		Score 5-9: high risk	

** Bliss MR., McLaren R., Exton-Smith AN. Mattresses for preventing pressure sores in geriatric patients. Mon Bull Minist Health Public Health Lab Serv 1966*

CUMULATIVE ILLNESS RATING SCALE (C.I.R.S.) *

	NONE	MILD	MODERATE	SEVERE	EXTREMELY SEVERE
1. Cardiac (heart only)	1	2	3	4	5
2. Hypertension (rating is based on severity)	1	2	3	4	5
3. Vascular (arteries, veins, lymphatics)	1	2	3	4	5
4. Respiratory (lungs, bronchi, trachea)	1	2	3	4	5
5. EENT (eye, ear, nose, throat, larynx)	1	2	3	4	5
6. Upper GI (esophagus, stomach, duodenum, biliary and pancreatic trees)	1	2	3	4	5
7. Lower GI (intestines, hernias)	1	2	3	4	5
8. Hepatic (liver only)	1	2	3	4	5
9. Renal (kidneys only)	1	2	3	4	5
10. Other GU (ureters, bladder, urethra, prostate, genitals)	1	2	3	4	5
11. Musculo-skeletal-integumentary (muscles, bone, skin)	1	2	3	4	5
12. Neurological (brain, spinal cord, nerves)	1	2	3	4	5
13. Endocrine-metabolic (including diabetes, hyperlipidemia, infections, toxicity)	1	2	3	4	5
14. Psychiatric (dementia, depression, anxiety, agitation, psychosis)	1	2	3	4	5

ILLNESS SEVERITY SCORE (CIRS-IS) mean of all single item (excluded the psychiatric item)	**COMORBIDITY INDEX (CIRS-CI)** number of items with a score of 3 or greater (excluded the psychiatric item)
_____	_____

** Conwell Y, Forbes NT, Cox C, Caine ED. Validation of a measure of physical illness burden at autopsy: the Cumulative Illness Rating Scale. J Am Geriatr Soc 1993; 41: 38-41.*

MINI NUTRITIONAL ASSESSMENT (MNA) *

A) Anthropometric Assessment

1) Body Mass Index (BMI) Weight:_____kg Height:_____cm	0 BMI <19	1 BMI = 19-20	2 BMI = 21-22	3 BMI ≥ 23
2) Mid-arm circumference (MAC) in cm_____	0 MAC<21	0.5 MAC ≤ 22	1 MAC > 22	
3) Calf circumference (CC) in cm_____	0 CC < 31	1 CC ≥ 31		
4) Weight loss (last three months)	0 loss > 3Kg	1 does not know	2 loss between 1-3Kg	3 no weight loss

B) General Assessment

5) Lives independently (not in a nursing home or hospital)			0 no	1 yes
6) Takes more than 3 prescription drugs per day			0 yes	1 no
7) Has suffered psychological stress or acute disease in the past 3 months			0 yes	2 no
8) Mobility	0 bed or chair bound	1 able to get out of bed/chair but does not go out	2 goes out	
9) Neuropsychological problems	0 severe dementia or depression	1 mild dementia	2 no psychological problems	
10) Pressure sores or skin ulcers			0 yes	1 no

C) Dietary Assessment

11) How many full meals does the patient eat daily?	0 1 meal	1 2 meals	2 3 meals	
12) Consumes: Points if: 1 yes 0 2 yes 0.5 3 yes 1	at least 1 serving of dairy products (milk, cheese, yogurt) per day yes no	2 or more servings of legumes or eggs per week yes no	meat, fisk or poultry every day yes no	
13) Consumes 2 or more servings of fruits or vegetables per day?	0 no	1 yes		
14) Has food intake declined over the past 3 months due to loss of appetite?	0 severe loss of appetite	1 moderate loss of appetite	2 no loss of appetite	
15) How much fluidi s consumed per day?	0 less than 5 glasses	0.5 5 to 9 glasses	1 more than 9 glasses	
16) Mode of feeding	0 with assistance	1 self-feed with some difficulty	2 self-feed without any problem	

D) Self Assessment

17) Do they view themselves a s having nutritional problems?	0 major malnutrition	1 does not know	2 no nutritional problems	
18) In comparison with other people of same age, how they consider their health status?	0 not as good	0.5 does not know	1 as good	2 better
			TOTALE (max 30 punti)_____	
MALNUTRITION INDICATOR SCORE: ≥ 24 = well-nourished, 17-23.5 = at risk of malnutrition, < 17 = malnourished				

** Vellas B et al. The Mini Nutritional Assessment (MNA) and its use in grading the nutritional state of elderly patients. Nutrition 1999; 15: 116-22.*

Normogram for the calculation of knee height

HEIGHT (m)	Men (18-59 years)	1.94	1.93	1.92	1.91	1.90	1.89	1.88	1.87	1.865	1.86	1.85	1.84	1.83	1.82	1.81
	Men (60-90 years)	1.94	1.93	1.92	1.91	1.90	1.89	1.88	1.87	1.86	1.85	1.84	1.83	1.82	1.81	1.80
	Knee height (cm)	6 5	64.5	64	63.5	63	62.5	62	61.5	61	60.5	60	59.5	59	58.5	58
HEIGHT (m)	Women (18-59 years)	1.89	1.88	1.875	1.87	1.86	1.85	1.84	1.83	1.82	1.81	1.80	1.79	1.78	1.77	1.76
	Women (60-90 years)	1.86	1.85	1.84	1.835	1.83	1.82	1.81	1.80	1.79	1.78	1.77	1.76	1.75	1.74	1.73
HEIGHT (in)	Men (18-59 years)	1.80	1.79	1.78	1.77	1.76	1.75	1.74	1.73	1.72	1.71	1.705	1.70	1.69	1.68	1.67
	Men (60-90 years)	1.79	1.78	1.77	1.76	1.74	1.73	1.72	1.71	1.70	1.69	1.68	1.67	1.66	1.65	1.64
	Knee height (cm)	57.5	57	56.5	56	55.5	55	54.5	54	53.5	53	52.5	52	51.5	51	50.5
HEIGHT (in)	Women (18-59 years)	1.75	1.74	1.735	1.73	1.72	1.71	1.70	1.69	1.68	1.67	1.66	1.65	1.64	1.63	1.62
	Women (60-90 years)	1.72	1.71	1.70	1.69	1.68	1.67	1.66	1.65	1.64	1.63	1.625	1.62	1.61	1.60	1.59
HEIGHT (in)	Men (18-59 years)	1.66	1.65	1.64	1.63	1.62	1.61	1.60	1.59	1.58	1.57	1.56	1.555	1.55	1.54	1.53
	Men (60-90 years)	1.63	1.62	1.61	1.60	1.59	1.58	1.57	1.56	1.55	1.54	1.53	1.52	1.51	1.49	1.48
	Knee height (cm)	5 0	49.5	49	48.5	48	47.5	47	46.5	46	45.5	45	44.5	44	43.5	43
HEIGHT (in)	Women (18-59 years)	1.61	1.60	1.59	1.585	1.58	1.57	1.56	1.55	1.54	1.53	1.52	1.51	1.50	1.49	1.48
	Women (60-90 years)	1.58	1.57	1.56	1.55	1.54	1.53	1.52	1.51	1.50	1.49	1.48	1.47	1.46	1.45	1.44

MPI - Multidimensional Prognostic Index

	Score given to each domain		
	Low (Value = 0)	**Middle (Value = 0.5)**	**High (Value = 1)**
SPMSQ [a]	0-3	4-7	8-10
ESS [b]	16-20	10-15	5-9
ADL [c]	6-5	4-3	2-0
IADL [c]	8-6	5-4	3-0
CIRS [d]	0	1-2	≥ 3
MNA [e]	≥ 24	17 to 23.5	<17
Number of drugs	0-3	4-6	≥ 7
Social status	Lives with family	Institutionalized	Living alone
Add up the scores assigned to each domain, and then divide the sum by 8		**TOTAL SCORE**	

Legend:

RISK	**Mild (MPI 1)**	**Moderate (MPI 2)**	**Severe (MPI 3)**
RANGE	**0.00 - 0.33**	**0.34-0.66**	**0.67-1.0**

[to] Number of errors
[b] Exton Smith Scale Score: 16-20, minimum risk, 10-15, moderate risk; 5-9, high risk of developing
[c] Number of active functional activities
[d] Number of pathological (score> 3)
[e] ≥ 24: satisfactory; 17-23.5: at risk of malnutrition; <17: Malnutrition