Università
della
Svizzera
italiana

Faculty
of Informatics

Bachelor Thesis

June 20, 2017

Convolutional Neural Network Visualization of a Realtime Classifier

Din Tamari

Abstract

Convolutional Neural Networks are currently being used in state-of-the-art applications ranging from general image and video recognition, natural language processing and recommendation systems, to more industry-specific and science-related fields. These models are highly complex and their inner states are normally unobserved. This project provides a general tool to visualize some of the internal states of a neural network. By using a webcam and interacting with a pre-trained model in real-time, users will be able to see how a change of input can affect the network's central representation and gain insights into its inner working. More specifically, the tool showcases the different levels of convolutional layers as they become more abstract, as well as the final fully connected layer of neuron activations. Furthermore, running the visualization through a predefined set of images may lead to identifying new patterns and provide further intuition into how the network works.

Advisor
Prof. Luca Gambardella
Assistant
Dr. Alessandro Giusti

Advisor's approval (Prof. Luca Gambardella): Date:

Contents

1 Introduction

New technologies and trends such as Social Networks, Internet of Things and Big Data, to name a few, have paved the path to a truly data-driven society. It is estimated that every day, around 2.5 quintillion bytes of data are created, with 90% of today's data being created in the last two years alone ([4] IBM, 2017). Never before has humanity had such a variety of information, that is applicable to almost every aspect of our lives, and available from every corner of the world. With such a vast amount of data at the tip of our fingers, the goal is now to maximize its value, by finding patterns and insights that can improve our way of life.
Thanks to the meteoric rise of computational power, faster and cheaper parallel processing of GPUs, seemingly infinite amount of storage space (cloud services) and an endless stream of data, Artificial Intelligence has recently boomed at an astonishing rate. The past decade alone has seen significant advances in natural language processing (communication), image and video recognition, robotics, and a myriad of industry and science related areas, mainly due to the developments in Machine Learning and Deep Learning techniques. The following introductory section will therefore give a brief description of these advanced algorithms, enabling the reader to get the basics needed to understand the project goals.

1.1 Artificial Intelligence

Artificial Intelligence (AI) was conceived in the 1950's. Originally defined as "*the study of agents that exist in an environment and perceive and act*" ([2] Copeland, NVIDIA 2016) it is a branch of computer science that deals with simulating intelligence in computers by empowering devices to perceive a situation and "think of" the optimal solution. Main research problems that AI often faces include machine knowledge, reasoning, learning and perceiving ([10] Luger, 2005 & [16] Russel, Norvig 2003). Over the years, AI has seen shifts in its techniques and models in order to overcome some of the aforementioned problems. Some of the major developments in the field were the move towards knowledge-driven and data-driven machine learning, and, most recently, deep learning with artificial neural networks. Figure 1 depicts a short overview of the methodological and technical changes that have occurred in this field since its emergence over half a century ago.

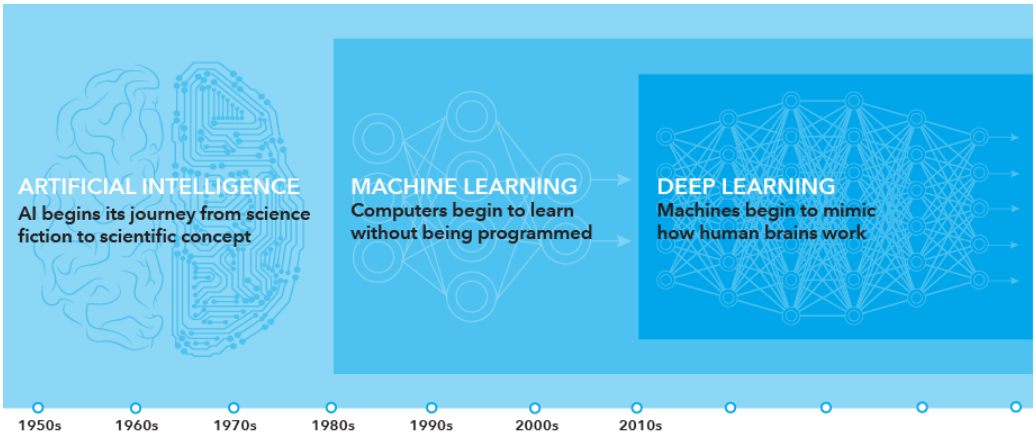


Figure 1. Shifts within the field of artificial intelligence

1.2 Machine Learning

Machine learning (ML) is a sub-field of AI. As indicated by its name, this concept refers to a computer algorithm that is able to autonomously parse and learn from data, whilst improving itself simultaneously. In other terms, it has become possible to train a machine for a specific task, given that it is provided with the necessary resources (data) and an initial set of instructions (an algorithm) on how to process, understand and learn said data. With every new piece of data that it interprets, the machine gets smarter and can make better predictions of the result of the input. Therefore, algorithms and programs are no longer statically defined at compile time, but change and improve with every training iteration.
The 1990s was a particularly interesting period for ML, where the shift from a knowledge-driven to a data-driven approach was made. It was also approximately around this time period that computational capability had risen to become tremendously powerful, as had been predicted by Moore's law. Machine learning algorithms could therefore be trained faster, using much larger data sets of information.

1.3 Deep Learning & Artificial Neural Networks

Deep learning is a sub-branch of the more general machine learning approaches. The main difference between machine learning and deep learning is that, in order to construct an algorithm able to recognize patterns, the former requires specific domain-dependent knowledge, whereas the latter can process data in its raw form by making use of a general model of learning. Moreover, most general neural networks have only one or two hidden layers, whereas deep learning artificial neural networks are renowned to having many more, hence the term "deep".

The principles of Artificial Neural Networks (ANNs) were inspired by the biology of the brain, more specifically our visual cortex. ANNs try to simulate the same concept: individual neurons (inputs) respond to unique stimuli and then connect with each other, constructing discrete layers of interconnected neurons. These layers have defined connections and directions of data propagation ([2] Copeland, NVIDIA 2016). This computational model allows a machine to think in layers, with every deeper layer representing more abstract and complex features of a problem ([1] Capital Ideas, 2017).
A neuron (or node) is the basic unit of computation within the model. It receives input from a source and then computes an output. We can generally say that a neural network is built up from three types of nodes ([6] Karn, 2016):

- **Input nodes**: these nodes correspond to the raw data provided from an external source. Combined together, the set of input nodes is referred as the **input layer**. In case of images, these nodes are the pixels representing the image as a whole. No computations are done on these nodes, rather they serve as the providers of information into the network.
- **Hidden nodes**: nodes that build up the heart of the network and do not have any contact with the outside world. Whether a one-layer or multi-layer network, all hidden nodes make up the various **hidden layers**. The main task of these nodes is to obtain data from a previous layer, perform computations and then send new data onto neurons in the next layer.
- **Output nodes**: the final set of nodes that build up the **output layer**. These nodes are responsible for the final computations and transferring data from the network into the outside world. Usually this layer is a vector of result probabilities, with each cell indicating the probability that the given input belongs to a certain class.

Before going into a more technical description, let's consider the example of classifying shapes (provided as images) using an ANN. Consider a set of characteristics that belong to a specific shape. When given a square, for example, there are certain features which we humans look for that help us determine if it is indeed a square or not. The first would be to check if the shape has four lines. If this were the case, the next steps would be to check if they were connected to each other, perpendicular at the edges and if all edges were equal in size. This creates a hierarchy of features that we look for - each being more abstract than the previous. ANNs go through the same process, dividing a bigger problem into smaller tasks, and having deeper layers responsible for more abstract features ([17] Shaikh, 2017). What is critical to understand at this point is that **deep learning ANNs are capable of identifying abstract features and attributing valid weights in an automatic way**. This is the major difference with machine learning, in which the **programmer would have to explicitly state the features that the network should consider.**

Figure 2a shows an image of a square and a circle. After a little pre-processing (changing dimensionality, transforming into grayscale and normalizing), these shapes can be fed as input to the one-hidden-layer network in figure 2b for classification. The feature that this network considers is the orientation of lines. Squares will have higher weights (values) for horizontal and vertical lines, whereas circles will have weights of higher value for curved lines. The weights are represented as the connecting lines between the nodes.
Figure 3 represents the main variables and elements that contribute to each neuron. Each neuron is the sum over all inputs from a previous layer, multiplied by their own particular weight. The associated weight of an input is relative to how important it is to the other inputs it connects to. A non-linear *activation function* is then applied to this summation, in order to introduce non-linearity to the neuron output. Activation functions are used so that the network can learn these non-linear representations, since real world data is non-linear ([6] Karn, 2016). A few examples of an activation function are for example:

- **ReLU** (Rectified Linear Unit): returns the max between 0 or the given value:

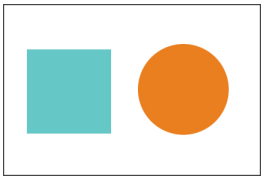
$$f(x) = max(0, x)$$

- **Sigmoid**: parses the value to be between the range of 0 and 1

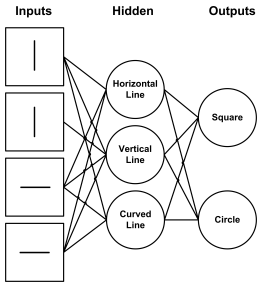
$$\sigma(x) = 1/(1 + e^{-x})$$

- **TanH**: parses the value to be between -1 and 1

$$tanh(x) = 2\sigma(2x) - 1$$



(a) Square and circle images



(b) Basic network representation given a square as input

Figure 2. Simple network using line orientation for shape classification

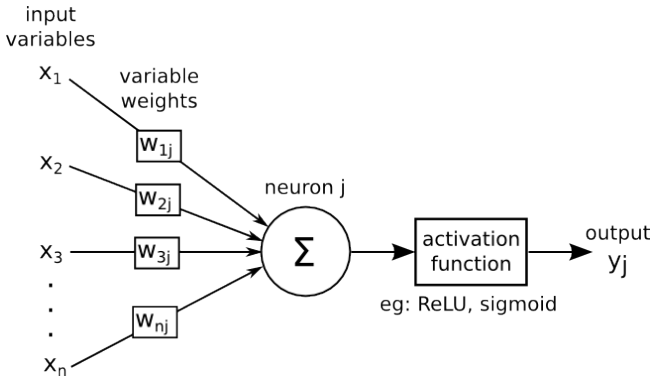


Figure 3. A single neuron representation in an ANN

The choice of an activation function depends mainly on the classification task at hand. The result of applying the non-linear function is then propagated to each neuron in the subsequent layer (or to the final output). Considering the amount of computations needed for just a single neuron, it is not surprising that deep learning, which consists of multiple layers each of which is composed of a multitude of neurons (figure 4), needs the computational power of today in order to become so powerful.

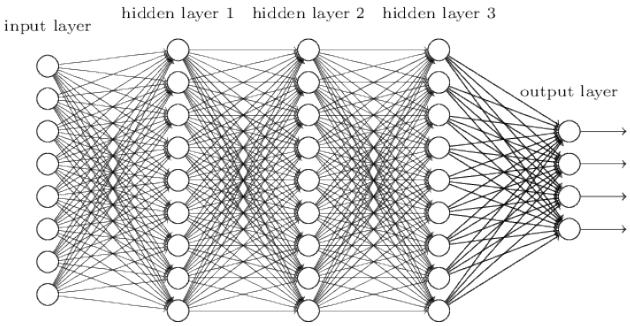


Figure 4. A deep learning ANN

ANNs are trained on large sets of images and use *back-propagation* techniques to learn from their mistakes (reduce false weight values) or reinforce their validity (increase weight values). With every new iteration of training, the networks weights are marginally tweaked in hope of obtaining better results. Referring back to our shape classification above, a network could be trained on thousands of different images of shapes, each having a corresponding label indicating what shape they actually are. This is called *supervised* learning: the network receives images and labels, and during training it adjusts the weights in certain ways depending on if it made a correct or incorrect prediction. By going through hundreds to thousands of training iterations on the data set, the weights converge to values that give the best results. These set of values are then put together to create a **model**, which can then be deployed on any new image for classification of the same task.

1.3.1 Convolutional Neural Networks

A Convolutional Neural Network (CNN, or ConvNet) is another deep learning architectures and a special type of ANN. A CNN has the distinct characteristic of being a "*feed forward network that can extract topological properties from an image*" ([15] rsipvision). A CNN uses its own form of a back-propagation algorithm to learn **filters** (or **kernels**) which are then applied to subsets of an image. The network will consider a small square of an image, referred to as the **window** or **receptive field**, to which it will then apply a filter. The purpose is to see if any receptive fields of the image match a particular filter, for example there may be a filter that matches to horizontal lines in an image.

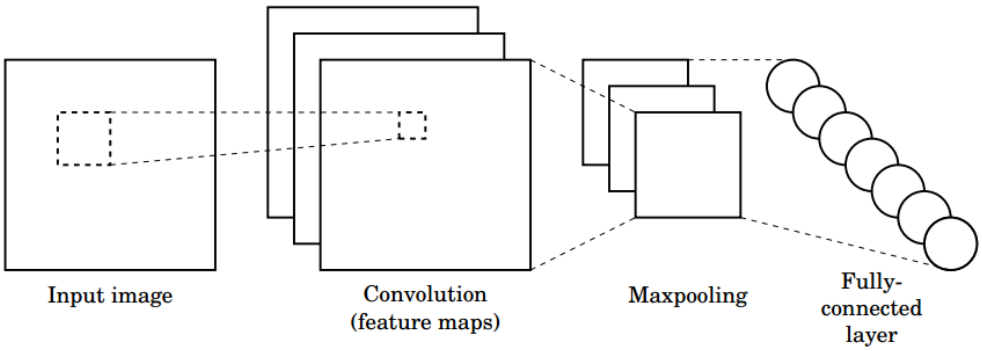


Figure 5. A Convolutational Neural Network showing the various layers

A CNN typically consists of three (main) types of layers: convolutional layers, subsampling/pooling layers and fully connected layers:

1. **Convolutional layers**: When a machine is given an image as input, it receives it as a large array of pixel values. The form of an image array usually consists of its size (width x height) and resolution (e.g. RGB). For example, the MNIST classifier used in the demo for this project accepts images with dimension 28x28x3. For the purpose of explaining convolutions, only one resolution will be considered, hence we are dealing with a 28x28x1 array of pixels. Each of these 784 cells represents the pixel intensity in the corresponding location on the image, ranging from 0-255.
   As mentioned earlier, convolutional layers use *filters* (also known as ***kernels***) on a subset of the image pixels, the *receptive field*, to recognize certain features. For example, figure 6 shows a 5x5 filter, represented as the weight lines, being applied to the top left most receptive field of 5x5 cells. The filter and receptive field compute elementwise multiplication, resulting in a final single value that is placed in the first neuron of the hidden layer. This process of **convolving** continues over the whole image, by continuously shifting the receptive field and using the appropriate filters. Once this process is complete, each neuron can be considered as a representative of the receptive field from the previous layer, leading to what are called **feature** or **activation maps**.

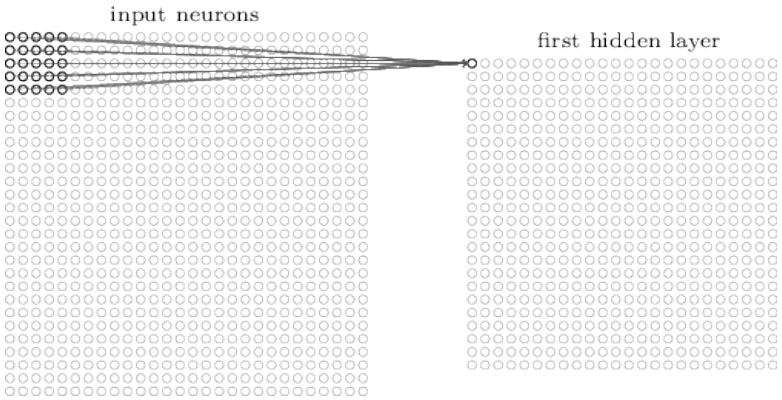


Figure 6. First convolution on top left 5x5 cells

To get a more numerical idea, figure 7 shows an example of convolving a 5x5 image with a 3x3 filter (and respectively 3x3 receptive field) ([3] Deshpande, 2016).

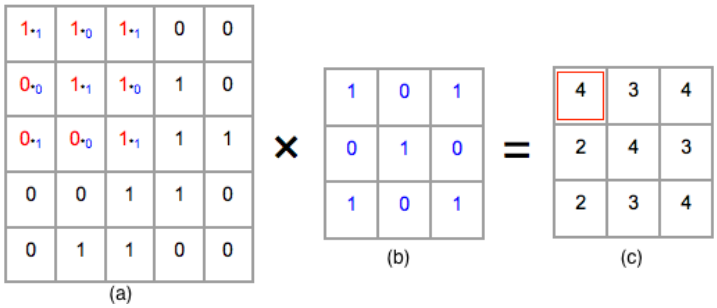


Figure 7. Convolution of a (a) 5x5x1 sample image with a (b) 3x3x1 sample filter, yielding the (c) 3x3x1 convolved feature map

In this example, the filter is applied to the top left-most receptive field. The elementwise multiplication between pixel intensities and weights results in the summation:

$$\sum_i w_i x_i = (1*1)+(1*0)+(1*1)+(0*0)+(1*1)+(1*0)+(0*1)+(0*0)+(1*1) = 4$$

The result, 4, is the first value of the convolution layer. Convolving through the rest of the image completes the feature map with all nine representative values.

A more concrete example of filters, convolutions and feature maps will now be explained in detail. Imagine that we have an edge detector filter such as the one below. Figure 8a shows the visual representation of the edge, how we humans would perceive it. Figure 8b shows the pixel representation, with each pixel representing the intensity in its location (in this case, all are 30). Now consider the digit seven in figure 8c with two different receptive fields, shown in orange boxes, to convolve with the filter.

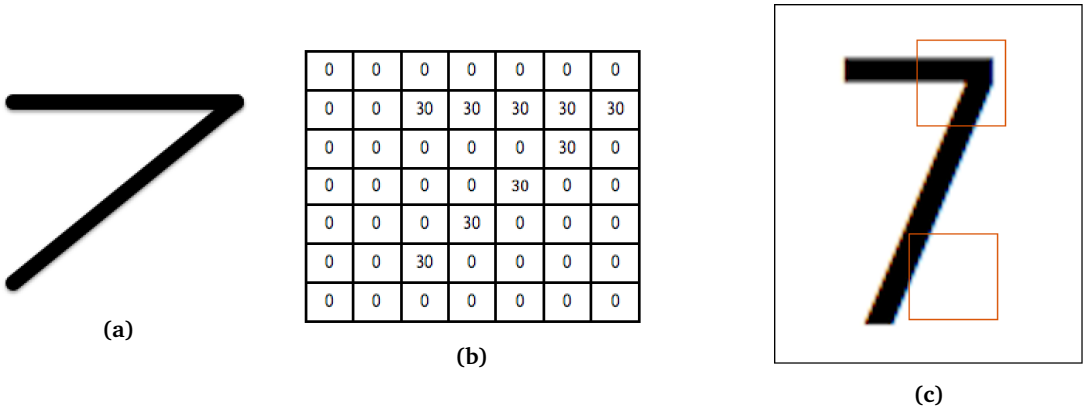


Figure 8. (a) Image of the digit seven, with two receptive fields taken for demonstration; (b) A 7x7 edge detector filter

Let's start by considering the top receptive field. Figure 9a shows an enlarged image of the field. Figure 9b, on the other hand, depicts the possible pixel intensities we would find in this field (overlaid with a transparent version of the receptive field). The summation of multiplication between the receptive field pixels and filter pixels would yield the following:

$$\sum_i w_i x_i = (30*40)+(30*40)+(30*40)+(30*40)+(30*0)+(30*30)+(30*20)+(30*0)+(30*0) = 5100$$

This is a relatively high result, which will be placed in a single neuron of the feature map. When observing the map, and seeing such a high value, we can safely deduce that a right edge exists as a lower-level feature in that location.

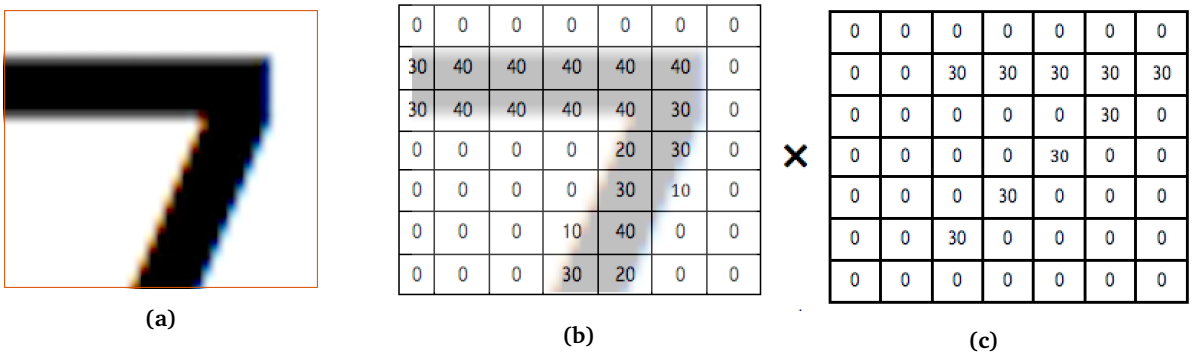


Figure 9. (a) Top receptive field enlarged; (b) pixel representation of receptive field (c) right edge filter

Looking at the second receptive field of our original image 7, and going through the same process (figure 10) we obtain the result equal to 0. This will indicate that nothing that even resembles a right edge was found in that particular area of the image.

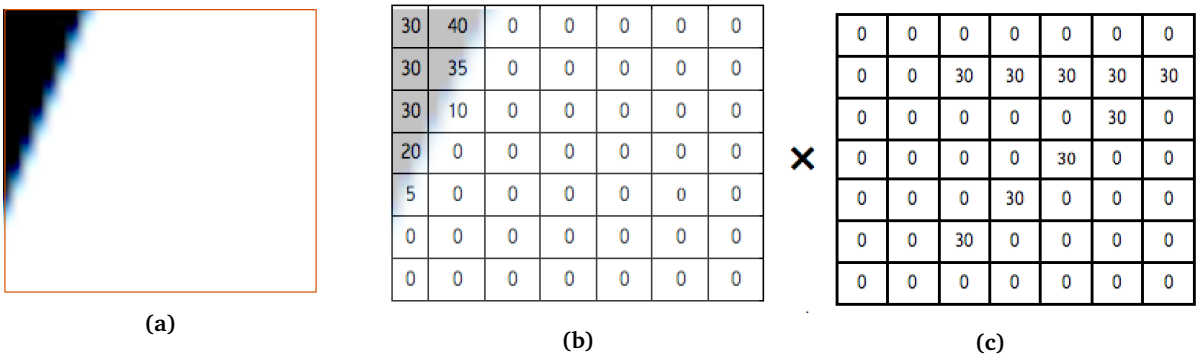


Figure 10. (a) Bottom receptive field enlarged; (b) pixel representation of receptive field (c) right edge filter

Putting it all together: a filter can be considered as a feature identifier, with features being like the ones previously mentioned (a colour, a straight line, an edge, etc.). Therefore, a filter "scans" the whole image, trying to identify what area of the image may possess a certain feature. If it matches, then the result will either be a large number indicating that the feature was identified; a low number if the patch didn't resemble the feature at all; or somewhere in between representing only a part of the feature. The first few convolutional layers will result in mostly simplistic features, whereas higher-level layers will have more abstract features.

It's important to note that there are some parameters which can be chosen for how a filter convolves an image ([3] Deshpande, 2016):

- **Size**: deciding on the width * height number of cells
- **Stride**: the stride defines how the filter convolves around the image. It is the amount by which the filter shifts its position to the next receptive field. In the example above, the stride was 1. Different strides may lead to different feature abstractions, depending on the task at hand. Increased strides leads to less overlapping receptive fields and smaller spatial dimensions.
- **Padding**: padding is used to control the dimensionality between previous and subsequent convolutional layers. By adding different borders of 0-valued intensities around the input, the dimensions of the convoluted layer can either be the same or be reduced by a lesser fraction. For example, if we added a single border of zeroes around the 5x5 input of figure 7, maintained the same 3x3 filter with stride 1, then the result would be a convoluted layer having dimension 5x5. This parameter is of particular importance for the lower layers of the network, where we wish to maintain most of the data so that the network can extract the majority of low-level features. If we do not add any padding, then convolving cuts a substantial amount of information and some features may be lost.

2. **Subsampling/Pooling layers**: Pooling layers, also known as subsampling or downsampling layers, are an optional layer that can be added to a CNN architecture. There are many different techniques for pooling, with

the most popular being **maxpooling**. Maxpooling simply takes the max value out of a subset of pixel values. Figure 11 shows a simple example of maxpooling in action. The main idea is that, if a feature is identified then the subsequent layers need only know of its existence, while its exact location is not as critically important. A pooling layer will therefore be a much smaller dimension than the previous layer, which consequently means less computations. In the example, there is actually a 75% reduction of parameters and weights. A second advantage of pooling relates to the problem of **overfitting**. When training a network over a training set of data, it is possible to *overfit* the network to match the exact data provided, rather than train a network for general task-related data. This means that a network may provide results approaching 100% for the training data set, but then have much worse results with the test dataset. Pooling is therefore one indirect way that overfitting can be dealt with.

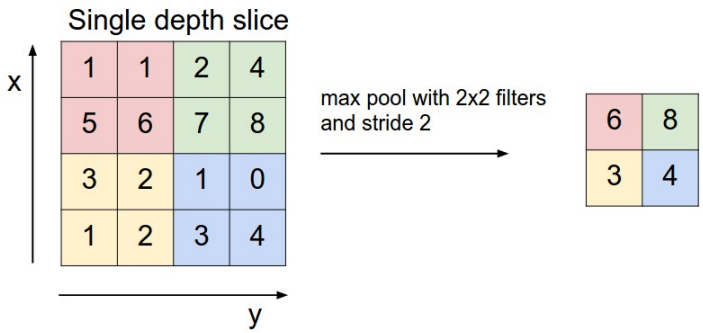


Figure 11. Maxpooling of a 4x4 image patch with a 2x2 filter and stride 2

3. **Fully Connected layers**: The main responsibility of a fully connected layer is to map the input it receives to a probability score for every class of output. It does so by identifying the abstract features in the previous map and correlating them to a particular classification group. The fully connected layer is usually at the end of a network, since its output is the N dimensional vector representing the classification probabilities for each of the N possible classes. An example of the output of a fully connected layer for MNIST digits may be: [0, 0.12, 0, 0, 0.05, 0, 0, 0.83, 0, 0], indicating that the input image has 12% probability of being a 1, 5% probability of being a 4, and 83% of being a 7.

While the previous section has delved into the main elements of CNNs with some standard examples, imagining what a classification task of a thousand different objects with thousands of abstract features may look like remains a difficult endeavor. The power of Deep CNNs comes from their ability to discover the critical abstract features of a classification task without the use of human domain expertise. By just providing a network architecture and training through a large enough supervised set of raw data, the final model will have adjusted the networks weights and be able to classify any one of the thousand objects, with a high probability result.
An example of what abstract features a CNN can pull out from a complicated task is demonstrated in figure 12, which shows three hidden layers. The first of these layers tries to learn low-level features like lines and edges while the second layer searches for more abstract, mid-level features which in this case include various parts of a human face. Finally, the third layer has high-level features which resemble actual faces. (Shaikh, 2017) This network, once trained, will be able to identify faces and probabilistically assign them to particular individuals.

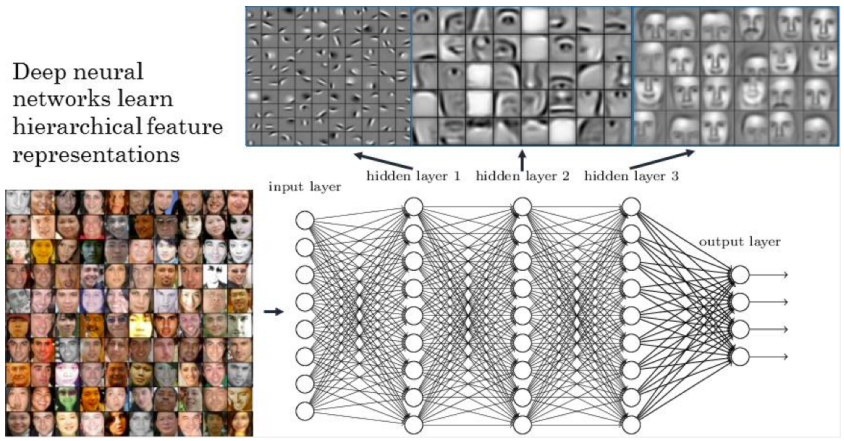


Figure 12. Deep learning CNN face example

1.4 Motivation

Deep learning CNNs have made breakthroughs in object detection, classification and recognition in both still images and videos ([8] Krizhevsky, Sutskever, Hinton, 2012). It has beaten records involving both image and speech recognition, and has ground breaking implications for other areas of interest ranging from self-driving cars, to more scientific applications such as cancer detection in MRI scans, reconstruction of brain circuits and analysing particle accelerator data to name a few ([9] Lecun, Bengio, Hinton, 2015). This technological disruption has proven to be very powerful, mainly thanks to these highly complex and abstract computational models. Considering the fact that these models are actually able to teach themselves and consequently alter their parameters and algorithm, perceiving what is actually going on in the heart of the network has become a very difficult and ambiguous task. The visualization tool that this project has created intends to make CNNs more intuitive, by shedding light on the inner states of a network as the input to the classifier changes in **realtime**.

The possibility of interacting with a trained model, whilst controlling the input, offers various advantages. This is especially true in the case of many deep learning CNN applications, since they involve classification of various types of images or videos.
For users that are new to the field of deep learning, the tool offers an initial step towards developing an intuitive understanding of the different layers, including concepts like convolutions, pooling and abstract features. By starting with a relatively simple (few layered) model, as this project has chosen for the demo, a visual aid is the ideal instrument to use for understanding what is going on in a step by step manner.
For more experienced researchers, on the other hand, this tool offers a base model to which they can apply their own trained model. Although deep learning CNNs are capable of learning their own weight parameters, they still require a learning architecture which is applied for the training to work. This architecture is provided by the programmer, who has to make decisions regarding what techniques and models are to be used. Therefore, a CNN will learn differently and output differing results based on which architecture it has been trained on. Some of these choices include the type of activation function, hyperparameters such as the filter size, stride, padding as well as what pooling methods and fully connected functions should be used. Therefore, the programmer can visualize the differences in the network after altering some of these techniques. Visualizing the layers of a network and seeing what kind of abstract features it considers may shed light on what techniques work better or worse. The tool can therefore be considered as a visual aid in cross-validation of the different hyperparameters in order to optimize the performance.

Based on these factors, a realtime visualization tool of a CNN was implemented using the open source deep-learning framework Caffe.

1.5 State Of The Art

Visualization techniques are tools used by both computational and data scientists to better understand and make sense of large datasets. An effective strategy for visualizing data is to represent each data instance as a two-dimensional point in a graph. By plotting all data instances, a grouping is formed with similar instances clustering together, and dissimilar instances being further apart from each other. The final result resembles a scatterplot of points, with each data cluster representing an individual type of data with its unique features.

The **t-Distributed Stochastic Neighbor Embedding (t-SNE)** is a *"technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets"* ([18] Maaten). It is a technique that was first introduced in 2008 by Maaten and Hinton ([11]). The results it has produced in the area of dimensionality reduction have proven to be better than any other alternative.
t-SNE is a reduction machine learning algorithm, taking in multi-dimensional data and mapping it to either two or three dimensions. Multi-dimensional data is data that has multiple features and variables, which may be correlated with each other. The main reason for implementing a dimensionality reduction algorithm is that, although computers are very capable of processing thousands to millions of dimensions, humans simply cannot. Perceiving data points on either a two or three dimensional map comes much more naturally to the simple human mind.

On a higher level, t-SNE works by searching for patterns, all the meanwhile preserving the topology (neighborhood structure) of the dataset. ([12] Olah, 2014) The algorithm tries to find a set of neighbours around a specific point, and aims to do so in a way that all points have the same number of neighbours. Clearly, neighbouring points have a close 2D or 3D coordinate location, indicating a feature similarity.

When applied to the MNIST dataset, the resulting scatterplot, provided in figure 13, clearly identifies the different groups of each digit. The image shows 10 distinct clusters of digits (0 to 9). Every data point has been mapped to a 2D location where its image has been placed to showcase the feature similarities or dissimilarities.

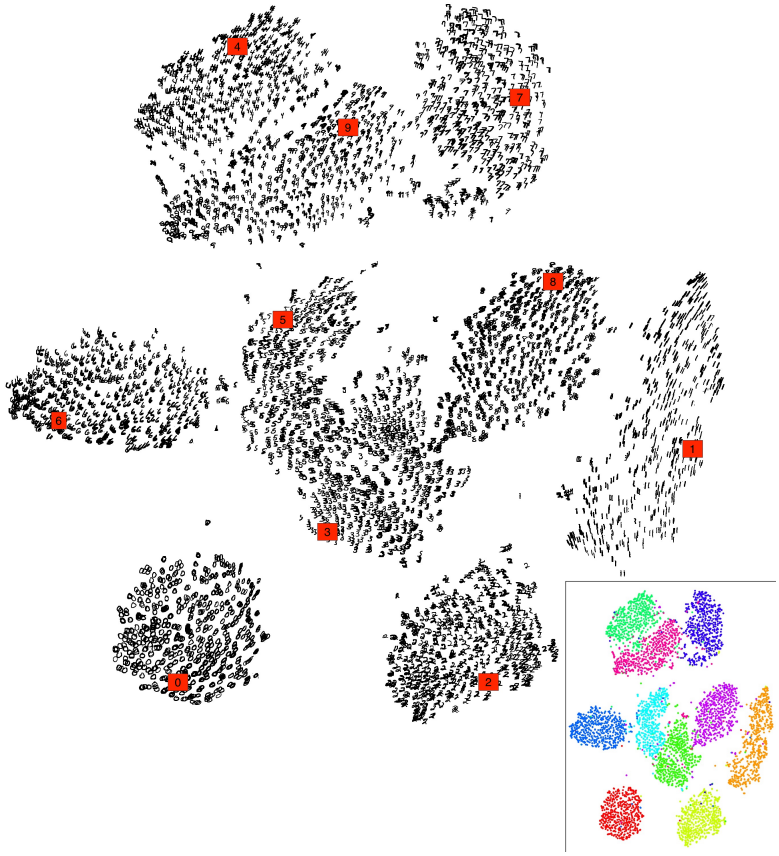


Figure 13. t-SNE result on the MNIST dataset

This visualization immediately provides us with possible insights into what digits may have similar features. For example, possible features similarities may exist between the sets (4, 9) or (3, 5, 8). Dissimilarities between digits such as (1, 6) or (0, 7) may also be inferred. Naturally, some of these assumptions could have been made even without the use of such a complex visualization tool. However, when dealing with a large dataset and not knowing

what patterns or feature similarities the different points may bring, having a visual aid such as the t-SNE has proven to be very useful. Figure 14 shows the result of applying the algorithm to a much larger and complex dataset: that of ILSVRC (Imagenet Large Scale Visual Recognition Challenge). This visualization took 50'000 of the validation images from the dataset, and arranged them by feature similarity using the 4096-dimensional array obtained from the seventh fully connected layer using Caffe ([7] Karpathy). Note that in the figure, only a subset of the images is represented, since the complete result would be too large to show and the single images would be unrecognizable.

Figure 14. t-SNE result on the ILSVRC 2012 dataset

2 Project Design

The project was defined by Professor Luca Gambardella, together with Dr. Alessandro Giusti (project assistant/tutor), and can be subcategorized into the following two tasks:

1. Building a deep learning classifier by using a CNN. The original classification task envisioned was that of face detection - a binary yes/no classifier depending on if an image patch contained a face or not.

2. Creating a visualization tool to showcase the inner states of the CNN. Additionally, this tool needed to work in real-time by means of a webcam, so that users could interact with it.

2.1 Initial project plan

The following project plan served as a guideline to identify the tasks that needed to be completed within a given period of time. After the first week of discussion and learning about the project, the main phases of the project and the amount of time needed for them were set using the following table (figure 15)

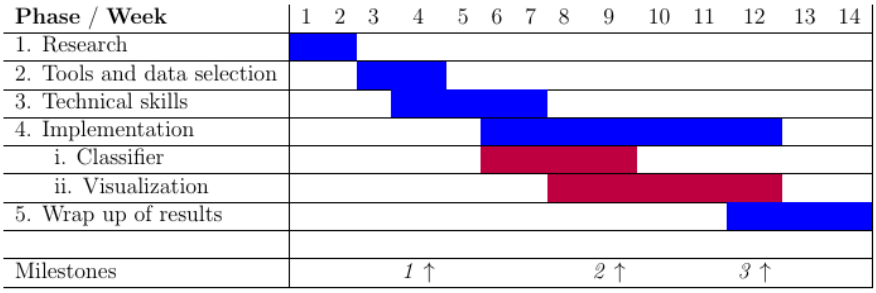


Figure 15. First plan of project, including tasks and milestones

- **Research**: gain a deeper understanding about classifiers, deep learning and convolutional neural networks. Create a brief literature review pertaining to the current face detection approaches. Completing this phase should lead to a clearer project definition.

- **Tools and data selection**: learn and compare the various open source deep learning frameworks (e.g Caffe, Brainstorm). Additionally, find the most suitable publicly-available data set to train the neural network with. Finally, get comfortable using Python image/video processing (finding the appropriate frameworks). Having gained deeper insight into the existing technologies brings us to **milestone 1**: final decisions on what to use and potential workarounds for any encountered complications.

- **Technical skills**: learning the above mentioned tools and frameworks required for implementing the classifier and its visualization. This phase may lead to the creation of code pieces used in the final product, hence it will partially be parallel to the following phase of implementation.

- **Implementation**: working with the chosen techniques and tools to develop the proposed project. Two major components will be implemented: the classifier which uses a deep neural network, and its visualization along with user interaction. These two parts have been divided in the time schedule - with the classifier being implemented first and the visualization later. Such big tasks will clearly need to be broken down into subtasks after phase 3. **Milestone 2** will come once the classifier is completed and it'll give a clear idea of what can be achieved for the visualization. **Milestone 3** will naturally come towards the end of this phase - once the implementation of the full project has been completed.

- **Wrap up of results**: writing of report and putting together all of the project deliverables (research, software). Final preparation of poster and thesis presentation.

2.2 Midterm: project plan revised

After gaining a broad enough understanding of deep learning, CNNs, classifiers and the Caffe library, milestone 1 was reached. At this point, a different continuation was agreed upon for the project. It was decided that the best way forward would be to begin the implementation on a simpler, yet just as visually interesting, task — that of digits. The MNIST dataset is usually used as the first stepping stone to learning deep learning, much like "HelloWorld" is often

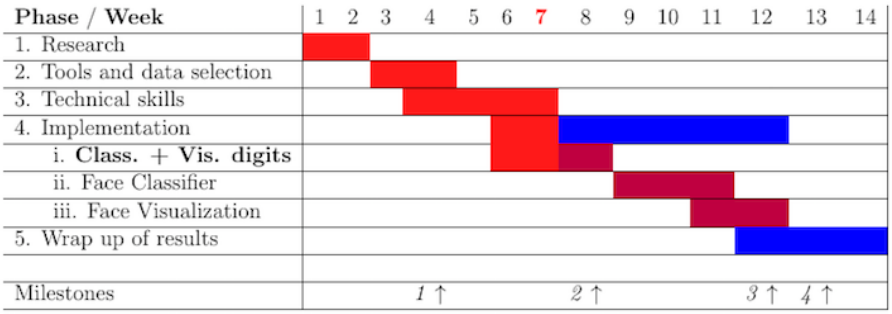


Figure 16. Updated plan of project, seven weeks in

used to as a basic example to learn a new programming language. This change of direction proved advantageous, as it allowed the project to focus on the actual visualization tool, rather than creating yet another face classifier.
The main changes of the initial project plan were visible in phase 4: implementation. The idea was to focus on the visualizing the MNIST classification in realtime, observe its execution and the problems encountered during execution, and then move on to applying it to a face classifier. These changes are reflected in figure 16.

2.3 Final decisions

With the focus of the project having shifted to the actual visualization of the network, creating a face classifier was no longer the priority of the project. Rather, the aim was to find an already available face classification model, which could then be run with the final visualization tool. Unfortunately, a simple binary face classifier was not found. From the research that was done, most of the available face models online were either face detection classifiers (which finds *all* possible faces in an image), or face recognition classifiers (which, provided a face and a list of possible people, classifies who's face it is). Therefore, the decision was taken to focus on further ways to utilize the visualization tool, for instance by running a predefined set of images (e.g. a subset of the test images) to recognize patterns in both the convolutional layers as well as the activated neurons in the fully connected layer. Additionally, a special model for age classification with faces as input was utilized to showcase the general flexibilities of the visualization tool. For more on these last few features, see section 4 'Results'.
A schematic view of the final structure of the project is defined in figure 17. A classification model (classifier) must first be trained over a multiple number of iterations on a particular data set, by using a deep learning framework (Caffe). Once the model learns the network parameters, it can be deployed on new instances of data for classification.
This visualization tool runs the trained classifier with realtime input received from the webcam. From the classifier, convolutional and fully connected layers are abstracted and visualized, along with the classification results.

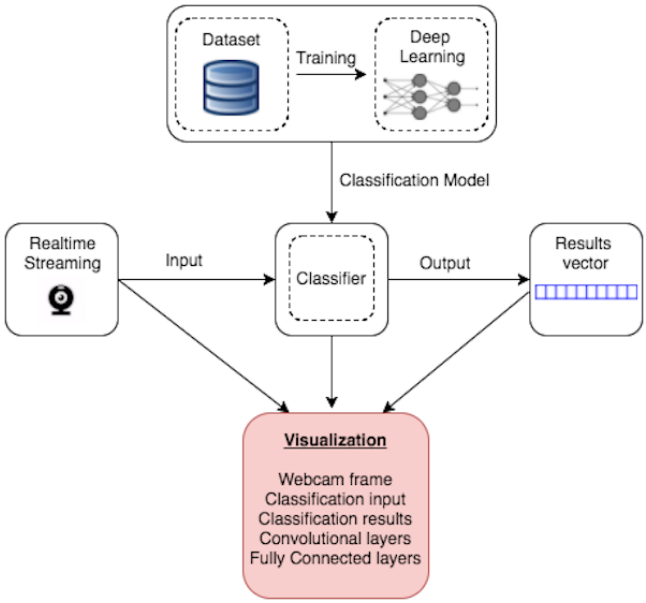


Figure 17. High level overview of project

3 Project implementation

3.1 Deep learning framework

There are dozens of open-source deep learning frameworks available on the web. A few of the most popular are shown in figure 18. All frameworks provide APIs to train, solve, and deploy a CNN. The differences lie in the core language and what bindings are provided by the framework; whether they offer GPU computation capability for speed and efficiency (crucial for training modern CNNs); as well as if they offer pre-trained models ready for deployment.

Framework	License	Core language	Binding(s)	CPU	GPU	Open source	Training	Pretrained models	Development
Caffe	BSD	C++	Python, MATLAB	✓	✓	✓	✓	✓	distributed
cuda-convnet [7]	unspecified	C++	Python		✓	✓	✓		discontinued
Decaf [2]	BSD	Python		✓		✓	✓	✓	discontinued
OverFeat [9]	unspecified	Lua	C++,Python	✓				✓	centralized
Theano/Pylearn2 [4]	BSD	Python		✓	✓	✓	✓		distributed
Torch7 [1]	BSD	Lua		✓	✓	✓	✓		distributed

Figure 18. Comparison of some popular deep learning frameworks

The framework chosen for this project was **Caffe** (http://caffe.berkeleyvision.org/). The creators of Caffe define it as a *"Convolutional Architecture for Fast Feature Embedding"* ([5] Yangqing, et al, 2014) The main benefits of using Caffe are that it is modular, extremely fast and is used in state-of-the-art applications in both academia and industry. Caffe allows the programmer to start training a network using just a few simple steps. First, by defining the model architecture and solver methods in a .prototxt (plaintext, almost JSON-like) file. Then, providing the necessary dataset, running the training on the network by using either the CPU or the GPU. Once training is completed, the result is a .caffemodel file which can be deployed (providing a deploy.prototxt file) on any new images to classify. Another reason for having chosen Caffe for this particular project is that it offers many pretrained models, which can be immediately placed into and utilized with the visualization tool.

3.2 Other technologies & libraries

- **OpenCV2**: used for the realtime capturing of frames from a webcam. The frame is captured, then goes through some image preprocessing (depending on model requirements), and is then fed as input to the classifier. A problem encountered with this was that the computational time needed for classifying a frame was greater than the speed at which new frames were captured. OpenCV2 uses a buffer to store a sequence of frames, and returns the next (instead of the most recent) frame when requested. This lead to delays for both visualizing the webcam feed as well as the corresponding classification, as they were no longer running in parallel. To fix this solution, the capturing of frames worked on a separate thread, and only captured the current frame when it was requested. The package *imutils* was used to resolve this particular issue.

- **Matplotlib**: in order to visualize all the separate elements, including the webcam's stream, the result plot and different network layers, matplotlib provided the required functionalities to make everything work in realtime. More specifically, the funcAnimation enabled the resetting of data within each plots, without having to re-render or re-draw the axes for every frame.

- **NumPy**: python package used for scientific computing and critical to classification with Caffe, as well as working with N-dimensional array objects with matplotlib.

3.3 MNIST dataset

As mentioned earlier, in order to demonstrate this projects visualization tool, the LeNet architecture was used to train a digit classification model. The model learned from the images taken from the MNIST dataset of digits. This database consists of 70'000 examples of handwritten digits divided into two sets: 60'000 for training and 10'000 for testing. All images of the dataset have been centered and normalized, having a grayscale colour sheme (black to white) and dimensions of 28x28.

Figure 19 shows a schematic view of the network (note: fully connected layers in Caffe are called **Inner Product (ip) layers**):

- input: 28x28 images
- conv1: convolution (kernel size 5, stride 1), resulting in 24x24 maps
- pool1: maxpooling (kernel size 2, stride 2), resulting in 12x12 maps
- conv2: convolution (kernel size 5, stride 1), resulting in 8x8 maps
- pool2: maxpooling (kernel size 2, stride 2), resulting in 4x4 maps
- ip1: fully connected layer with 500 outputs, followed by an in-place ReLU layer
- ip2: fully connected layer with 10 outputs
- output: vector of size 10 containing digit class probabilities

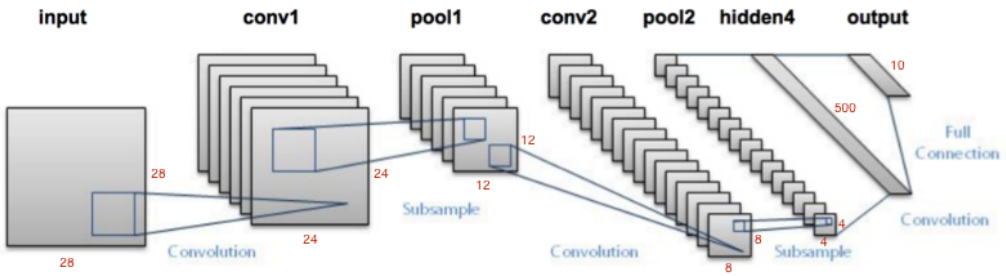


Figure 19. LeNet MNIST CNN architecture

3.4 Visualization layout

All elements pertaining to the classification process, including the camera stream, results and layers of the model are inserted into one single figure. Having everything running and visually available in parallel allows the user to identify what exactly is going on inside the classifier after changing an input. The layout can be divided into the following parts: camera and image patch, results graph, convolutional layer(s) and fully connected layer. The figures 20 and 22 in the following results section are examples of the final visualization tool.

3.4.1 Camera & image patch (box)

The camera used is either the usual internal computer webcam or externally attached. The live stream is shown in the top-left of the main figure. A square is drawn in the center of the frame, enabling the user to see what part of the image will be taken for the classification. The shot is then enlarged, and shown on the right. Note that we show the shot exactly as it is from the webcam, however before it goes through the classifier, there is some preprocessing of dimensions and colour schemes (model dependent).

3.4.2 Results graph

The usual output of a classifier, as already mentioned, is a probability vector representing the likelihood that the input belongs to a particular class. For this demo, the LeNet trained model outputs a probability vector of size ten, with each cell corresponding to the classification probability of each digit (0 to 9). The data from the vector is taken and rendered as a bar plot, with each bar representing the percentage probability value for a single digit.

3.4.3 Convolution layer

Two convolutional layers can be chosen from the LeNet architecture: *conv1* and *conv2*. In other, much larger networks, there may be many more convolutional layers to choose from. The total number of convolutions that can be chosen for display in the standard visualization is two; the reason being mostly a question of space limitations.

For transforming the array of neurons into a visually representable convolution, the project uses a function taken from a Jupyter Notebook Caffe tutorial (http://nbviewer.jupyter.org/github/BVLC/caffe/blob/master/examples/00-classification.ipynb). The function, named *vis_square*, can be found in the utils.py file.

3.4.4 Fully connected layer

The fully connected layer is a long array of neuron activations (for LeNet, the total number is 500). In the general view (figure 20), a set of these neurons are first normalized, and their different levels of activations are shown depending on the input. It is possible to identify what neurons have a high effect (a large activation) for particular digits or special features. In the *sorted neurons* layer visualization (figure 22), however, the tool takes it one step further: sorting neurons by weight importance to a chosen digit. More on this in results.

3.4.5 Layout customization

The user is able to control several aspects of the visualization layout.

- Decide on whether the layout should include the combination of convolutional layers and fully connected layers (generic), or only the fully connected layer.
- Generic visualization:
  - Convolutional layers: decide on which convolutions should be shown in convolutional plot 1 and 2.
  - For each convolutional layer, choose the set of feature maps to show
  - Fully connected layer: decide on the set of neurons to visualize, and in how many rows they should be shown
- Fully connected visualization: a part from the set of neurons to visualize, select the digit that is to be considered. Neurons will be sorted by weight value of said digit.
- Possibility of choosing a particular colour map for every layer (e.g. with grayscale, the values range between minimum black and maximum white, with gray values in between).

4 Results

4.1 General visualization

Figure 20 is the final product for the generic view, which combines two convolutional layers and one fully connected layer. From the convolutions, the learned features can be identified and understood. By using the grayscale colour map, it's possible to observe which parts of an image are particularly important for a specific feature: white pixels (high) to black pixels (low). These can be clearly seen in the features of *conv2*. The diagonal lines which are activated (white) are due to the fact that the digit 7 has a diagonal line going from top right to bottom left. Since the digit 7 does not have any vertical line, the vertical features remain quite dark, indicating that they have not been *triggered* by this particular input. By changing the input and observing the feature alterations, various further conclusions can be made about the trained classification model. Naturally, having a more complex classification task than that of digits, will provide learned features, that are more abstract and interesting, to interpret. The fully connected layer simply shows the activation level of each neuron. Here, it is only possible to observe if any specific neurons are triggered for a particular kind of input. For example, in the figure there is a single highly activated neuron, which is bright yellow. This high activation is likely to reappear when the input resembles a 7, or perhaps only a specific part of a 7 like the top horizontal line, the diagonal line, or even the edge between the two.

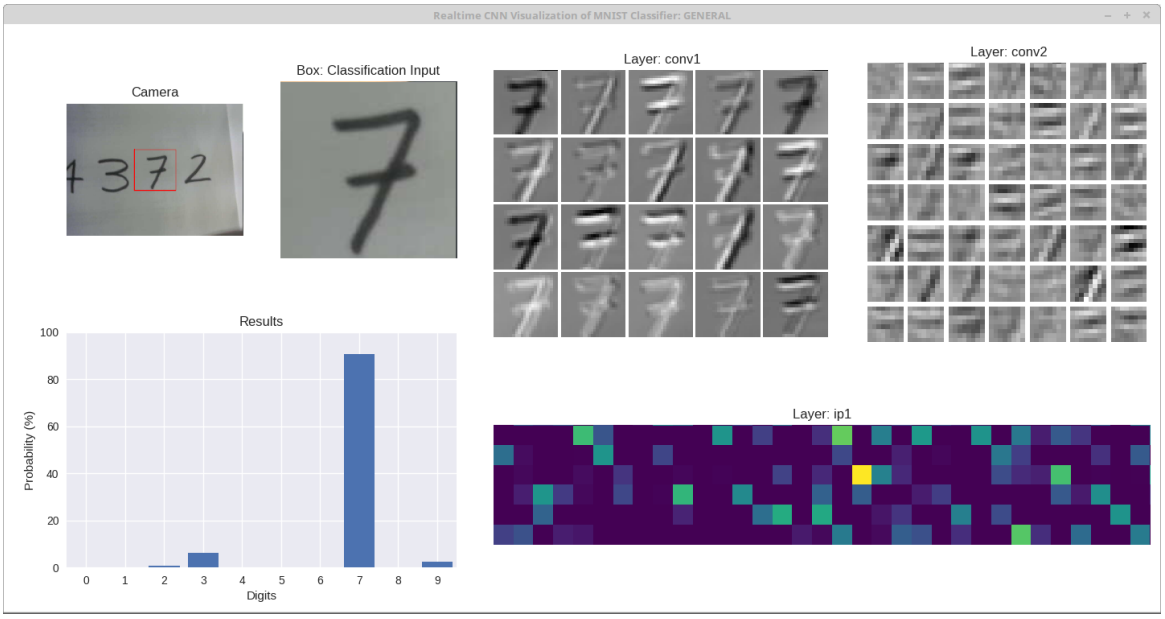


Figure 20. Visualization Tool: Generic View

4.2 Sorted neurons visualization

The output of the fully connected layer is the probabilities vector of the ten digits. Each of these probabilities is the final sum of all 500 neurons*weights multiplications of a single digit. Although the convolutional and fully connected layers that are displayed show only the neuron activations, it is also possible to abstract the set of **weights** from the model. It's important to remember at this point, that these weights are *what makes deep learning CNNs what they are*. The process of training is precisely to tweak these weights to correspond to the classification task. Therefore, when deploying a trained model, these weights are set and do not change, regardless of the input. For each digit, these weights can be extracted from the model and used to understand which neurons are important for that particular digit.

For example, consider the classification input being an image of the digit 3, and having the classification run up to the fully connected layer. Figure 21 shows the neurons of the fully connected layer as the array in the middle, that is referred to as '500 Common Neurons'. To the left there is the array of 500 weights for digit 1, and to the right the same for digit 3. Since the model is a digit classifier, there is a total of 10*500 weights for each digit, but only two are shown in this example. Now consider an arbitrary set of these neurons: n5, n83 and n202. If these three neurons were of particular importance to recognizing a digit 3, then they would be multiplied by *large* weights pertaining to features of a 3. Weights $w_3 5$, $w_3 83$ and $w_3 202$ should therefore have high values, contributing to the high probability of classifying a 3. However, the weights of another digit, for example 1, may not share the same set of important

neurons and therefore have lower values for $w_1 5$, $w_1 83$ and $w_1 202$. Naturally, the sum of neurons*weights is over all neurons, and not just a particular set like the example shows. The final probability vector will therefore have a low value for the digit 1, and a high value for the digit 3. This is how the process of classification in CNNs works.

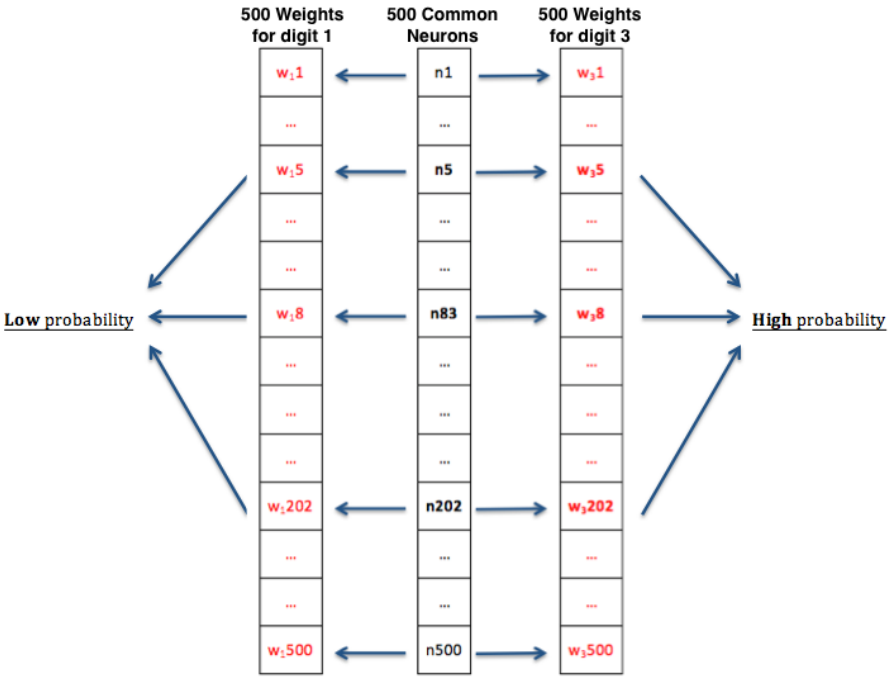


Figure 21. Example of fully connected neurons being multiplied by two arrays of weights for digits 1 and 3

The sorted neurons visualization (figure 22) in this project does the following: after deciding what digit to consider, the neuron activations are **sorted by the importance of their weight**. With an input image of the digit 3, and considering the weights for digit 3, higher neuron activations are expected to come first, while lower, less important activations are expected to come last. The colourmap chosen for this demo is called *viridis*, where the range goes from yellow (highest) to dark blue (lowest). On the right, the row averages of the activations are shown, clearly indicating the decrease of importance from top to bottom.

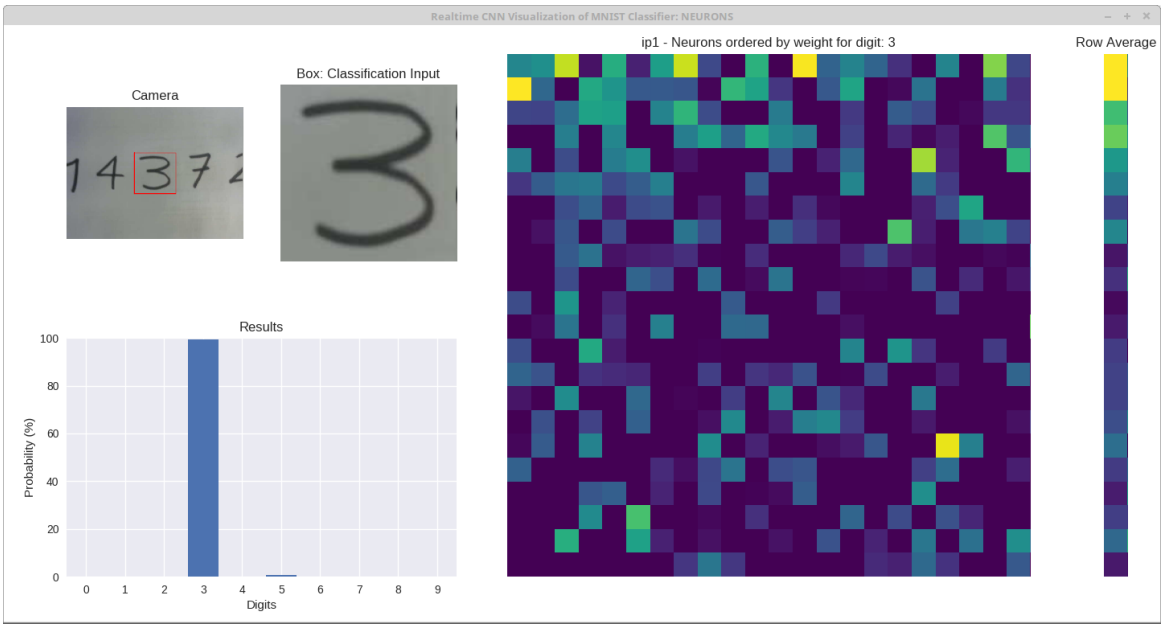


Figure 22. Visualization Tool: Sorted Neurons by Weight View

This arrangement of neurons (sorted by weights) is exactly how the fully connected layer is supposed to behave: for a selected digit, important neurons should be multiplied by high valued weights when the input corresponds to the same digit. No revelations are made here. Nonetheless, this visualization provides interesting insights of its own. For example, users may want to identify which classes have similar or complete opposite features to one another. By setting the tool to sort weights based on the digit 1, one can observe that inserting images of digits 3, 6, 8, 9 shows very low activations in the top level of neurons. This indicates that the high weighted neurons for the digit 1, representing important features, may not be shared with these curved digits. Visualizing the neurons by their sorted weights may provide more fruitful insights when being used on a more complex model, dealing with more abstract features.

4.3 Additional applications

Having completed the visualization tool, further applications were considered. Two use cases were of particular interest: first, using the tool to potentially find patterns by running it on a predefined dataset, and second, running the tool on a face age-detection classifier.

4.3.1 Finding patterns with a predefined dataset

It is expected that a single digit will share the majority of its important features when being fed slightly dissimilar looking inputs of the same digit. This is clearly the case of human handwriting, where no handwritten digit is the same. The network trains to find parameters to classify *any* kind of input that *should* be classified as a specific digit. To accomplish this, test images from the MNIST dataset were used and two videos (gifs) were generated. A few frames from these videos can be found in the appendices section below. The first consists of running the visualization on images of the digit 7 that have resulted in high classification probability. The video clearly depicts a set of shared features in the convolutional layers and a few neurons which are constantly highly activated (neurons sorted by weight for digit 7). Figure 23 displays three out of the forty frames from the video. The second video takes a random set of images (from all digits), which shows the wide range of features and a seemingly chaotic fully connected layer (this time, not ordered for any digit). See figure 24 to get a better idea. Note that since these videos are taking their input from a predefined set of images, the webcam is not needed and therefore the display (vis-a-vis results graph) has been slightly modified for a better visual representation.

4.3.2 Face age-detection classifier

The initial idea for the project was to create the visualization tool for a binary face classification model. Although the priorities of the project took a shift in another direction, at the end the tool was applied to a particular face *age-detection* classifier (Rothe, Timofte, Gool 2015 [13], 2016 [14]). This model receives an image of a face as input, and produces a 101 (0 to 100) long result vector corresponding to age probabilities. It is a highly more complex model than the LeNet for digit classification. This model has over 10 convolutional layers, and is trained with an advanced architecture that is beyond the scope of this project. Figure 25 (in appendices) showcases examples of the tool being run on the face age detection classifier with different input images. The results graph shows the age probabilities, while on the right four convolutional layers are displayed. One can observe how the feature maps become more abstract with every deepening convolutional layer.
This example is clear evidence that the tool created is able to support any general pre-trained Caffe classification model.

5 Conclusion

With CNNs now being applied in almost every industry, understanding the inner workings of these networks has never been so crucial. Being able to run a trained model in realtime, while observing all the different transformations in parallel, offers a multitude of advantages; including an intuitive understanding for new users, as well as providing new insights to experienced researchers.
The aim of this visualization tool was to delve deeper into this disruptive innovation, in an effort to gain a more human-interpretable view of the network's learned features, which was successfully achieved.

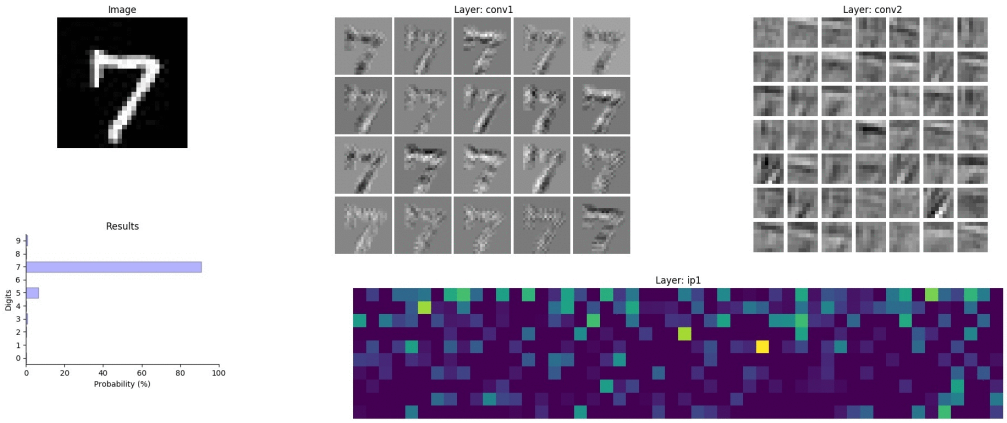
5.1 Future considerations

The visualization tool created during this project can be enhanced in a few ways. First of all, obtaining a better camera and input quality would improve the results of classification. While on the subject of image quality, the feature maps are not as clear as they ideally should be. Therefore, another improvement lies with using even more sophisticated (and computationally expensive) ways to interpolate and represent the data. Secondly, running the tool on GPU rather than CPU should provide a better flow of the visualization, and is critical for larger classification models like the ETH face-age classification. In fact, running the tool on the aforementioned model on just the CPU has proven to be too slow.
The tool provided can be easily customizable by providing different starting parameters as stated in section 3.4.5. For the more experienced programmer, it is also very easy to reconstruct the figure depending on individual needs. For example, for the face-age classification, the fully connected layer was removed in order to make space for the four smaller convolutional layers. The need for this was to visualize the different abstract features at different convolutional levels in parallel, which is exactly what the tool achieves.

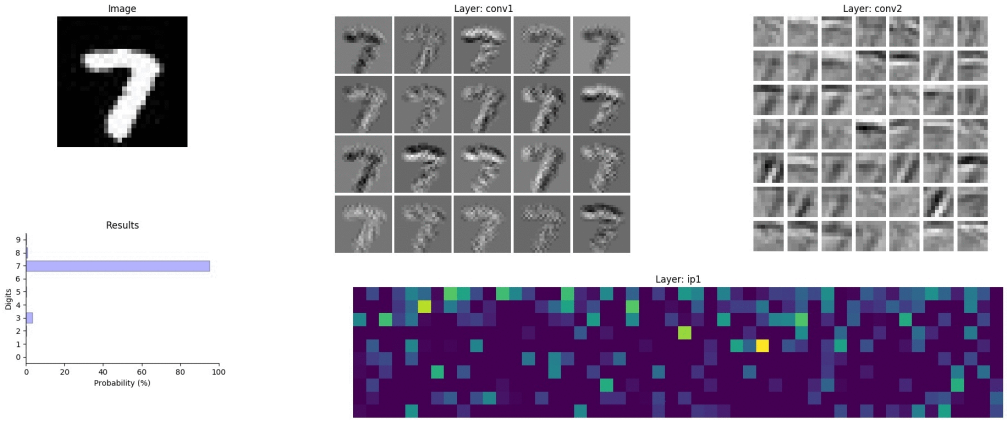
5.2 Acknowledgements

I want to thank prof. Luca Gambardella for introducing me to the fascinating world of deep learning CNNs. With regards to my project, it couldn't have been done without the help and guidance received from my meetings and ongoing correspondence with Alessandro Giusti; thank you for your time and patience. I'd also like to mention the optimal support received from my professors and TAs during these last few years. It's all thanks to your hard work.
However, most importantly, I'd like to thank my parents, sister, and close ones, for enduring my stress, staying optimistic, and reminding me to "live life" once in a while. Couldn't have done it without you.
To all my family, friends and classmates, thank you.

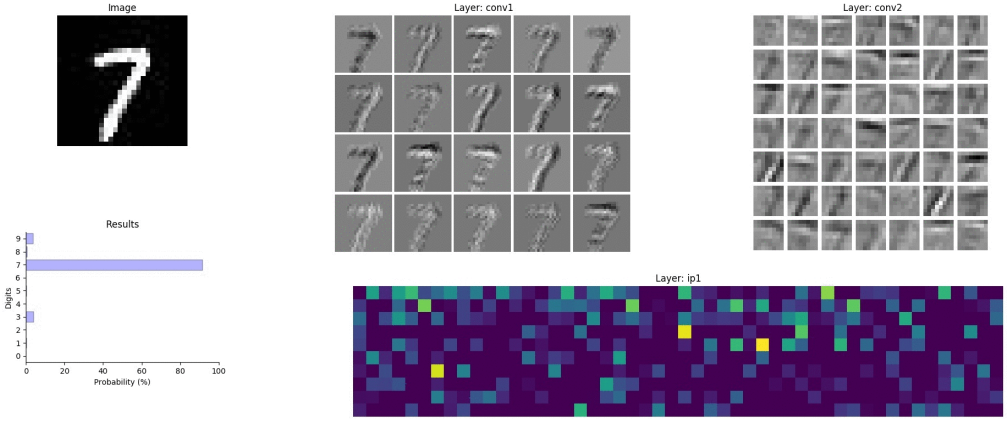
6 Appendices



(a)

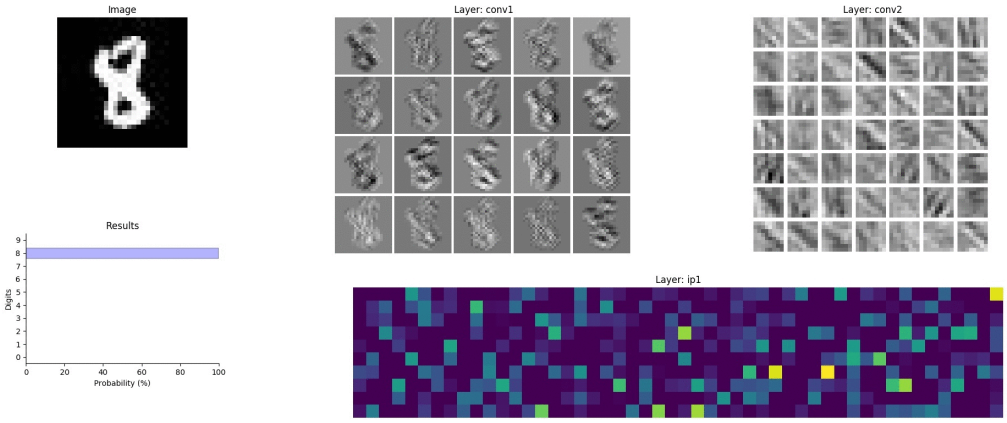


(b)

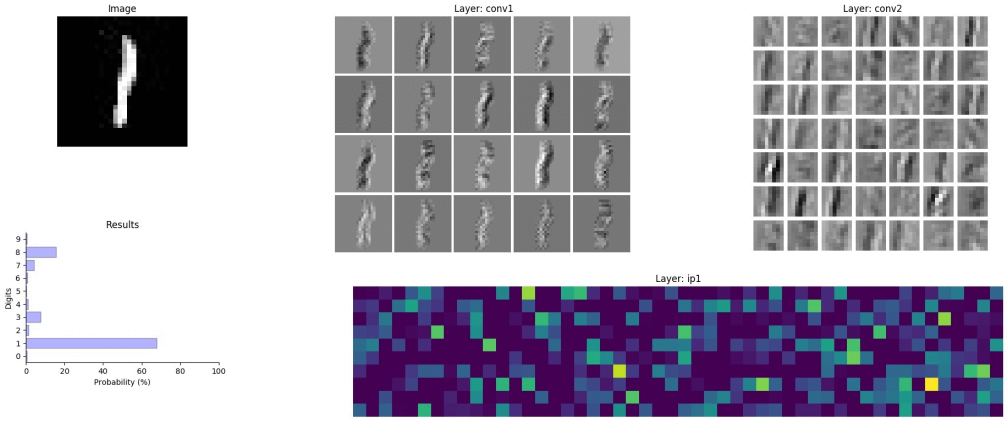


(c)

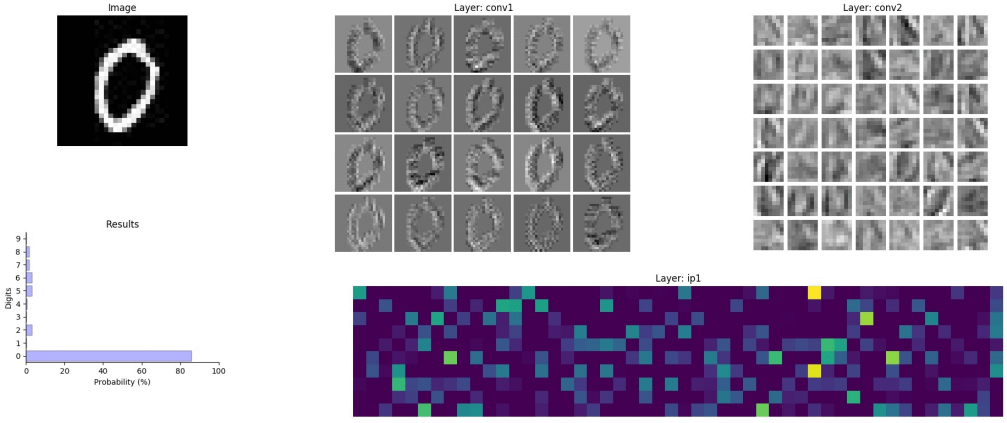
Figure 23. Visusalization on high probability classification of digit 7 with three different input images



(a)

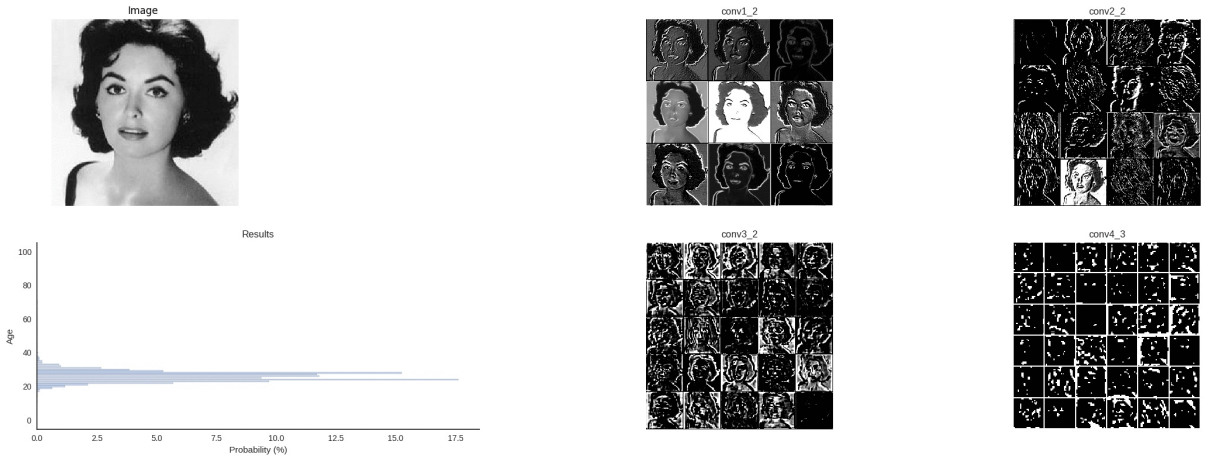


(b)

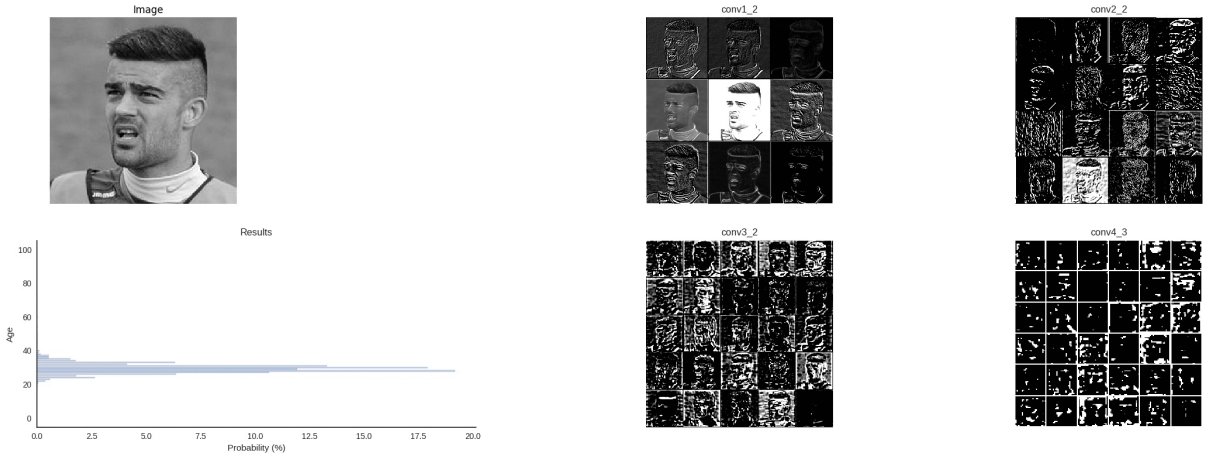


(c)

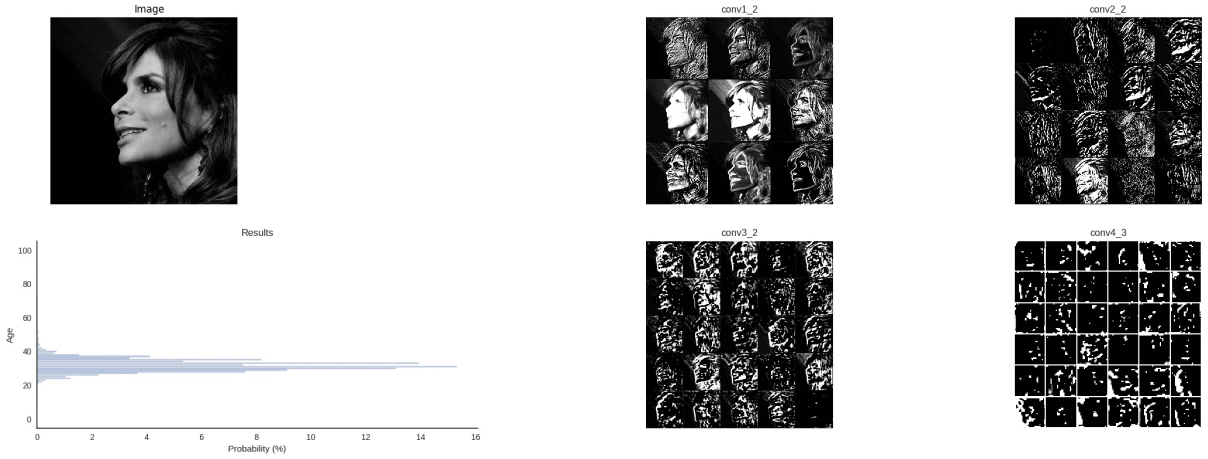
Figure 24. Visusalization on random numbers classification



(a)



(b)



(c)

Figure 25. Examples of the visusalization running on the face-age classification model

References

[1] Capital Ideas Editorial Team. Why artificial intelligence is (finally) coming of age, May 2017. [Online; accessed June 2017]. Retrieved from https://thecapitalideas.com/deep-learning/.

[2] M. Copeland. What's the difference between artificial intelligence, machine learning, and deep learning?, 2016. [Blog post; accessed June 2017]. Retrieved from https://blogs.nvidia.com/blog/2016/07/29/whats-difference-artificial-intelligence-machine-learning-deep-learning-ai/.

[3] A. Deshpande. A beginner's guide to understanding convolutional neural networks (part 1 and part 2), 2016. [Blog post; accessed June 2017]. Retrieved from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/.

[4] IBM. What is big data: Bring big data to the enterprise, 2017. [Online; accessed June 2017]. Retrieved from http://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

[5] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[6] U. Karn. A quick introduction to neural networks, 2016. [Blog post; accessed June 2017]. Retrieved from https://ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/.

[7] A. Karpathy. t-sne visualization of cnn codes. [Online; accessed June 2017]. Retrieved from http://cs.stanford.edu/people/karpathy/cnnembed/.

[8] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[10] G. F. Luger. *Artificial intelligence: structures and strategies for complex problem solving*. Pearson education, 2005.

[11] L. v. d. Maaten and G. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9(Nov):2579–2605, 2008.

[12] C. Olah. Visualizing mnist: An exploration of dimensionality reduction, 2014. [Blog post; accessed June 2017]. Retrieved from http://colah.github.io/posts/2014-10-Visualizing-MNIST/.

[13] R. Rothe, R. Timofte, and L. V. Gool. Dex: Deep expectation of apparent age from a single image. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, December 2015.

[14] R. Rothe, R. Timofte, and L. V. Gool. Deep expectation of real and apparent age from a single image without facial landmarks. *International Journal of Computer Vision (IJCV)*, July 2016.

[15] rsipvision. Deep learning and convolutional neural networks: Rsip vision, 2017. [Blog post; accessed June 2017]. Retrieved from http://www.rsipvision.com/exploring-deep-learning/.

[16] S. J. Russel and P. Norvig. *Artificial Intelligence: A modern approach*. Prentice-Hall, Egnlewood Cliffs, 1995.

[17] F. Shaikh. Deep learning vs. machine learning – the essential differences you need to know!, 2017. [Blog post; accessed June 2017]. Retrieved from https://www.analyticsvidhya.com/blog/2017/04/comparison-between-deep-learning-machine-learning/.

[18] L. van der Maaten. t-sne. [Online; accessed June 2017]. Retrieved from http://lvdmaaten.github.io/tsne/.

Images retrieved from:

- Figure 1: https://thecapitalideas.com/deep-learning/
- Figure 2a: https://s3-ap-south-1.amazonaws.com/av-blog-media/wp-content/uploads/2017/04/08032021/circle-square.jpg
- Figure 2b: https://www.packtpub.com/sites/default/files/Article-Images/B04228_003.png
- Figure 3: http://1.bp.blogspot.com/-aUAXUw-XrwQ/VZ8ZLrix7xI/AAAAAAAAA0M/ENUQU6ETif4/s1600/neuron.png
- Figure 4: http://neuralnetworksanddeeplearning.com/chap6.html
- Figure 5: https://www.vaetas.cz/assets/img/convolutional-neural-network.png
- Figure 6: http://neuralnetworksanddeeplearning.com/chap6.html
- Figure 11: http://cs231n.github.io/assets/cnn/maxpool.jpeg
- Figure 12: https://www.embedded-vision.com/sites/default/files/technical-articles/Algolux_GSAForum/Figure4.png
- Figure 13: http://lvdmaaten.github.io/tsne/
- Figure 14: http://cs.stanford.edu/people/karpathy/cnnembed/cnn_embed_full_1k.jpg
- Figure 18: http://www.kdnuggets.com/wp-content/uploads/tools.jpg
- Figure 19: http://www.pyimagesearch.com/wp-content/uploads/2016/06/lenet_architecture.png