Università Politecnica delle Marche

Facoltà di Ingegneria

Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Magistrale in Ingegneria Informatica e dell'Automazione

Master Thesis

Generative Adversarial Networks per la predizione dello sguardo nel dominio delle GUI

Generative Adversarial Networks for gaze prediction in the GUI domain

Supervisors

Prof. Domenico Ursino

Eng. Gianluca Porcino

Candidate

Alessandro Scopelliti

Academic Year 2018-2019

To my family, to my girlfriend and
to my friends who supported me during these years.

Contents

List of Figures

Abstract

Sommario

Negli ultimi anni, l'area di ricerca che si occupa di studiare l'attenzione visiva ha conosciuto una grandissima diffusione grazie all'Intelligenza Artificiale. L'invenzione di nuovi modelli predittivi rende possibile raggiungere ottimi risultati nel campo delle immagini naturali. Pochi approcci hanno tentato di applicare questi risultati al dominio delle interfacce grafiche web per la previsione dello sguardo degli utenti che osservano un certo layout. La prima parte della tesi si pone come obiettivo quello di predire mappe di salienza, per identificare le zone che attraggono maggiormente il focus dell'utente. La seconda parte, invece, ha lo scopo di prevedere il percorso dello sguardo più probabile che gli occhi dell'utente compiono mentre osservano un sito web per la prima volta. Verrà esplorato l'uso di Generative Adversarial Networks in entrambi i problemi che cercheremo di risolvere.

Abstract

In recent years, the research area dealing with the study of visual attention has become very popular, thanks to Artificial Intelligence. The invention of new predictive models makes it possible to achieve excellent results in the field of natural images. Few approaches have attempted to apply these results to the domain of web Graphical User Interfaces in order to predict the behavior of the eyes of users observing a certain layout. The first part of the thesis aims to predict saliency maps to identify the areas that most attract the user's focus. Instead, the second part aims at predicting the most likely path of the user's gaze while observing a website for the first time. The use of Generative Adversarial Network will be explored in both problems we will try to solve.

Keywords— Saliency Prediction, Gaze Prediction, Web GUI Domain, Generative Adversarial Network, Saliency Map, Visual Scanpath

Introduction

The speed and reliability of processes is crucial within a company. Many enterprises are trying to speed up their internal practices: from the prototyping phase to the production one, many efforts are being made to make companies more agile. Businesses need to keep up with a continuously changing market. The prototyping phase is certainly the one that mostly requires streamlined processes, as it is the one where innovation is produced.

Daimler has several business units dedicated to this important task. Among these, Datalab deals with prototyping innovative solutions to solve stakeholder problems. Every year, the team's designers have to create new designs for various different projects. Each project requires a design validation performed by future users; this way of proceeding requires a direct contact with them. The validation of a design is a very complex task, as it is strictly dependent on the subjectivity of the users' judgement. Many meetings are required for designers to gather a good amount of feedback from clients. The opinions collected are often contradictory, and limited to a few potential users interviewed. The user himself has no idea how his ideal layout should look like, and could propose changes that are not necessary or that could further complicate the design. This is a great waste of time for designers, who have to constantly modify their layouts. The company also wastes money to pay designers to perform tasks that can be avoided. The possibility of having a tool that automatically provides the same feedback that could be provided by users is challenging: clicking a button to get the same result as designers would obtain with several hours of meetings is a great improvement.

Scientific research in this field is very meager, with results that are often limited to proposing guidelines on how designers should carry out meetings with users in order to speed them up. There are few approaches that try to address the problem of predicting the users' gaze in front of a web page. Some examples are found in [1, 2] and predict interesting results. There are more contributions in the domain of natural images, but they need modifications to be adapted to the domain of graphic interfaces.

The growth of Artificial Intelligence has made it possible to propose some very performing models on which we base our research work. The use of Generative Adversarial Networks has obtained very interesting results in the domain of both natural images and web interfaces. More classical techniques, such as the use of filters, have been overwhelmed by the power of AI. The lack of available data has prevented further improvement in results. A very limited amount of datasets are available in the GUI domain, preventing further expansion of this research area. All of them include a small variety of samples and some major biases that make them not suitable.

All the attempts to predict the user's eye behavior are limited to saliency map generation: the areas most likely to be observed in a layout are identified. No approach, has attempted to fully predict the path of users' gaze in this domain yet. All the endeavors with natural images have been made, obtaining encouraging results. The apparent small number of attempts to solve this problem highlights a hole in research that needs to be sewn up. The need for an automatic tool able to predict the salient areas of an image and the path of the users' gaze for designers stimulates us to further investigate this problem.

In this way, we have introduced the two main objectives in this thesis: (*i*) improving the quality of saliency map predictions in the domain of Graphical User Interfaces, (*ii*) predicting the full path of the gaze of users observing a web layout. To accomplish these tasks, we have taken our cue from what is available in the scientific literature.

Before starting to evaluate the models, we had to create a more extensive dataset; in fact, deep learning needs large amounts of data to perform well. Bigger datasets allow better generalization of the models, as well as the training of more complex architectures.

For saliency map prediction we used TSGAN [2], the only available approach applied to web Graphical User Interfaces. Then we introduced changes and improvements to achieve better results. In addition, we compared this model with SalGAN [3], an architecture specifically designed for natural images. We have fine-tuned it and tried to reach even better results in the GUI domain. Finally, we compared the results achieved and defined the strengths and weaknesses of each architecture.

As far as the prediction of the path of the users' gaze is concerned, we chose to start our research from a model that obtained successful results in the domain of natural images. This model, called PathGAN [4], predicts sequences of fixations representing gaze paths. We have made changes and improvements to make the model applicable to our particular case study. We faced many different issues to make the training more stable, in a domain where this sort of task was never attempted before. Then, we evaluated the quality of the results using similarity metrics.

This thesis is organized as follows:

- In Chapter 1, we further highlight the problem we want to solve by introducing the Graphic User Interface design procedure in more detail.
- In Chapter 2, we present a general introduction on neural network architectures and explain which are the pillars of our research.
- In Chapter 3, we introduce in more detail Generative Adversarial Networks, i.e., the neural network architecture that we will often use during our research.
- In Chapter 4, we introduce the concepts of saliency map prediction and visual scanpath.
- In Chapter 5, we report the creation procedure we followed for our new dataset.
- In Chapters 6 and 7, we expose the saliency map prediction architectures and we evaluate the performance of each model.

- In Chapter 8, we discuss the visual scanpath prediction by introducing the model we have used and the obtained results.
- In Chapter 9, we provide an overview of the tool we created, describing its main functionalities.
- In Chapter 10, we draw our conclusions and look at possible improvements that could be introduced in the future.

1

Software User Interface Design

In this chapter, we briefly explore how Software User Interface Design works, explaining the two main steps of the process, namely design verification and design validation. Furthermore, we show why design validation is time consuming and how easily can be optimized using our proposed solution.

1.1 Introduction to Software User Interface Design

Software Design is the process by which an agent creates a specification of a software artifact, conceived to accomplish goals, using a set of primitive components and subject to constraints. User Interface Design is a subset of the more general Software Design activity, in which the main target is to produce graphical layouts for the developed software that satisfy both the requirements and the user usability constraints. These two necessary conditions are called Design Verification and Design Validation, respectively.

1.1.1 Definition of Design Verification

Design verification is defined as: “confirmation by examination and provision of objective evidence that specified requirements have been fulfilled” (21 CFR 820.3). In user interface design the verification step ensures that the output of the design process meets the input requirements, assuring that both the stakeholder and domain critic requests are fulfilled. This step is fundamental for creating software that allows users to access all its functionalities in the fastest and most proper way.

This phase assumes a different relevance depending on the domain of the developed software. Most of the activities spent to verify whether the requirements are satisfied or not are done manually. Indeed, only very sensible domains, such as the medical one, use formal methods. In general this process is time consuming, and since it requires knowledge of different domains, it cannot be automated easily.

1.1.2 Definition of Design Validation

Design validation means: "establishing by objective evidence that device specifications conform with user needs and intended use(s)" (21 CFR 820.3). Such a stage is probably the most complex one for a designer to accomplish, since it is strictly related to users' interactions. There is no *objectively* correct design; therefore most tests cannot be successfully done without the cooperation and contribution of good willing users, ready to provide feedback in order to improve the quality of the designed GUI. The diversification of the user target group increases the difficulty; in fact, tests usually reach a limited amount of users and not easily cover all the potential real customers of the final product. Furthermore, a user brings his/her own personal perspective, which does not necessarily merge with the concept of "perfect" design. Therefore his/her contribution is scoped to the test and is not automatically reliable.

1.2 How can Design Validation be sped up?

Since meeting users' desires can be a very time consuming task, we can think about trying to lighten the process by learning to forecast how users would react in front of a new design. Knowing in advance how a person would move his/her eyes by looking a brand new GUI can help a designer in preventing potential unwanted distraction elements in the design, as well as optimizing it by placing the most relevant parts of the GUI on the expected gaze path. This idea aims, even before testing the designs with users, at avoiding problems with elements' position across the design layout, without the need of an active user. Our goal is that of using AI to build a model able to predict how the user's gaze would move on a web page in advance. This approach would reduce the interaction with users and offers a prediction not based only on a few users but on all the users the model has been trained on.

2

Neural Networks

Neural Networks have become one of the most studied technologies in AI and are currently being applied in several fields. The diffusion of this technique, thanks to parallel computing, introduced new approaches for dealing with complex problems, that could not be easily solved via classic methods. Computer Vision, Speech recognition, Natural Language Processing and Big Data Analytics enormously benefited of the research in this field.

2.1 Introduction to Neural Networks

A neural network can be thought as a black box model, able to approximate linear and nonlinear functions. Its base component is an artificial neuron, i.e., a mathematical entity that mimics real neurons in the human brain. These simple units can be stacked in many different layers in such a way that the neural network can approximate more complex functions. In fact, a single artificial neuron is able to represent only linear functions, limiting the problems that can be solved to a very small subset. Perceptron is the most simple type of an artificial neuron [5]; its mathematical model and its graphical representation can be seen in Figure 2.1.

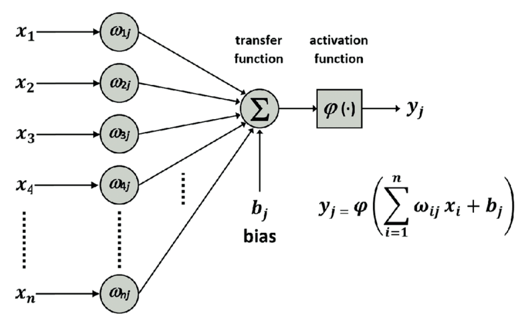


Fig. 2.1: Mathematical model and graphical representation of a perceptron.

A perceptron has several inputs $x_1 ... x_n$; each of them is multiplied by a weight $w_1 ... w_n$. In addition, there is a bias element b_j added to the transfer function. An activation function φ is then fed with the result of this operation and, according to the input value, returns a different output y. A perceptron uses a step function as activation, one ideal for solving binary problems. Recently, different functions have become more popular, such as the sigmoid function, the hyperbolic tangent and the Rectified Linear Unit. Compared to the classic step function, they offer continuity, making them suitable for using gradient based optimization algorithms. A graphical representation of the most popular activation functions can be seen in Figure 2.2. In general, we can state that choosing a particular function for the neural network architecture is strictly dependent on the problem one is trying to solve.

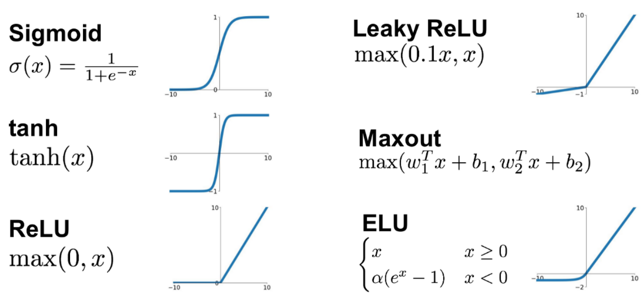


Fig. 2.2: Commonly used activation functions.

As mentioned before, artificial neurons are commonly stacked into layers in such a way that it is possible to solve more complex problems. In addition, also more than one layer can be used to create better performing models in more complex tasks. But how can these mathematical models be useful in practice?

Most of the time data scientists deal with tasks related to forecasting, classification, pattern recognition and many more. The huge advantage compared to classic approaches is that neural networks can learn patterns from some given training data; therefore no mathematical model has to be formulated by the scientist to solve a specific task. It is only necessary to define the input data of the neural network, the architecture and what are the expected outputs of the model. The neural net uses error back-propagation to tune its weights in such a way that the error is minimized. For this reason, at the beginning of this section, we stated that neural networks are function approximators. In fact, they try to minimize the error with respect to the latent function that solves a certain problem. The most basic optimization algorithm used for achieving this task is known with the name of "gradient descent": it minimizes the neural network's error, by updating the weights towards the direction where gradients are stronger, with the aim of finding a global minimum.

2.2 Architecture Overview

Stacking artificial neurons in layers to solve more complex problems is very common. Follows a short overview of the most common neural networks architectures. Each type of layer offers better performance with a certain subset of problems, making Neural Network (hereafter, NN) design not trivial.

2.2.1 Multilayer Perceptron

Multilayer perceptrons (hereafter, MLP) are the most basic NN architecture and one of the first used in research. They are part of the feed-forward neural network family, which means that the connections between nodes do not form cycles. Each perceptron receives the weighted outputs of the nodes of the previous layer in input and passes its outputs to each neuron of the next layer. A graphical representation is reported in Figure 2.3.

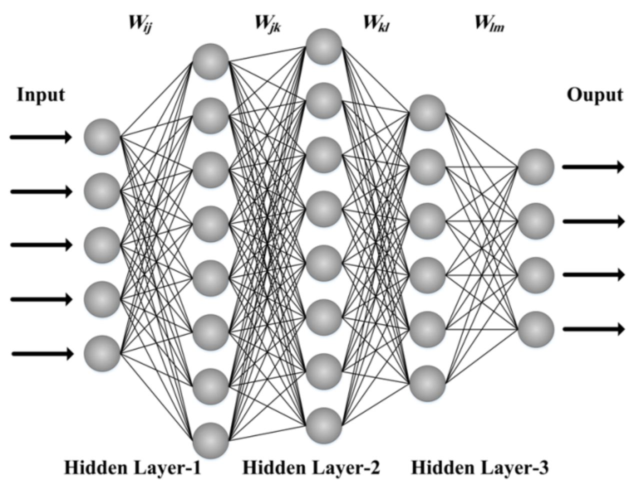


Fig. 2.3: Commonly used activation functions.

This architecture can be easily extended to have multiple layers, each with a different number of nodes; choosing the number of layers and neurons can improve or worsen the model performance. This increases the demand for a more formal architecture design. In general, MLPs are less used compared to other architectures because they have some known issues with the number of parameters to train: being every neuron connected to each one of the next layer, in case of deep networks, the number of weights to tune increases quickly, making the problem intractable even on current computers. Nowadays single layer MLPs are commonly used in bigger networks based on more recent architectures.

2.2.2 Convolutional Neural Networks

Convolutional Neural Networks (hereafter, CNNs), were created to better handle problems related to image processing. This architecture was inspired by the visual cortex of animals, as described in [6]. The approach that made CNNs very common in image-related research was LeNet, a NN created by Yann Lecun et al. [7], aimed to recognize numbers inside 32×32 images (Figure 2.4).

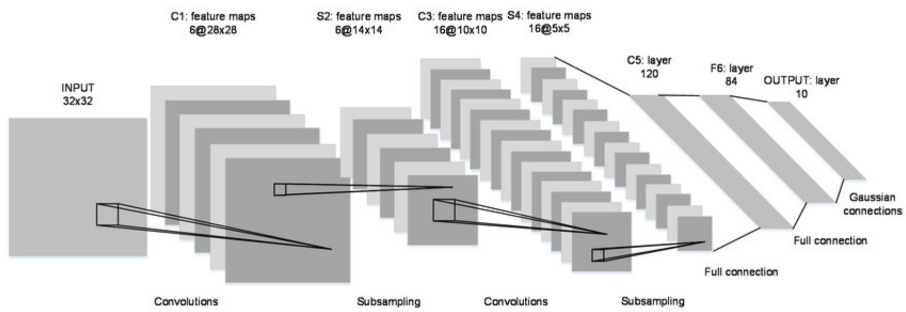


Fig. 2.4: Original LeNet Architecture

Compared to the classical multilayer perceptrons, Convolutional Neural Networks have hidden layers that take into account the spatial position of every pixel in a picture. To accomplish this, every neuron of a convolutional layer is fed with a small subarea of neurons in the previous layer, all spatially near to it. This subarea is typically square shaped and is commonly called "receptive field" of the neuron.

Each output is computed applying a filter to the input neurons followed by an activation nonlinearity, as the ones already mentioned in the NN introduction. Every filter consists of a matrix of weights and a bias vector, with the peculiarity that both are common to the whole layer. The result is a feature map representing the actual features of the image, such as borders and corners.

CNNs offer the possibility to have many different filters in every layer, allowing the extraction of more than one property simultaneously. For dimensionality reduction purposes, sometimes, convolutions are followed by a pooling layer. Many different kinds of layer exist, each with the goal of reducing the output of the convolutional layer. Figure 2.5 provides a visual representation of this structure.

In recent years CNNs are starting to be used also in domains different from image processing, thanks to their capability of extracting spatial features: audio and natural language processing are some examples.

2.2.3 Long Short-Term Memory

Long Short-Term Memory (hereafter, LSTM) is a neural network architecture that belongs to the wider family of Recurrent Neural Networks. It is different from the already exposed architectures because it contains feedback connections between nodes, in opposition to the feed-forward network family. It is ideal when applied

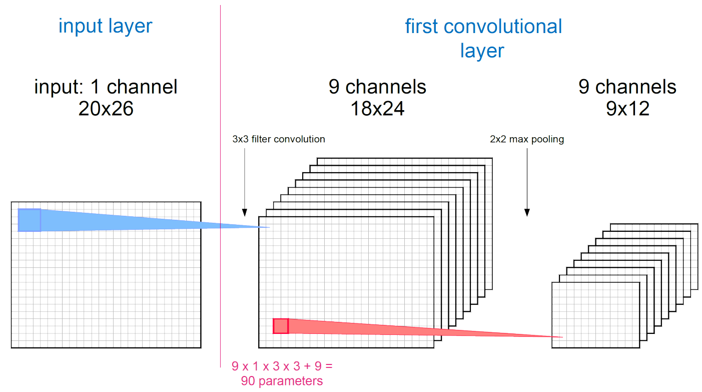


Fig. 2.5: Convolution and pooling layers

to time series data; some examples can be speech recognition, movement analysis, anomaly detection, and many more.

LSTMs were first discovered by Sepp Hochreiter et al. [8] in 1997 and still today they are one of the most used architectures to work with time series. Their structure relies on a combination of gates that emulate the process of storing, deleting and modifying data according to the inputs they receive. Figure 2.6 shw an LSTM unit, along with the detail of its internal gates.

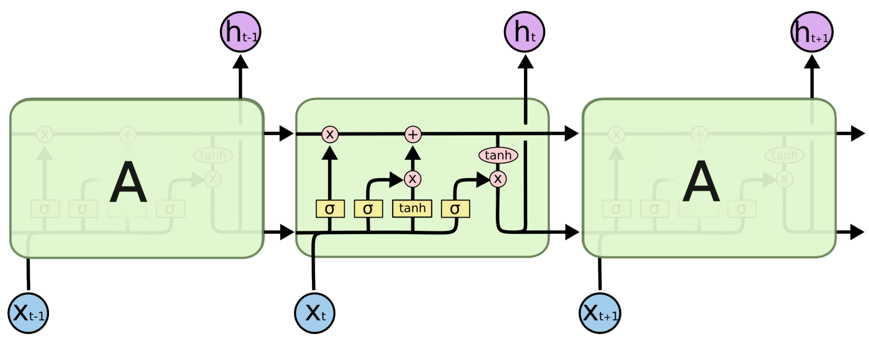


Fig. 2.6: Long Short-Term Memory Unit.

The core of an LSTM unit is its cell state; in the figure, it is represented as the top horizontal line. The multiplicative and additive nodes represent the places where the modifications are done by internal gates. The initial step for modifying the state is to use the first gate from the left to decide which information to forget using a sigmoid layer. Going further, the second phase is crucial to establish what new information should be stored in the cell state: a sigmoid layer is used to decide which values will be updated, whereas a *tanh* layer creates a vector of new candidate values.

The third step consists in updating the cell state with the output of the previous phases; this operation forgets the data result of the first gate and stores the updated information returned by the second gate. Finally the output of the unit itself has to be defined. A sigmoid gate decides what outputs the unit must have, while a *tanh* gate is fed with the cell state; both outputs are multiplied and the result is forwarded to the next unit together with the new cell state. Researchers have developed new versions of LSTMs; however the structure previously exposed is still a solid foundation even in newer versions.

2.2.4 Autoencoder

The last architecture that will be reviewed is a composition of different kinds of layer taken from the NN structures described above. Autoencoders aim at solving the problem of representation learning, thanks to a particular shape of the network that tries to compress the knowledge representation of the original input. It is commonly used in NLP; modifications of the original architecture are also applied in other fields. The basic structure of an autoencoder can be seen in Figure 2.7.

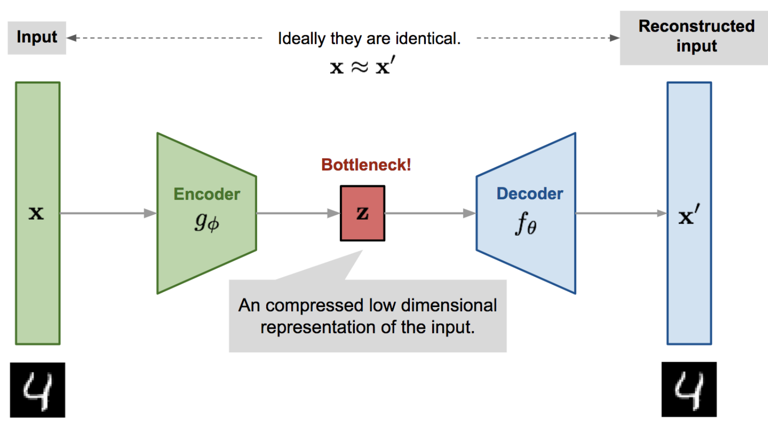


Fig. 2.7: The basic architecture of an autoencoder

The network has an input layer that is connected to an encoder part; it aims at reducing the dimensionality of the input data in order to extract high level features of the input. The encoder output corresponds to a bottleneck, which is then connected to a decoder. Dually to the task performed by the encoder, it tries to reconstruct the original data from the "bottlenecked" features. This architecture tries to extract features free from noise and capable of expressing the high level concept represented by the data itself. Depending on the application domain, different kinds of layer are used to achieve the best results; specifically, image processing exploits CNN layers whereas NLP adopts Multilayer Perceptrons. Later in this paper we will discuss a new evolution of autoencoders used to generate data.

2.3 Use Cases

Neural Networks play a fundamental role in today's technological applications. Things that seem normal for a common user, often have an AI back-end that makes them possible: vocal assistants, face detection and recognition, autonomous driving, smart devices are all widespread implementations of neural networks, aiming at improving the quality of our lives. Also medicine is getting benefits from the AI research, boosting the precision of diagnosis and assisting doctors and medical staff in every day operations. In the following paragraphs, we will provide some examples of groundbreaking AI applications.

2.3.1 Autonomous Driving

Autonomous driving is surely one of the most exciting applications of AI. Nowadays it is becoming more common seeing on roads cars that can juggle traffic by themselves needing as little as the human driver supervising with his eyes the car's maneuvers. First examples of fully automated car parking are starting to appear, and the presence of a human supervisor will no longer be necessary in the near future. Autonomous driving is not only influencing cars, but also many different kinds of vehicles, e.g., drones, airplanes, trucks and many more. Slowly replacing human at driving vehicles can be helpful to avoid potential crashes.

2.3.2 Healthcare

The sectors where AI can provide such a valuable help like in healthcare are few. Many different applications can be found in this domain and the benefits are evident in each of them. A lot of work has been done to help doctors to improve diagnosis of diseases using AI, especially where a human has to manually observe the patient's data. X-rays, ultrasonography and CAT produce images that can be easily submitted to a NN to obtain some valuable results; sometime they detect patterns of diseases that were not seen even by the most expert doctors. Also more advanced tests, not involving images, can take advantage of neural networks because they still have to deal with data. More advanced NN architectures are even able to discover new drugs, helping researchers in this complex task.

2.3.3 Anomaly detection and prevention

Anomaly detection and prevention is a problem common to many different fields. Industry is investing a lot in this, with the goal of managing its machines in the most optimal way. The advent of IoT in general raises the need of something that can process data coming from sensors to understand whether there is a malfunction or not. Deep Learning is commonly used to accomplish this task due to its ability to be very flexible: real-time and offline analyses are possible just by collecting the signals coming from sensors. Data can also be hints of potential failures, enabling the creation of models that can predict when a component needs a replacement. Anomaly detection is also a big issue in cybersecurity. An intruder trying to access a system without authorization is an anomaly by himself: blocking in advance further attempts to breakthrough the system defenses is vital to avoid information theft. Logs and time series data analysis are central also in this case, making once again neural networks ideal for solving this task.

2.4 Known issues concerning Neural Networks

So far we have listed all the applications and benefits that neural networks produce, leaving aside all the main potential flaws that their usage can have. In this section we focus on the main problems affecting NN, preventing them from being used in every field.

2.4.1 Lack of Data

One of the main issues that data scientists have to face when applying NNs in a new field is the potential lack of data. Having a big dataset, proceeding with the training of the model is almost as important as designing the architecture itself. Without data NNs are not able to train correctly and they very often encounter the problem known in literature as "overfitting". This phenomenon prevents models trained on very small dataset to perform well on never seen samples. In this cases, it does not allow the usage of the model in real world situations. The requirement of having data usually prevents the usage of neural networks in environments not yet explored, making still useful the application of classic methods not involving AI.

2.4.2 Computation Power

Even if computation power does not seem a big problem at first, it is fundamental to train networks in acceptable time frames. Training a new model is very often a long task using consumer hardware, that can require days or even weeks. Big companies invest a lot in datacenters so they can speed up the training process and also create bigger models. In recent years, the usage of GPUs for training NNs has contributed in reducing drastically the time of training, allowing the creation of very complex models. Still computation power is a limit that it is slowly overcome thanks to the giant leaps and bounds that technology is making.

2.4.3 Explainability of outputs

As explained in the introduction of this section, neural networks are black box models. These are a particular class of mathematical entities that approximate functions and produce a specific output from a given input, without giving explanation of how the result has been reached. This does not seem a big problem at first, but in reality there are some applications that require the explanation on how the result has been produced. For example, safety critical environment cannot use neural networks, since they are not able to provide how and why the output has been produced in case something goes wrong. Furthermore also some current regulations require algorithms to have an explainable output. This very recent area of research tries to create models that are interpretable by humans: explainable models are starting to become very common in fields such as autonomous driving, medical diagnosis, text analytics, and so on. Some researchers claim that creating transparent models comes with the cost of having poorer performances [9]. Having models that produce explainable outputs is probably one of the most compelling problems for the industry right now, that as of today breaks a further dissemination of Artificial Intelligence.

3

Generative Adversarial Networks

In this chapter, we will analyze in more detail which are the most used generative models, with a particular focus on Generative Adversarial Networks (hereafter, GANs). Furthermore, we will analyze why GANs are very often preferred to other models, without forgetting the known issues of this architecture. Finally, we will list a number of changes to overcome these structural problems, that are commonly used in research.

3.1 Some Generative Models Examples in Research

Before introducing GANs, we will provide a brief overview of the first examples of generative models in literature. Prior to the boom of generative adversarial networks in the last three years, there were already some models accomplishing the same tasks with comparable results. We will analyze the most popular ones, namely Boltzmann Machines and Variational Autoencoders.

3.1.1 Restricted Boltzmann Machines

Boltzmann machines were invented in 1984 by Hinton et al. [10]. They belong to the family of stochastic neural networks, due to the stochastic transfer function of each neuron. These type of networks are one of the first examples of generative models in the scientific literature, but they have not received enough attention yet because of their training complexity. A simplified version, known as restricted Boltzmann Machines (hereafter, RBMs), was later published by Hinton himself [11], making their training possible using more efficient algorithms.

In Figure 3.1, we can notice the main difference between the two architectures: RBMs have no connections linking the neurons to each layer, reducing the overall number of weights to train. The peculiarity of RBMs (and Boltzmann Machines in general) is that no output neurons are present and the whole network is divided in two layers, the visible and the hidden ones. Each layer has a bias term useful during the training process. Training is split into a first phase of forward passing of the inputs followed by a subsequent phase of reconstruction of the original data. This last

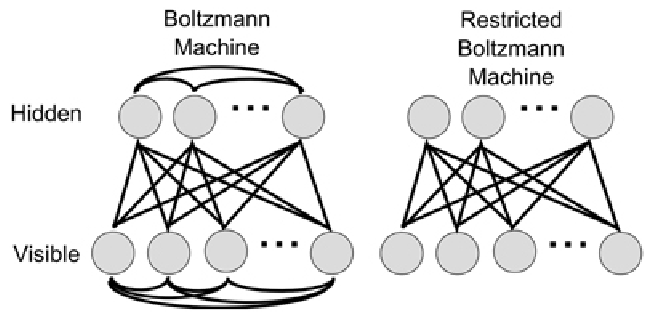
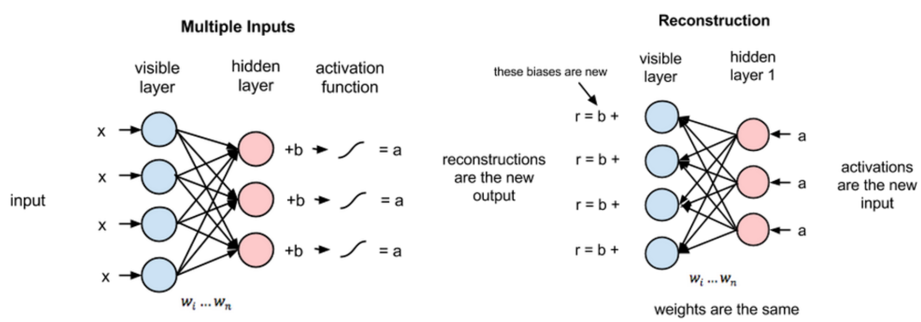


Fig. 3.1: Boltzmann Machine and Restricted Boltzmann Machine

step is necessary to calculate the error and consequently modify the net weights to reduce it as much as possible. The final goal of training is to approximate the input data distribution in order to generate data similar to the inputs. A visualization of the two training steps is reported in Figure 3.2.



(a) (b)

Fig. 3.2: Training steps of a Restricted Boltzmann Machines: (a) forward pass, and (b) reconstruction

3.1.2 Variational Autoencoders

Invented by Kingma et al. [12], variational autoencoders (hereafter, VAEs), together with GANs, represent the most widely used generative model architecture. Compared to classic autoencoders, VAEs organize the latent space of features in order to guarantee some useful properties. For data generation, the features' latent space needs to be regularized, respecting continuity and completeness constraints. Continuity means that two near points in this space must have a similar content once decoded; completeness, instead, suggests that a point of the same latent space must have a "meaningful" content after the decoding process. To enforce these constraints, instead of encoding an input as a single point, we encode it as a distribution over the latent space. In comparison to a classic autoencoder, the loss is made up of two main terms, namely: (i) a reconstruction term, which makes the encoding-decoding scheme as performing as possible, (ii) a regularization term, which tends to regularize the organization of the latent space by making the distributions returned by the encoder close to a standard normal distribution. A graphic representation is available in Figure 3.3.

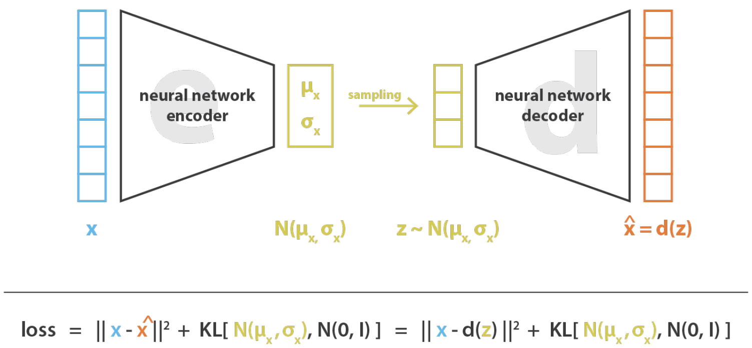


Fig. 3.3: Variational autoencoder architecture including its loss formulation

From a performance viewpoint, we can generally say that the outputs are slightly noisier than those that can be achieved with a generative adversarial network in some applications. However, training is stable, offering a valid alternative to GANs.

3.2 Introduction to Generative Adversarial Networks

Generative Adversarial Networks are a type of neural network architecture discovered in recent years. As reported in the first GAN paper, written by Goodfellow et al. [13], we can think of GANs as a band of banknote counterfeiters trying to trick the police. The first part of the network, called generator, embodies the counterfeiters while the last part, called discriminator, represent the police trying to discover the fake banknotes. The generator simply tries to create fake data that is as close as possible to the distribution of the real data in the training dataset, while the discriminator attempts to distinguish whether the input data is coming from the real or fake data distribution.

A basic representation of the network can be seen in Figure 3.4. In the following subsections we will analyze each component of the network in more detail.

3.2.1 Generator

As already seen, the generator is intended to generate fake data as similar as possible to the ground truth. The generator's structure is not fixed, but it varies according

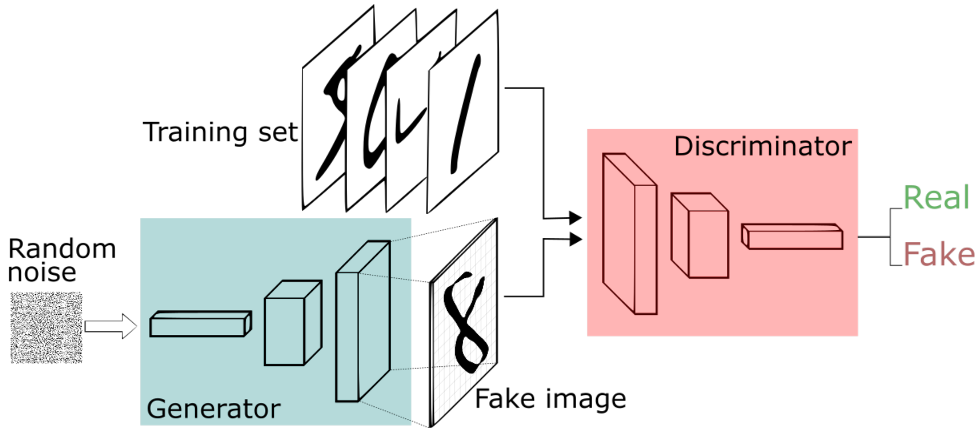


Fig. 3.4: Generative adversarial network architecture example

to the type of data it should try to generate. For example, in the image domain, an architecture similar to that of a decoder is often used, while, for the generation of sequences, the LSTM layers are preferred. Customized models can also be used as long as the condition to generate new data is met. The simplest version of generator takes a noise vector as input, typically Gaussian, and uses it to generate new data. Noise is intended to act as a seed for the process of generating new data, thus avoiding always getting the same output. This noise vector is also often called Latent Sample. A Latent Sample, or a Latent Variable in general, is one of those variables that are important for a domain albeit not directly observable. These variables can be thought as a compressed form of input data that preserves its high-level features, just like the output of an encoder inside autoencoders. During training, the generator learns to associate the latent space noise vectors with the actual distribution of the problem. In Figure 3.4 the generator is highlighted with a light blue color.

3.2.2 Discriminator

The output of the generator is fed, together with some elements of the original dataset, to the discriminator with the purpose of identifying which are the real or fake samples. This part of the network is fundamental as it allows us to understand the quality of the data produced by the generator. During the training phase it provides useful information on how to improve the performance of the generator. In its most used form, the discriminator is a binary classifier that labels the input data as true if it believes that this data comes from the training set or as false if it believes that this data is generated. Also in this case, a general architecture valid for all problems is not defined, making a careful planning necessary. More recent versions of discriminators [14] return probability maps taking into account the spatial dimension of the output in the generation of images. For example, it can be useful to understand in more detail which parts of an image is not realistic, allowing better results during the training phase. In Figure 3.4, the discriminator is highlighted with a red color.

3.2.3 Training

Although a GAN is composed of two separate neural networks, with different tasks, the training process requires to train generator and discriminator together. It is an unsupervised learning problem, even if the procedure resembles in every way a supervised one: the input data are all labelled as real or fake making it apparently similar to the training of a binary classifier. Recalling the metaphor mentioned above, generator and discriminator are "adversary" and try to improve during training, based on how the corresponding opponent is behaving. It is a zero-sum game where opponents try to reach the Nash equilibrium. The standard workflow requires the generator to output fake data. These artificial samples, together with a set of real data samples of the same size, are fed to the discriminator, which must recognize the real ones from the fake ones. The discriminator's weights are then updated according to how it behaved. Right after this step, the generator is also updated, based on how well it managed in deceiving the discriminator. Like any other neural network, it is necessary to define a loss function to be minimized to allow the correct training; it must take into account the workflow above mentioned. The objective function can be expressed as the following:

$$\underset{G}{min}\,\underset{D}{max}\,V(G,D) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_{data}(z)}[log(1-D(G(z)))] \quad [3.1]$$

Here, G represents the generator, D denotes the discriminator, x is a sample coming from the real data distribution and z indicates a sample from the noise data distribution. The equation consists of two different terms; the former refers to the real data while the latter to the generated one. We can split the equation into two parts for more clarity.

$$\underset{D}{max}\,V(D) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z\sim p_{data}(z)}[log(1-D(G(z)))] \quad [3.2]$$

$$\underset{G}{min}\,V(G) = \mathbb{E}_{z\sim p_{data}(z)}[log(1-D(G(z)))] \quad [3.3]$$

Equation [3.2] represents the function to be maximized by the discriminator. The first term is maximized when the real data is recognized correctly as real; instead, the second term is maximized when generated data is recognized as fake. Equation [3.3], instead, focuses on maximizing the error done by the discriminator on the generated data by minimizing the error done by the generator while outputting data. The main purpose of this procedure is to ensure that the distribution of the generated data is as close as possible to that of the real data. The training cycle continues ideally until the generator creates perfect replicas of the input domain in such a way that the discriminator is no longer able to distinguish between true and false data; in this last case the accuracy would be around 50% (like flipping a coin).

3.2.4 Known Issues

The classic architecture of generative adversarial networks presents some problems and limitations that have been solved during the last years of research. The biggest issue with this type of neural network is undoubtedly the difficulty of converging the model to an acceptable solution. There may be many causes that produce this phenomenon, so we limit ourselves to reporting the most frequent ones. It very often happens that one between the generator and the discriminator learns too well to carry out its task, preventing the other one from being able to learn. This event is also known as Diminished Gradient, and most of the time affects directly the generator. Since it learns from the mistakes made by the discriminator, when this issue occurs there is no learning: the gradients vanish quickly, thus canceling the possibility of obtaining an acceptable solution. Another very common phenomenon leading to a lack of convergence is known as mode collapse. This leads the generator to generate a very limited variety of samples, thus preventing the generation of realistic data. In Figure 8.2 we can see an example of mode collapse: the generator after many training iterations, will create very similar outputs that match only with some modes of the original dataset.

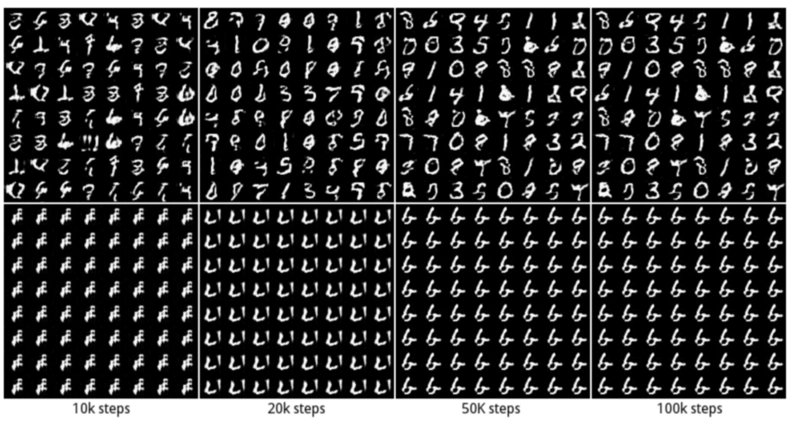


Fig. 3.5: Mode collapse example: the top row contains the training without mode collapse, while the bottom one shows mode collapse across different training steps

Although these problems are common, they are not always the only ones to occur. The non-convergence of the model may also depend on the fact that the generator and the discriminator are not well balanced, requiring each to have a different frequency of weight updates. In general, there are many hyper parameters that need to be tuned, and this introduces even more complexity to the training process. Finally, a last factor to consider is that a GAN has a loss that does not express objectively if the generation of a certain data is taking place in an optimal way or not, so the training procedure must be stopped manually based on qualitative parameters. In the following subsection, however, we will analyze how some of these problems have been overcome in research.

3.3 Techniques for solving known issues

In this section, we will present some proposed solutions to solve the problems affecting GANs. There is not a way to ensure the successful training of this type of networks yet, but there are some techniques helping to achieve good results.

3.3.1 Wasserstein GAN

One of the best techniques to facilitate convergence during training is to modify the cost function. Since we work with probability distributions it is necessary to have a reliable way to measure the distance between the model distribution and the real one. Not all distance or divergence measures have the same effects on the convergence of GAN training, which is why research is very active in this area. There are many types of loss functions for GANs. However, the one returning better results is called Wasserstein GAN. It was first formulated by Arjovsky et al. [15] taking its inspiration from the Earth-Mover distance (hereafter, EM distance). Many examples are given in [15] where it is shown that classic distance or divergence measurements do not always provide a usable gradient to minimize the cost of the model. By contrast, the Earth-Mover distance, also called as Wasserstein-1 is able to satisfy just this feature, ensuring a continuous gradient that can be used with optimization algorithms based on the latter. The Earth-Mover distance is defined as:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \underset{\gamma \in \Pi(\mathbb{P}_r, \mathbb{P}_g)}{inf} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||] \qquad [3.4]$$

where $\Pi(\mathbb{P}_r, \mathbb{P}_g)$ is the set of all joint distributions $\gamma(x, y)$, whose marginals are $\mathbb{P}_r$ and $\mathbb{P}_g$, respectively. The formula can be interpreted as the amount of "mass" that must be transported from x to y in order to transform the distribution $\mathbb{P}_r$ into $\mathbb{P}_g$. The EM distance is the transport plan with the minimum "cost". In real application scenarios the EM version introduced above is intractable. However, as reported in the paper, we can rewrite the Wasserstein distance through the Kantorovich-Rubinstein duality as:

$$W(\mathbb{P}_r, \mathbb{P}_g) = \underset{||f||_L \leq 1}{sup} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_g}[f(x)] \qquad [3.5]$$

Here, the supremum is over all the 1-Lipschitz functions $f : X \rightarrow \mathbb{R}$. As stated in this research, in reality, we need the GAN's weights to be included in a compact space and, then, proceed with back-propagation. To enforce this constraint, weight clipping is used, even if it is not the best technique. In spite of this, as we can see in Figure 3.6, the gradient maintains a linear trend compared to a classic GAN with the vanishing gradients problem. In practice, the loss function, compared to that already seen for classic GANs, has a slightly different structure. If we observe the Formula [3.6] and [3.7], we can notice how the function $f_w(x)$ represents the raw output of the neural network. In normal GANs, the discriminator does a binary classification using a final sigmoid layer; on the other hand, WGANs use the Wasserstein distance with no need to employ the final sigmoid layer. Generator's and discriminator's losses are the following:

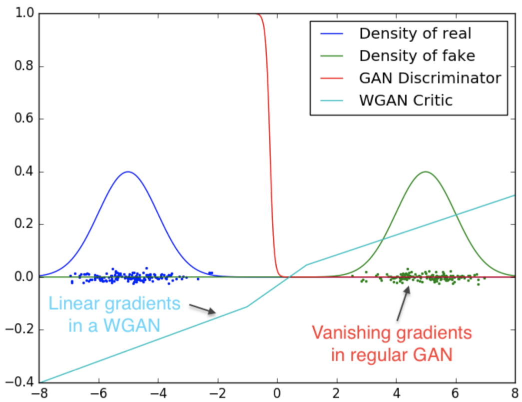


Fig. 3.6: Normal GAN and WGAN gradient evolution over training.

$$L_{gen} = -\mathbb{E}_{x\sim\mathbb{P}_g}[f_w(x)] \quad [3.6]$$

$$L_{discr} = -\mathbb{E}_{x\sim\mathbb{P}_r}[f_w(x)] + \mathbb{E}_{x\sim\mathbb{P}_g}[f_w(x)] \quad [3.7]$$

In WGANs, the discriminator is called "critic" because it no longer performs a classification task between true and false samples, but it computes only a distance. Compared to the classic GAN loss performance is better, both in the quality of generated samples and in the stability of training, but more improvements can still be done.

3.3.2 Wasserstein GAN with Gradient Penalty

Since weight clipping is a drastic method to enforce the 1-Lipschitz condition, research has identified a better technique to achieve the same result. In addition, there are still many problems which Wasserstein GANs still suffer from. Indeed, gradients that explode or vanish are common if there is no careful calibration of the clipping parameter. Furthermore, there is a tendency for the network to learn simpler distributions. All these problems are taken up by I. Gulrajani et al. [16], who introduced an alternative technique to weight clipping known as Gradient Penalty. Instead of modifying the network weights, the gradient is penalized so that its norm does not deviate too much from 1. This is achieved by correcting the loss of the critic, as reported in the following formula:

$$L_{discr} = -\mathbb{E}_{\tilde{x}\sim\mathbb{P}_r}[f_w(\tilde{x})] + \mathbb{E}_{x\sim\mathbb{P}_g}[f_w(x)] + \lambda\mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\left\|\nabla_{\hat{x}} f_w(\hat{x})\right\|_2 - 1)^2] \quad [3.8]$$

The last addend represents the gradient penalty: the greater the difference of the gradient's norm from 1, the greater the penalty inflicted to the loss of the critic. The authors of [16] report better results, both in training speed and sample quality, compared to classical WGANs.

3.3.3 Conditional GAN

The classic architecture of Generative Adversarial Networks is able to generate samples only from noise input. But what if we have additional data that could help the network at disposal? In [17], there is a change compared to normal GANs that foresees to feed additional data to both the generator and the discriminator.

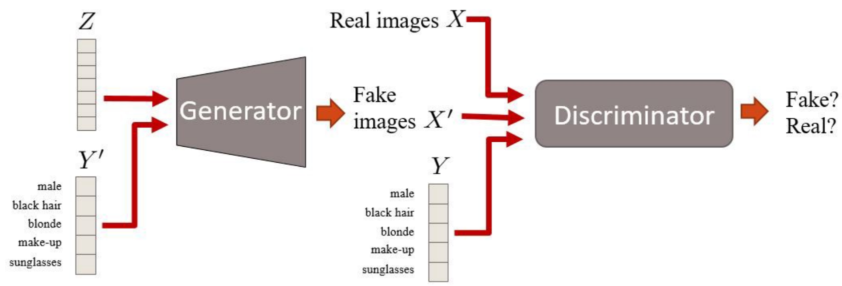


Fig. 3.7: Conditional GAN architecture

As represented in Figure 3.7, the architecture is very similar to the original GAN with only the addition of input data conditioning both generator and discriminator. This modification allows better results, as well as better image generation, when the domain is very specific. These architecture is known in literature as Conditional Generative Adversarial Network (hereafter, cGAN).

4

Saliency and Path Prediction on Web pages

Since our aim is to analyze graphical user interfaces, in this chapter, we present an overview of the techniques used to evaluate the visual impact they have on users. Users' gazes are analyzed in two different ways, namely: (i) through heatmaps generated from the points that are fixed by the gaze or, (ii) through the complete path that the user's eye takes in front of the interface. The former technique considers heatmaps as saliency maps, which represent the areas to be most likely fixated of the web page. The latter, instead, analyzes the paths of users' glances, also called visual scanpaths. In the following sections, we will analyze them in more detail. We also consider the datasets publicly available to train and test these models and the evaluation metrics most frequently used in research.

4.1 Saliency Maps

Before addressing the state of the art of saliency map generation, we give a formal definition of this concept. In [18], saliency is defined as something that

> "intuitively characterizes some parts of a scene - which could be objects or regions - that appear to an observer to stand out relative to their neighboring parts".

Most of the studies on saliency maps have been performed on natural images in the past. Itti et al. [19] have introduced a methodology based on orientation, intensity, color information and other low-level features that are combined to produce saliency maps. In [20], the authors propose the generation of saliency maps through a graph-based methodology using Markov chains: first activation maps are created for each feature channel, then they are normalized to highlight important areas.

More recent approaches try to use deep learning in order to exploit neural networks to identify the most salient areas of images. The first appearance of saliency maps associated with deep learning date back to [21]. Since then, it has proved that they are the preferred technique to be exploited in this particular problem. The first remarkable results are obtained in [22], using a recurring convolutional neural network, able to extract features and to take the spatial long-short memory into account. In [3], the authors describe an innovative approach that uses generative adversarial networks in order to predict saliency maps as real as possible.

State of the art results in MIT saliency benchmark are accomplished by [23] and [24]; the former obtains the best qualitative results. Unlike real images, the layout of web pages has been taken into analysis by these predictive techniques very rarely. Available datasets are also very limited. The first paper that tries to generate saliency maps for website layouts is [1]. It proposes the usage of multiple kernel learning to integrate various feature maps. In [25], text features and discriminative regions of a web page are used to generate saliency maps. Finally, the approach described in [2], uses a two-stage adversarial network, which exploits an autoencoder to generate saliency maps. The approach's strength relies on the usage of a two-stage generator, which creates a coarse saliency map in the first stage and refines it in the second one.

4.2 Visual Scanpaths

While image prediction based on saliency maps collected a great interest from the scientific community, the same cannot be said for visual scanpaths. Indeed, in the literature, the attempts that try to accomplish this task are few, and all are limited to real images. For this reason we will take the cue from what has already been done in the natural image domain and apply it to websites. In [26], an approach based on regions of interest and inhibition of return is described. Inhibition of return avoids predicting fixations that have already been observed in the path; detecting regions of interest, instead, ensures that fixations are only in salient regions of the image. LSTM layers are used to consider the time dimension of the problem. [4] uses a conditional generative adversarial network to predict gaze paths. The generator receives the website's layout image as input and generates the corresponding path. The original image and the generated data are, then, fed to the discriminator, who evaluates its quality. Both approaches use interesting concepts that we will try to use in our research.

4.3 Datasets

In the Artificial Intelligence field, the availability of datasets is fundamental to train the model. All the models already mentioned use neural networks, and, then, need image datasets with associated fixation points which users have looked at. Only few datasets meet this feature, and those that do it are very often small. Collecting glance data is a complex and time-consuming task, as it requires a suitable tool to track the pupils of the users.

Most of the time eye trackers are used for data collection; they are often very accurate, but they are not very flexible because their availability is rare. To face this problem, solutions using webcams have been proposed. These are much more common but, at the same time, less precise.

A large-scale data collection methodology that uses webcams has been proposed in [27]. If we focus on the web page domain, there is only one dataset that collects both the layouts and the fixations of the gaze of a set of users.

This dataset, called Fiwi [1], contains images of 149 websites, each with the fixation data of 11 people. In this case, an eyetracker was used to collect better quality data. The images are equally shaped (1366x768 pixels) and distributed in 3 different classes, namely: pictorial, text and mixed.

All the fixation data comes from users in the 21-25 years range, 4 males and 7 females. Users had to free-view every layout for five seconds. Lastly, data was gathered in a controlled environment. Specifically, they used a dark room and the users had their head 60cm far from the screen. This very limited amount of data can have some limitations to train more complex models; that's why we will address this problem later on.

4.4 Evaluation Metrics

After introducing saliency maps and visual scanpaths, we analyze which metrics are able to evaluate them. In [28], the authors summarize which are the most used metrics to evaluate the quality of outputs in literature. Let's start introducing the ones concerning the quality of saliency maps. The paper introduces the Pearson correlation coefficient (hereafter, CC) as the first metric. Given two saliency maps H and P the coefficient is defined as:

$$CC_{H,P} = \frac{cov(H,P)}{\sigma_H \sigma_P} \tag{4.1}$$

where $cov(H,P)$ is the covariance between H and P, and σ_H and σ_P are the standard deviation of H and P, respectively. This coefficient has a value between -1 and 1: zero indicates no correlation between the two maps, 1 indicates perfect correlation, while -1 still indicates perfect correlation but with the data varying in the opposite direction. In our case study, we will try to get values as close to 1 as possible.

The second metric we introduce aims at measuring the dissimilarity between two probability distributions. In order to work with this metric on saliency maps it is sufficient to transform them into two-dimensional probability density functions. For this purpose, we divide each pixel of the map by the sum of all the pixel values [28]. The metric into examination is called Kullback-Leibler divergence (hereafter, KL-divergence) and is expressed as:

$$KL(H,P) = \sum_k p_k log \frac{h_k}{p_k} \tag{4.2}$$

where h_k and p_k are the probability distribution functions associated with the saliency maps H and P. This metric can range from 0 to infinity, and does not respect triangular inequality. The KL-divergence assumes a value of 0 if two probability distribution functions are equal.

The third metric, called Receiver Operating Characteristic (hereafter, ROC), is the most used one. It consists of comparing two saliency maps by setting a threshold on both. One of the two saliency maps represents the ground truth while the other one denotes the prediction. Every pixel of the maps above the threshold is considered as a fixation. We obtain a binary saliency map where white areas are considered as fixated, whereas black areas are not considered fixated. Four numbers representing the quality of the classification are calculated: true positives, false positives, false negatives and true negatives. The true positives represent the number of pixels fixed in the ground truth that are labelled as such in the prediction. False positives are all those pixels that are marked as fixed in the prediction that are not fixed in the ground truth. True and false negatives follow the same idea just expressed for the positive case. A ROC curve representing the rate of false positives as a function of the rate of true positives is generally used to evaluate performance and calculate the metric. The area under the ROC curve is computed to obtain the metric's value.

This metric has several variants specifically designed to be applied to the saliency map domain. AUC-Judd and AUC-Borji, described in [29], modify the way the threshold is calculated with respect to the classic ROC curve. They compare the predicted saliency map with the ground truth fixation map. A more meaningful value than the classic version is obtained, since only the specific areas that were actually observed are taken into account. Both metrics compute the true positive rate as the proportion of saliency map values above threshold at fixation locations.

In the former metric, a series of n thresholds are sampled from the values that the saliency map assumes in the pixels corresponding to the fixations in the fixation map. The latter, instead, uses random uniform samples of image pixels as negatives and defines the saliency map values above a certain threshold as false positives. Threshold values are sampled at a fixed step size. Both metrics range from 0 to 1: our goal is getting as close as possible to 1.

The fourth and last metric we introduce for the analysis of saliency maps is called Normalized Scanpath Saliency (hereafter, NSS). It is measured as the average value of the saliency maps in the fixation locations. The first step for computing it is to standardize the saliency map in order to have a zero mean and unit standard deviation ([4.3]).

$$Z_{SM}(x) = \frac{SM(x) - \mu}{\sigma} \qquad [4.3]$$

The NSS value for a given fixation is computed on a small neighborhood centered on that location, to avoid focusing on a particular point ([4.4]):

$$NSS(x_{f(k)}) = \sum_{j \in \pi} K_h(x_{f(k)} - x_j) Z_{SM}(x_j) \qquad [4.4]$$

$$NSS = \frac{1}{M} \sum_{k=1}^{M} NSS(x_{f(k)}) \qquad [4.5]$$

K is a kernel with a bandwidth h and π is a neighborhood. The NSS value is the average of $NSS(x_{f(k)})$ for all fixations M of an observer. The higher the value the better the quality of our results.

Let's now analyze the main metrics that are used to evaluate different visual scanpaths. Indeed, there are several evaluation metrics, but there is one of them that contains almost all the information of our interest. This metric was first published by Jarodzka et al. in [30] and, then, it became one of the most informative metric in this domain. Each scanpath is treated as a sequence of vectors each representing saccades. A saccade is a movement of the eye between two fixation points. The set of vectors of a scanpath is aligned with a certain characteristic. Each vector corresponds to one or more vectors of another scanpath. For each characteristic for which the scanpath is aligned, a similarity metric is calculated. The procedure reported in the paper obtains 5 measurements of similarity: difference in shape, amplitude and direction between saccade vectors, distance between fixations and fixation duration.

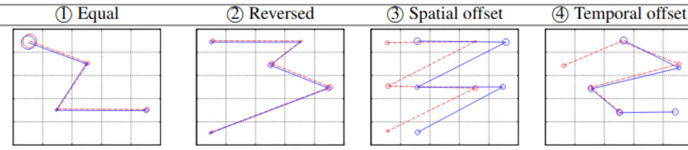
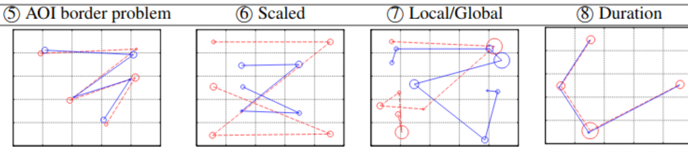
Metric	① Equal	② Reversed	③ Spatial offset	④ Temporal offset
				
$r(\mathrm{AM}_1, \mathrm{AM}_2)$	1.00	1.00	-0.23	0.62
Levenshtein distance	0.00	0.80	1.00	0.40
Proposed:				
Shape	0.00	0.31	0.01	0.11
Position	0.00	0.34	0.15	0.16
Length	0.00	0.16	0.00	0.01
Direction	0.00	0.26	0.00	0.12
Duration	0.00	0.43	0.32	0.32
Metric	⑤ AOI border problem	⑥ Scaled	⑦ Local/Global	⑧ Duration
				
$r(\mathrm{AM}_1, \mathrm{AM}_2)$	0.90	-0.36	-0.05	1.00
Levenshtein distance	1.00	1.00	0.76	0
Proposed:				
Shape	0.05	0.32	0.29	0.00
Position	0.04	0.28	0.39	0.00
Length	0.04	0.32	0.23	0.00
Direction	0.01	0.01	0.33	0.00
Duration	0.57	0.32	0.33	0.88

Fig. 4.1: Representation of how the similarity metrics change given two scanpaths to compare. The similarity metrics are also compared to the correlation coefficient (r) and the Levenshtein distance

Figure 4.1 represents how every similarity metric is influenced when comparing two different scanpaths. In the model evaluation section we will use these metrics to assess the quality of our results.

5

Dataset

In this chapter, we analyze the main problems of the only available dataset in literature. Then, we discuss the main reasons why it was necessary to create a new one, and describe our data gathering procedure. Finally, we provide an overview of the main characteristics of the newly created dataset.

5.1 Why a new dataset?

As already reported in the previous chapters, scientific literature provides only one example of dataset, called Fiwi [1], containing both web page layout images and gaze data. All of the research regarding saliency map and gaze path prediction in the GUI domain has to use it to train its models due to its uniqueness.

We can identify many improvements that can make this dataset even better than its current form. In fact, it has been created collecting data from only 11 volunteers; each of them observed 149 websites each. The limited number of users involved in data collection is not able to completely enclose the way of observing images of the human being. Moreover, the users' gender is not balanced (7 women and 4 men): this fact can introduce a gender bias. If we also consider the fact that each user has to observe numerous websites, each for five seconds, we can think that this data collection procedure generates a considerable amount of stress on the user.

Furthermore, data is collected in an unnatural way that prevents the creation of a realistic model. In fact, users are placed with their chin on a head rest, in a dark room, 60 cm away from the screen. The set of images that make up the dataset come from the same time period. The design style of the web pages of that period is different from today's web pages. This fact introduces a bias related to the evolution of design through time.

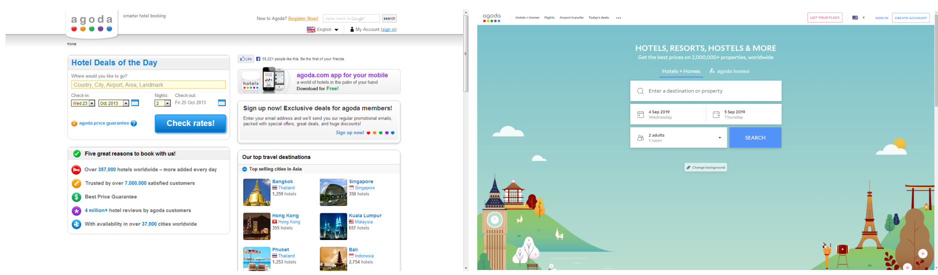
All these considerations led us to create a dataset aiming at mitigating these biases.

5.2 Creation of a new dataset

In this section, we are going to analyze how we proceeded with data gathering and how we structured the new dataset. Finally, we try to highlight why our dataset is better than Fiwi [1].

5.2.1 Web pages and Users

During the evaluation phase of the dataset described above, we wondered if and how it would be better to introduce some modifications to it. Since the beginning, we thought it was necessary to increase the total number of images present therein. In spite of the large variety of page layouts, no enterprise websites were still included. We also added new layouts, derived from the ones of the company, as they are useful to make the results as general as possible. In order to face the problem of the variability of design principles over time, we have chosen to introduce also the updated layouts of the pages present in Fiwi (Figure 5.1) in our dataset. Starting from the original layout core of 149 images, we have arrived to a total of 262 ones.



(a) (b)

Fig. 5.1: Old and new versions of web page layout included in the dataset: (a) old Fiwi version, (b) new updated version

The choice of how many and which users should be selected for data collection was not random. First of all, we felt it was necessary to reach more people than the Fiwi dataset. For this reason, we collected data from 100 volunteers to further include the differences of how different people look at images. The need to have more users made us focus also on which are the potential users of the websites under consideration.

We felt that collected data should come from both an enterprise and a more general environment. For this reason, 25% of the data collected comes from users within the company, who are used to work with websites. Also gender is balanced, with 50/50 male/female ratio. Due to time limitations, we were not able to balance the dataset by age groups, making it slightly unbalanced towards younger users. However, compared to Fiwi, we have a better representation of all age groups, without limitations to a small range of ages. The users' age ranges from 15 to 70 years old.

5.2.2 Data collection approach

When we faced the problem of collecting data, we wondered what was the correct way to reach as many users as possible. As already reported in the introductory chapters, data collection is entrusted to webcams and eyetrackers. From the beginning we thought about the potential of each of the two methods: the former allows a greater flexibility while the latter provides a better precision. At first, we explored the usage of webcams for data collection. Since every laptop is equipped with one of them, they would have allowed us to reach a very large audience without effort. Moreover, in Turkergaze [27], a methodology aiming at accomplishing this task is presented. Turkergaze is completely free and open source, based on a web interface. It is essentially designed to allow the collection of data on a large scale, because it can be easily executed on the user computers. The idea is to present data collection as a small quiz game to the user. In this game, he has to observe the layout images one at a time and remember which sites have been shown. The larger the number of recalled sites, the higher the score of the corresponding user. In the meanwhile fixation points of each layout are registered.

Our first attempts to use this approach have not turned out to be as positive as we had hoped. The webcam has proved to be very sensitive to environmental conditions, preventing the collection from running smoothly in most cases. Users had to make very limited head movements in order to complete the test successfully: it was often necessary to restart from the beginning by performing a new calibration phase. The high sensitivity of the system, the high failure rate and very noisy collected data led us to use an eye tracker.

Giving up using the webcam considerably limited the number of users we were able to reach. Having an easily scalable solution would have made it possible to get much more than 100 people's data. Nevertheless, we were able to create a larger dataset than those available in literature.

Two methods were possible for data collection using an eye tracker: either setting up a controlled environment where users would have had to go to take the test, or reaching potential volunteers. For organizational and timing constraints, we opted to use the latter method. This approach allowed users to stay in a comfortable position and made them more willing to take the test. This also granted us to collect more natural data, compared to a "laboratory" situation.

The data must reflect a real setting in which the user observes web pages; actually, a controlled environment, on the other hand, can cause a different user behavior. We asked users if they were willing to do a short test with the purpose of collecting data. Only with the user consent we could then proceed in setting up for the data collection.

The computer was connected to an eyetracker fixed to the base of the display and was placed on a horizontal plane (e.g. desk, table) in front of the user. The screen distance was variable according to the eyetracker's ability to correctly detect the eyes of the user. After explaining the task to the volunteer, we carried out a quick calibration of the device, assuring a high quality of the gathered data. Then, we started the data collection procedure, with an estimated duration of around 3 minutes. Each image appeared on the user's screen for four seconds. Images were interspersed with a black screen with a central black dot to allow the user to rest their eyes. When ready, user could press the space bar to continue to the next image. In total, every volunteer observed 30 web page layouts extracted from the image dataset. Our algorithm selects the images to display in such a way as to keep balanced the number of times users observed each page in total. In the following subsection, we analyze the data collected in more detail.

5.2.3 Dataset characteristics

For the purpose of our research, we did not had the need to collect the whole path of the user's gaze. We only focused on collecting fixation points. The shift of the eyes between two fixation points does not carry any useful information for our analysis, because it identifies the lack of importance of the crossed section. For each captured fixation point, the x and y coordinates and the timestamp in which it was observed were recorded. Of course, each path consists of a sequence of fixation points, associated with a user id and the observed image. Each of the 262 images in our dataset were viewed by 11 or 12 users in total. Since each user observed 30 images, our dataset contains a total of 3000 gaze paths. Compared to the Fiwi dataset, the number of available paths is more than twice, allowing us to have an even more solid basis for training the path prediction model. Saliency map prediction, instead, uses very similar maps for the sites also present in the other dataset. However, it also has a greater number of total web sites. In the following, we analyze in more detail how this raw data was pre-processed.

5.2.4 Saliency Map generation from raw data

Immediately after the completion of the data collection, we had a CSV file containing all the fixations collected. We performed a first cleaning phase of the gathered data, deleting all those values that were clearly the results of errors (e.g., NaN values). To create a saliency map of a particular image, it was first necessary to create a fixation map. This is nothing more than an image with a totally black background, on which white pixels are drawn in these corresponding points where a fixation took place. All the fixations observed by users who looked at that specific image are taken into account and drawn on it (Figure 5.2).

Switching from a fixation map to a saliency one requires to perform a convolution with a 2D Gaussian filter on the image containing the fixation map. We took Fiwi as the reference. It uses a filter with a standard deviation of 25 pixels. Once the saliency maps were generated, we used them as ground truth to train our models (Figure 5.3).

5.2.5 Gaze Paths preprocessing

As in the case of saliency maps, we needed to clean fixation data from the values not compatible with the dataset, caused by collection errors. In this case, we group the set of paths that we collected for each image.

Each image has associated a number of paths equal to the number of users who observed it. In this case, we do not have a single ground truth, because images are observed differently by each person. Our network will then try to generate a plausible path that takes into account the main features present in real paths.

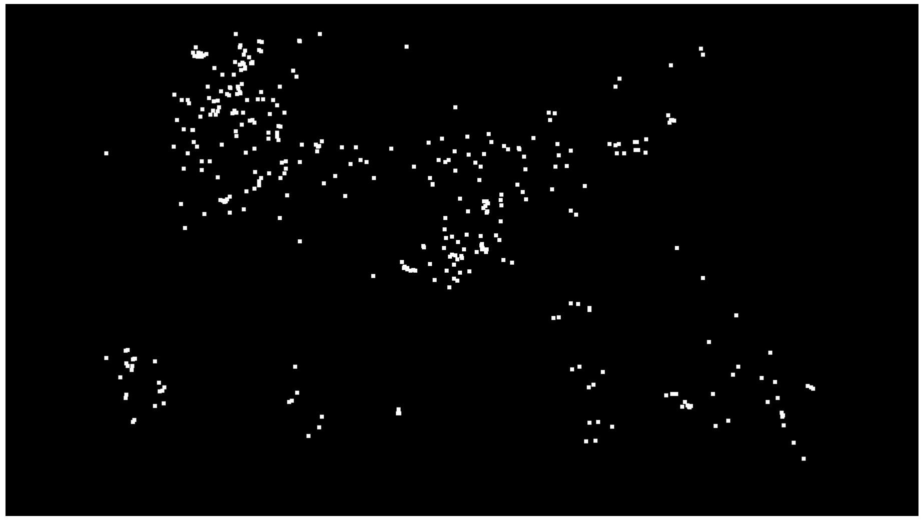


Fig. 5.2: Example of fixation map - observe the presence of several white dots at the center and the upper left corner of the figure

Fig. 5.3: Example of a saliency map

5.2.6 Imporvements w.r.t. the Fiwi Dataset

One of the biggest limitations of the Fiwi dataset is the limited variety of people who participated in the data collection. Our dataset collects data from 100 people, ensuring a better gender distribution and a wider age range. In addition, we have also increased the number of web pages contained, and we introduced enterprise websites.

Overall, more than twice the number of Fiwi's eye paths were collected, in order to improve the amount of data available for training.

We also removed the bias of having an overly controlled environment that can potentially change the user's behavior.

As a final improvement, we avoided to over-stress users with data collection, making it faster and more immediate.

Further improvements can still be done, especially with regard to increasing the overall size of the dataset by expanding both the number of users and the number of website layouts stored therein.

6

Saliency Map Generation: proposed architecture

In this chapter, we expose the approach that we used to create a model able to predict the saliency map of an image of a graphical user interface of a website. We start by analyzing the architectures of neural networks that we used as a starting point for our research; then, we get to describe our proposed solutions.

6.1 Reference Architectures

As already introduced within the state of the art, the prediction of saliency maps is not an unexplored research field. Various solutions have been proposed in natural image analysis, but just a few of them have been applied to graphical interfaces of websites. In the following section, we are going to analyze in detail the architectures from which we have taken our inspiration from.

6.1.1 TSGAN for saliency map prediction

TSGAN, short form for Two-Stage Generative Adversarial Network, is a neural network architecture first proposed in [2] for predicting saliency maps of websites. It belongs to the GAN family, and has been modified to better fit the domain of web pages. The network's generator has a two-level architecture in order to generate more accurate maps (Figure 6.1). This feature gave rise to the name of this network.

The first stage, given an input image, produces a raw saliency map. This output is fed, together with the original image, to the second stage in such a way as to generate a finer prediction. Both stages are autoencoders which share all the weights, except the first layer due to the different input size. In addition, the shared part of the encoder uses the pre-trained VGG network weights in order to reduce the total number of variables to be trained. Given the particular structure of a website between the encoder and the decoder of the generator, an image containing the outlines of the elements of the analyzed web pages is also inserted as input to facilitate predictions. TSGAN is more specifically a Wasserstein GAN, allowing for more stable training. The discriminator receives both the generated saliency maps and determines whether they are realistic or not. Compared to the classic

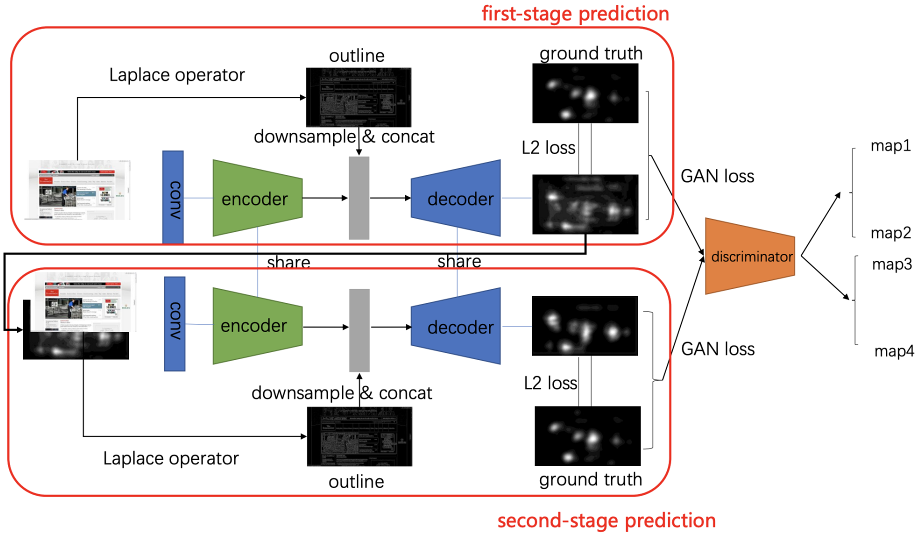


Fig. 6.1: TSGAN architecture

architecture of a GAN, the discriminator produces a probability map as output; the highest probability areas are the most realistic ones. This network reaches the state of the art in predicting saliency maps in the domain of graphical user interfaces.

6.1.2 SalGAN

SalGAN [3] is a neural network designed for saliency map prediction in the natural image domain. It is not thought to be applied to the specific domain of graphic interfaces; thus, we want to investigate how it behaves with respect to an architecture designed for this purpose. SalGAN is a GAN with a much simpler architecture than TSGAN (Figure 6.2).

The generator is a simple single stage autoencoder that generates saliency maps from input images. The discriminator receives both generated and real images and has to identify which of them are coming from the real data distribution. SalGAN's loss is built around the standard GAN loss function, customized to obtain better results on images. Authors have also published pre-trained weights of their network on a dataset made of natural images. Analogously to what happens to TSGAN, the layers corresponding to the encoder inside the generator match with the structure of the first layers of the VGG network. Pre-trained weights on the Imagenet dataset are also loaded to achieve better convergence and have less weights to train. This approach achieves very good results in the prediction of saliency maps on the MIT300 dataset, and proves to be the best GAN present among the published results.

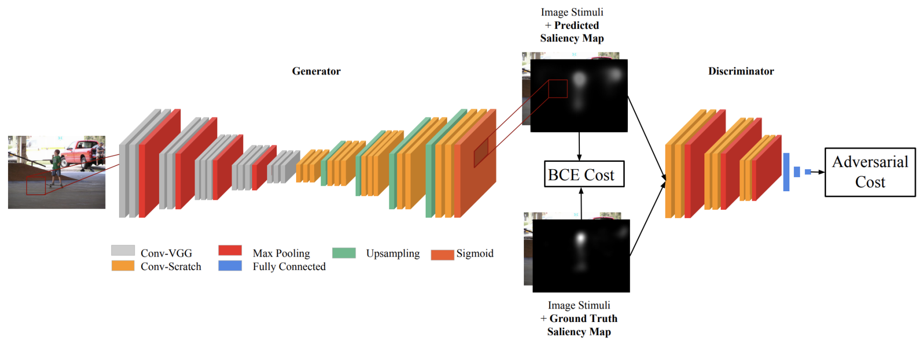


Fig. 6.2: SalGAN architecture

6.2 TSGAN reimplementation

In order to take advantage of the architecture proposed in [2], it was necessary to proceed with the re-implementation of the whole neural network, because the source code was not released by the authors. In order to rebuild this articulated architecture, we decided to use the Tensorflow framework for flexibility reasons. We followed all the indications in the paper, introducing further improvements. We improved the TSGAN Wasserstein loss used by the network with the more recent Wasserstein loss with Gradient Penalty, to be sure to obtain the best results. We have also implemented different versions of the architecture, along with the original one, trying to improve the paper results. In the following subsections we will analyze in more detail the different network implementations we have created during our research. We will start by explaining in more detail the structure of TSGAN in order to better understand the choices we made during our re-implementation of the network.

6.2.1 TSGAN in detail

TSGAN's architecture was specifically designed to work with images of web pages. Let us start by describing in more detail the basic structure of the various parts of the network. As briefly mentioned in the previous section, the generator has the structure of an autoencoder. The encoder that is part of it has a fairly standard structure, with pairs of convolutional layers interspersed with max pooling. The purpose is to extract a set of feature maps useful for the generation of saliency maps.

At the output of the encoder, the authors have chosen to introduce a feature map that collects information about the edges of the elements of the web page in order to keep information about the structure of the page. It is concatenated with the encoder output feature map. The set of these feature maps are given as input to the decoder, whose task is to generate the actual saliency maps. It uses pairs of dilated residual blocks, interspersed with a single deconvolution layer, in order to recreate an image of the original size.

The autoencoder we just described is common to both stages of the generator, including all the weights. Only the first layer of convolution is not shared because they have different inputs according to the stage into consideration. In Figure 6.3 we can observe in detail the characteristics of the various layers.

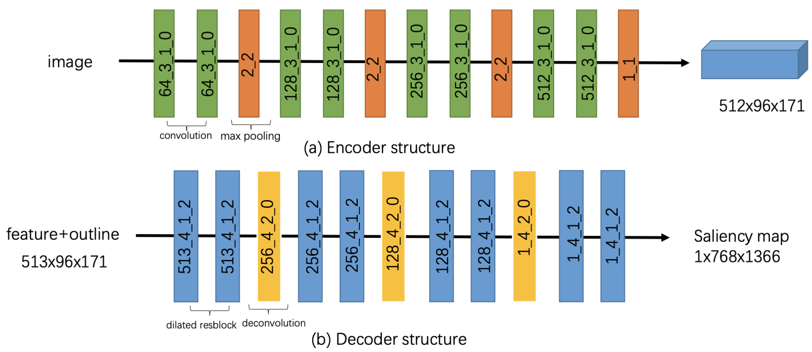


Fig. 6.3: Generator autoencoder architecture: (a) encoder structure, (b) decoder structure. Each layer has an overlayed sequence of numbers that represent: the number of channels, the kernel size, the stride and the holes. Pooling layers have only the kernel size and stride

The first stage of the generator is fed with the image of which we want to generate through the saliency map. The authors have decided to keep the original size of the dataset images which they used during training. In their paper, they used the Fiwi dataset that contains images consisting of 1366×768 pixel. The second stage, instead, together with the same image received by the first stage, adds the saliency map predicted by the previous step. The generated saliency maps are fed to the discriminator, who must evaluate their quality. The discriminator, as already mentioned, generates a probability map in order to identify whether the various areas of the generated saliency map are realistic. Figure 6.4 shows the detailed characteristics of the various layers of the network.

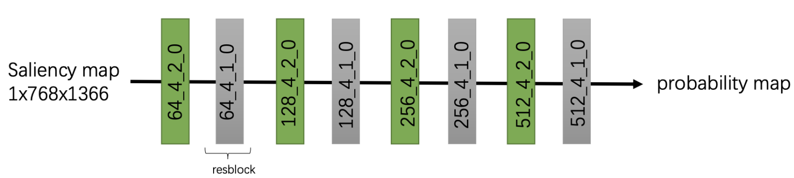


Fig. 6.4: Discriminator architecture. Each layer has an overlayed sequence of numbers that represent: the number of channels, the kernel size, the stride and the holes

The loss function used in TSGAN aims to make the convergence possible in this particular domain. It is made up of two main terms. The first term is a simple L2 loss, which performs a comparison between the generated saliency maps and the ground truths. This term helps a lot the convergence as it avoids big deviations from the original saliency map. The second term is the adversarial loss, typical of every classic GAN, used to measure how the generator and the discriminator are behaving against each other. Since our network consists of two stages, we deal with a more complex loss. The L2 loss is computed only for the images generated by each of the two generator stages. Therefore, the discriminator is not affected. By contrast, the adversarial loss is calculated considering both the generator and the discriminator. Also in this case, it is computed for both generator stages. The final result is shown below.

$$Ł_1(x, s) = \sum_{c=1}^{C_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \frac{(s_{i,j} - G_1(x)_{i,j})^2}{2W_l H_l C_l} \qquad [6.1]$$

$$Ł_2(x, s, \hat{s}) = \sum_{c=1}^{C_l} \sum_{h=1}^{H_l} \sum_{w=1}^{W_l} \frac{(s_{i,j} - G_2(x, \hat{s})_{i,j})^2}{2W_l H_l C_l} \qquad [6.2]$$

$$Ł_3(G_1, D) = \underset{G_1}{min}\ \underset{D}{max}\ \mathbb{E}_{s\sim p_{(s)}}[D(s)] + \mathbb{E}_{x\sim p_{(x)}}[1 - D(G_1(x))] \qquad [6.3]$$

$$Ł_4(G_2, D) = \underset{G_2}{min}\ \underset{D}{max}\ \mathbb{E}_{s\sim p_{(s)}}[D(s)] + \mathbb{E}_{x\sim p_{(x)}}[1 - D(G_2(x, G_1(x)))] \qquad [6.4]$$

$$Ł_{tv}^{\alpha}(x) = \sum_{i,j}((x_{i,(j+1)} - x_{i,j})^2 + (x_{(i+1),j} - x_{i,j})^2)^{\frac{1}{\alpha}} \qquad [6.5]$$

$$Ł(G_1, G_2, D) = \lambda Ł_1 + \lambda Ł_2 + \mu Ł_3 + \mu Ł_4 + \lambda Ł_{tv}^{\alpha} \qquad [6.6]$$

Equations [6.1] and [6.2] represent the loss L2 of the first and second stage of the generator, respectively. Equations [6.3] and [6.4] represent the adversarial loss of the two stages, while Equation [6.5] represents the total variation regularizer. The objective function can be expressed in an overall way like in Equation [6.6]. Note that each loss is multiplied by a constant indicating the weight that a particular term has inside the function. The paper sets λ to 0.1 and μ to 1.

6.2.2 Architectural enhancements

Since there is no pre-trained model available, we had to proceed in training the model from scratch with our dataset. The architecture we used during our research has some differences compared to the one reported in [2]. Our improvements try to produce better results than those obtained by the creators of the network. The first choice to make was the reduction of the size of the input images. At a first sight, the choice to work with real size images might be the best one. However, in the training phase, it requires a considerable amount of resources. We chose to downsample the images of our dataset to a size of 640×384. This allowed us to work with a larger batch size, thus accelerating convergence.

Then, we have chosen to use WGAN-GP, instead of simple WGAN, since better results are reported in [16]. The loss formulation has remained identical to the one described in TSGAN's paper, with the addition of a term that deals with penalizing the gradient norm. We also decided to remove the total variation regularizer from the loss because it introduced instability in the training process. The new objective function replaces Equation [6.5] with the gradient penalty expressed in Equation [6.7].

$$Ł(\hat{x}) = \lambda \mathbb{E}_{\hat{x} \sim \mathbb{P}_{\hat{x}}}[(\left\|\nabla_{\hat{x}} f_w(\hat{x})\right\|_2 - 1)^2] \qquad [6.7]$$

In this formula, $\hat{x}$ is the averaged combination of the generator output and the ground truth. We have set λ to 10, as it is reported in the original paper. In our specific case, we decided to use only the output of the second stage of the generator to compute the gradient penalty, because it is the most interesting output for us.

With regard to the network architecture, we have chosen to modify the structure of the discriminator. The fact that reference TSGAN produces a probability map with many channels made us consider to modify the last layers of it. Instead of having an output of 512 channels, we used a convolutional layer to reduce the dimensionality of the map itself. This allowed us to have a 2D map in which each pixel corresponds to an area of the image: the intensity represents how much a certain area of the image is realistic or not. The probability map has a reduced size, compared to the real image, because there is no one-to-one pixel correspondence with the saliency map taken into consideration.

Then, we went through a more radical approach that led us to totally change the structure of the discriminator. We modified the layers in such a way that the discriminator had the architecture similar to that of an encoder. The dimensionality of the input was reduced in such a way as to obtain a two-dimensional map as output. Also in this case, this map represents the reality of the various areas of the generated saliency map. It is possible to view the modified discriminator architecture in Figure 6.5.

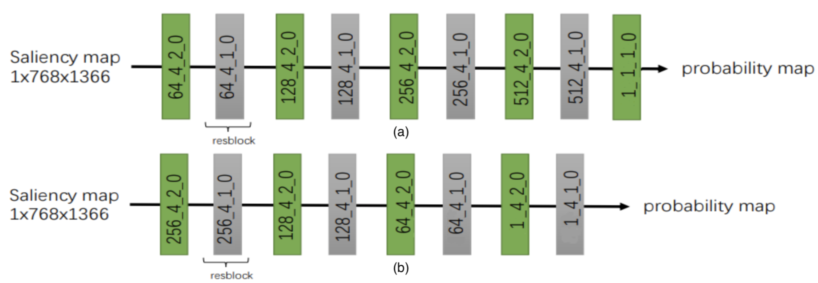


Fig. 6.5: Discriminator's architecture modifications: (a) represents the original architecture with a final convolution layer added, (b) represents the radically modified discriminator. Each layer has an overlayed sequence of numbers that represent the number of channels, the kernel size, the stride and the holes

The changes we had to make also concerned the training procedure, in order to improve the convergence of the model. As reported in [16], we had to verify that the critic (i.e., discriminator) was trained with a different frequency than the generator. This step is necessary to make sure that the gradient penalty has positive effects. Each training step of the generator is balanced by 5 successive training steps of the discriminator, in order to avoid that the discriminator weakens too much, being already penalized by the gradient penalty. The paper also states the need to remove all the batch normalization layers from the network, since the gradient penalty is computed on each input element independently, and not on the whole batch. We also added some Gaussian noise on the input saliency maps of the discriminator in order to avoid overfitting. We have chosen to add a very small noise to prevent the model from not converging.

6.3 SalGAN Fine Tuning

Unlike TSGAN, SalGAN is designed to work in the real image domain. The pre-trained model provided allowed us to consider fine tuning the network. To verify if fine tuning actually leads to better results, we chose to also perform a total network retraining. We only focused on fine tuning the model in the best possible way leaving most of the network structure unchanged.

We did not consider necessary changing the architecture, as it already performs very well when applied to real images. Furthermore, we left the structure of the loss unchanged; it is similar to that of TSGAN with one part dedicated to measure the content loss and one to measure the adversarial loss [3].

The main requirement we had was keeping frozen the layers part of the encoder inside the generator: they use the pre-trained VGG network structure and weights, as this technique proved to speed up the convergence of the model.

TSGAN also used this approach improving enormously the amount of time required to achieve a good training.

Let's start by specifically analyzing how we have modified the training procedure to perform the fine tuning of the network. In addition to the first part of the generator of which the VGG pre-trained network is part, we have also opted to freeze the first four convolutional layers of the discriminator.

This allows a complete re-training of the second part of the generator, i.e, the decoder, in which the saliency map is generated, adapts itself to the web pages domain. On the other hand, freezing the first layers of the discriminator keeps almost completely intact its ability to distinguish between real and fake saliency maps. In fact, if we compare two saliency maps, one from the domain of natural images and one from the domain of web pages, it is not easy to determine that they actually come from two completely different domains. Saliency maps from different domains can be considered similar, as all of them have a very similar structure.

The features that the discriminator has learned to consider as discriminating to assess the quality of a saliency map will be common in both domains. That is why it is important to preserve what the network has learned previously, preventing the training of these first layers.

In this way the training improves the quality of the produced saliency maps having a discriminator already with a lot of "experience".

In Figure 6.6, we can see which are the layers of the network that have been frozen. The number of optimal layers to keep out of training was obtained after making preliminary observations on the quality of the generated images.

In addition, we decided to lower the learning rate of the neural network from 3×10^{-4} to 1×10^{-4}, allowing a more gradual convergence towards an optimal

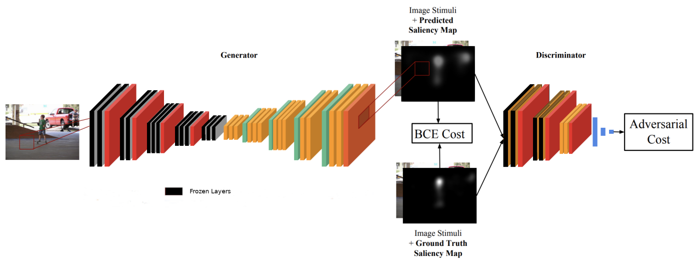


Fig. 6.6: SalGAN frozen layers during training.

solution. We left the main training workflow as in the reference architecture as it already proves to be capable of producing good results. To verify that our fine-tuning procedure was bringing benefits, we also tried to continue the weight training without introducing any layer freeze. We will compare the results obtained in the section concerning the evaluation of the models used.

7

Saliency Map Generation: result evaluation

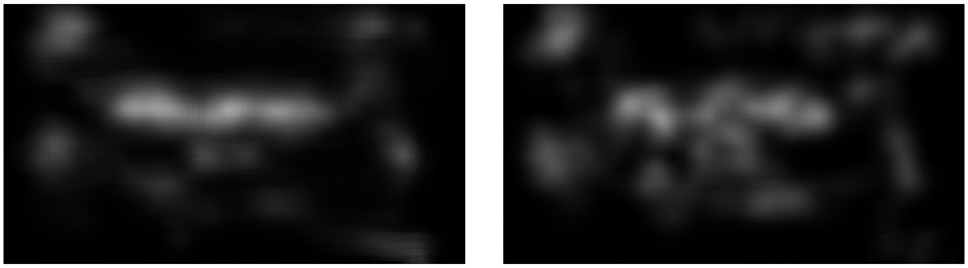
In this chapter we evaluate how the proposed models perform. We first introduce how the training losses behave in both TSGAN and SalGAN. Then, we underline potential issues, such as overfitting. Finally, we compare the results achieved by each version of the models.

7.1 TSGAN training analysis

As already introduced in detail in the previous chapter, TSGAN's training procedure was designed following the Wasserstein GAN guidelines. All variants of the model we have created are also WGANs, with the addition of gradient penalty to improve performance. Our models have been trained using 80% of the dataset, leaving out the remaining 20% for validation purposes. During training, we have monitored the loss variations of both the generator and the discriminator; the goal was to promptly identify any problems preventing the convergence of the network.

Since TSGAN did not have pre-trained weights publicly available, the first training iterations were quite standard: the discriminator and generator losses started dropping sharply due to the early stages of the process. During this phase we could immediately appreciate how the generated saliency maps started to take shape, improving the quality as the training progressed. We can see an example of this in Figure 7.1.

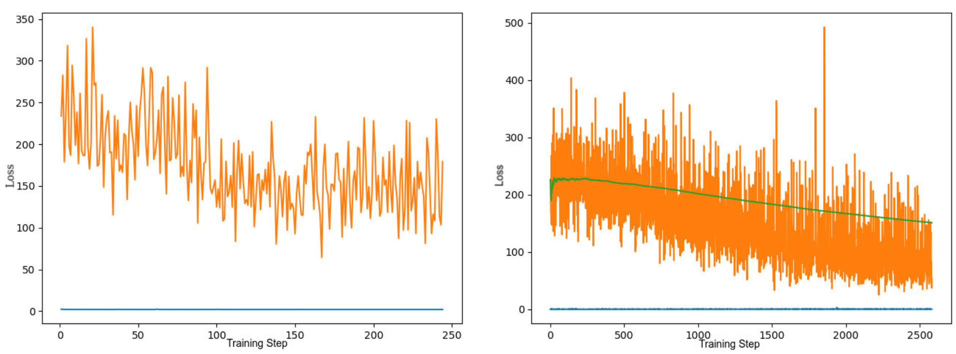
The discriminator, from the very first iterations, proved to be much more effective than the generator. The WGAN-GP architecture prevents the generator from becoming too weak against the discriminator. We have also chosen to introduce noise on the input saliency maps to keep the discriminator from overfitting on the training set, further improving the stability of the training process. During training, the generator goes on with minimizing its loss boosting the maps' quality at every training step. Over time the generator's predictions become more and more similar to those of the training set. After several training epochs the generator overfits on the training set; it is required to suspend the training because no learning is performed by the generator. There is no improvement in the quality of the generated samples making it senseless to proceed. We identified the source of this problem in the limited dimension of the dataset. We made several attempts to remove this



(a) Epoch 30 (b) Epoch 90

Fig. 7.1: Comparing early training saliency maps with the final result.

phenomenon. We tried to add noise on the input images in order to simulate data augmentation. The results did not live up to expectations as the quality of the generated samples was significantly reduced. Further research can be done in this regard as we do not feel that we have explored all possible paths. Since there is no objective way to determine whether the GAN training has reached an optimal solution, we have relied on assessing the quality of the generated samples qualitatively. We have also taken into account the loss calculated on the validation set to understand if our network was achieving some benefits from the training process.



(a) Normal GAN loss (b) WGAN-GP Loss

Fig. 7.2: Graphs comparing Generator loss trend. (a) WGAN-GP assures minimization while (b) the normal loss reaches a constant state. X-axis represents the training step number while Y-axis denotes the loss value. Losses are not on the same scale.

We also wanted to verify whether the use of a WGAN-GP would improve the training of the network or not. As we can see from Figure 7.2, WGAN-GP manages to constantly minimize the loss of the generator. This does not happen in the classic GAN, where, after an initial learning phase, the process is blocked due to the excessive ability of the discriminator. The considerations just made apply to all the network variants we have proposed. In one of the following sections we will analyze the quality of the saliency maps generated more specifically, using the metrics available in literature.

7.2 SalGAN training analysis

The training procedure used during fine tuning and retraining of SalGAN is the same as described in the reference paper. Since, in this case, the source code of the network is publicly available, we have only adapted it to our needs. As SalGAN is based on the classic Generative Adversarial Network formulation, the authors have taken steps to avoid potential non-convergence of the model. Weight regularization was used by them to avoid potential instability during training. Also in this case we have monitored the loss trend to determine if the training was proceeding in the right direction. We observed the loss on the validation set to see if there is a general improvement on the generated predictions. The qualitative evaluation of predictions remained essential to understand when training could be stopped. In the following section, we compare the various models introduced, determining which one performs best.

7.3 Models evaluation

In this section, we compare the quality of the predictions of the two models analyzed, in each of their variants. We begin the analysis starting from TSGAN variants, and, then, continue with the different versions of SalGAN. Finally we determine which of the two models performs better. To make these considerations we will use the metrics introduced in Section 4.4, namely: NSS, Pearson Coefficient, KL-divergence, AUC-Judd and AUC-Borji.

7.3.1 TSGAN-based models

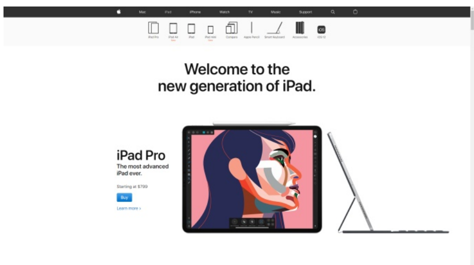
We have created three TSGAN different variants with the aim of comparing them. The first model (hereafter, Reference) is the one created following the original TSGAN paper [2]. No improvements have been added: we followed the described structure step by step. The second variant (hereafter, WithConv) taken in analysis is the one obtained by adding a final convolution layer to the TSGAN's discriminator to generate a 2D map representing the prediction quality. Finally, the third model (hereafter, AltDiscr) is the one with the structure of the discriminator totally modified, in the way we described in Section 6.2. Now, we start our analysis by focusing on the quantitative performance of the different models. In Table 7.1 the values obtained for the various metrics on the test dataset are reported.

All the values are very close to each other. The changes we have introduced have not contributed to improve the results. Although the values are very similar, it is clear that the Reference model is the one that performs best. It gets the highest score in all the metrics considered.

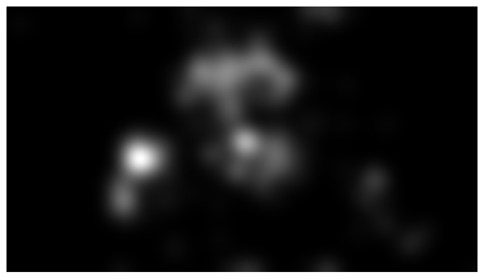
	NSS	AUC_Judd	AUC_Borji	CC	KL
Reference	**1.43**	**0.82**	**0.76**	**0.66**	**0.63**
WithConv	1.39	**0.82**	**0.76**	0.62	0.75
AltDiscr	1.4	0.81	**0.76**	0.62	0.95

Table 7.1: Values of the metrics obtained by TSGAN models

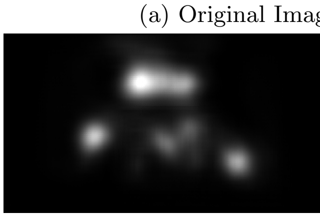
Further considerations can be made on the quality of predictions. If we compare the generated saliency maps, we see how the metrics' values are realistic. There is no clear winner among the predictions of the various models. If we compare them with the map representing the ground truth, we observe once again how difficult the choice of the best prediction is (Figure 7.3).



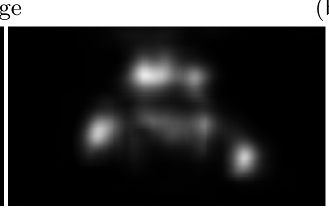
(a) Original Image



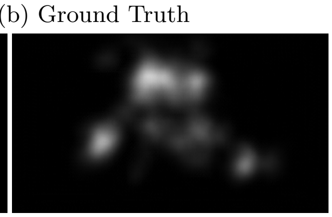
(b) Ground Truth



(c) Reference model



(d) WithConv model

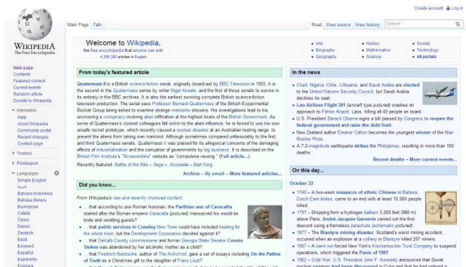


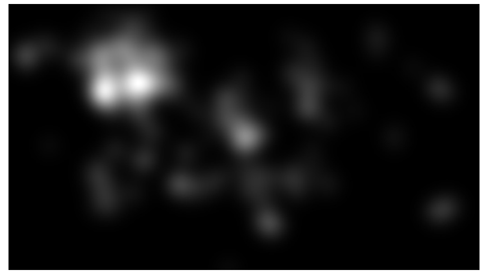
(e) AltDiscr model

Fig. 7.3: Comparison of the TSGAN variants on a layout rich of images

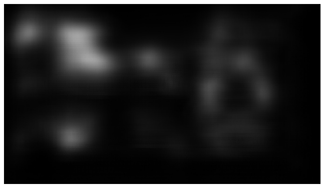
All models are able to identify the salient areas of the image, sometimes detecting false positives. This first image is simple for the network since there are only few elements in the page layout. If we analyze, for example, a densely written website, the behavior of the network suffers a slight worsening (Figure 7.4).

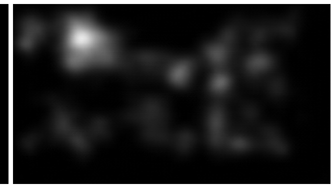
Each model is able to understand which is the most important area of the image, but the false positives are even more evident than in the previous case. A comparison of the metrics calculated on individual images of the Reference model let us draw some interesting considerations. In the first image, NSS obtains a very high score (1.88), much higher than the average performance of the model. In the second image the metric score is 1.67 much lower than in the first one. Also AUC-Judd and AUC-Borji undergo a considerable decrease: the former goes from 0.85 on the first image to 0.79, while the latter goes from 0.8 to 0.76 on the second image. The performance decrease is tangible, but still all the metrics are above the model's average.





(a) Original Image (b) Ground Truth

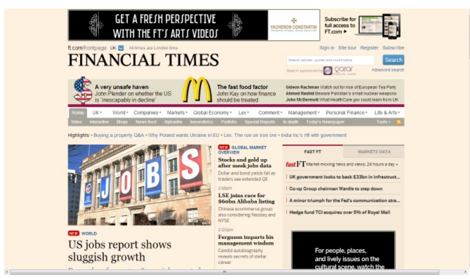


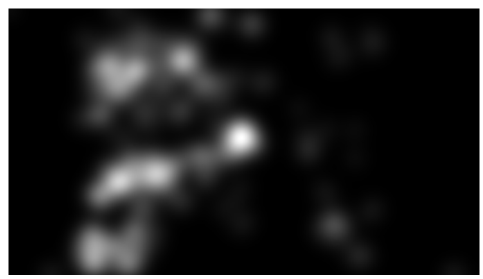


(c) Reference model (d) WithConv model (e) AltDiscr model

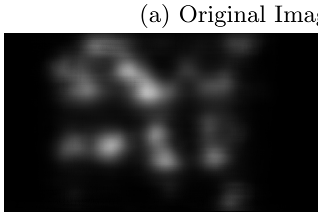
Fig. 7.4: Comparison of the TSGAN variants on a layout rich of text

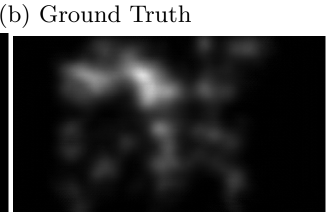
The model suffers even more with web pages that mix large amounts of images and text: we have identified it as the weak point of the model. Let us take the image in Figure 7.5 into consideration.





(a) Original Image (b) Ground Truth





(c) Reference model (d) WithConv model (e) AltDiscr model

Fig. 7.5: Comparison of the TSGAN variants on a layout rich of images and text

False positive areas are now very common. The network has problems detecting also the actual salient areas. None of the models performs with sufficient accuracy.

A metrics' comparison calculated on this specific image underlines a huge performance drop. NSS on the Reference model scores just 1.0, going below the model performance average. A same behavior characterizes all the other metrics; all of them are below the average. Therefore, we can conclude that the model struggles with web pages densely filled with images and written text because numerous potentially salient areas are identified. In the following, we go on to analyze how SalGAN behaves.

7.3.2 SalGAN-based models

As already done for TSGAN, we compare the different SalGAN variants we have created. This time, each model version does not differ on the architecture's structure, but, instead, on the different training approaches. We evaluate four SalGAN models in total, each with a different training approach. The first model (hereafter, Reference) is the one provided by the creators of SalGAN; it uses pre-trained weights on natural images. The second model (hereafter, FineTuned) is the version of the network that we fine-tuned by ourselves. The third model (hereafter, KeepTrain) is simply the Reference model that we kept training, using our dataset, without any fine tuning. The fourth and last model (hereafter, FromScratch) has been completely re-trained with our dataset. This comparison helps us to understand which of the four methodologies is more successful. Table 7.2 shows the metric values for the four models.

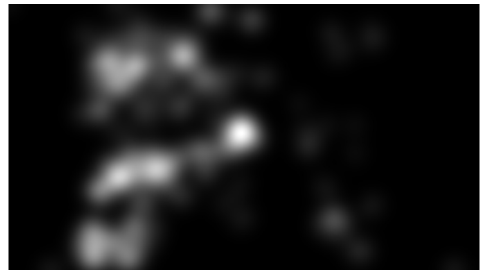
	NSS	AUC_Judd	AUC_Borji	CC	KL
Reference	1.25	0.8	0.76	0.56	0.9
FineTuned	**1.61**	**0.85**	0.82	**0.74**	**0.52**
KeepTrain	1.58	0.84	**0.83**	0.73	**0.52**
FromScratch	1.49	0.83	0.8	0.68	0.65

Table 7.2: Values of the metrics in the four models

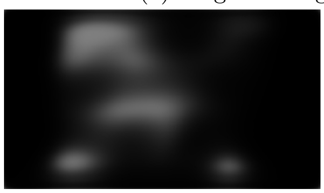
The metrics tell us that the worst performing model is the Reference one. Since SalGAN was previously trained only on natural images, the change of domain causes a significant performance drop. Web layouts are much more complex than natural images: we can find several natural images and text together in the same page. Since this model has a worse behavior than any other one seen so far, we will not take it into account for a qualitative analysis of the results.

The other three models prove to have a great ability in predicting saliency maps. All of them have similar or higher metrics values than TSGAN. Overall, we notice a considerable superiority of the model that has undergone fine tuning. In some metrics the performance is also superior to the one achieved in the natural image domain [3] in the MIT300 dataset. We can observe that both of the models previously trained on natural images benefit a lot in the results. Our limited dataset is not able to make the models generalize well on all sorts of web page layouts. In fact, the model FromScratch reaches similar performances to the ones of TSGAN. Furthermore, websites are very often rich of natural images, providing both FineTuned and KeepTrain with a big advantage. In the following, we analyze the predictions qualitatively.

(a) Original Image

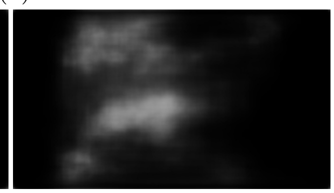


(b) Ground Truth



(c) FineTuned model

(d) KeepTrain model



(e) FromScratch model

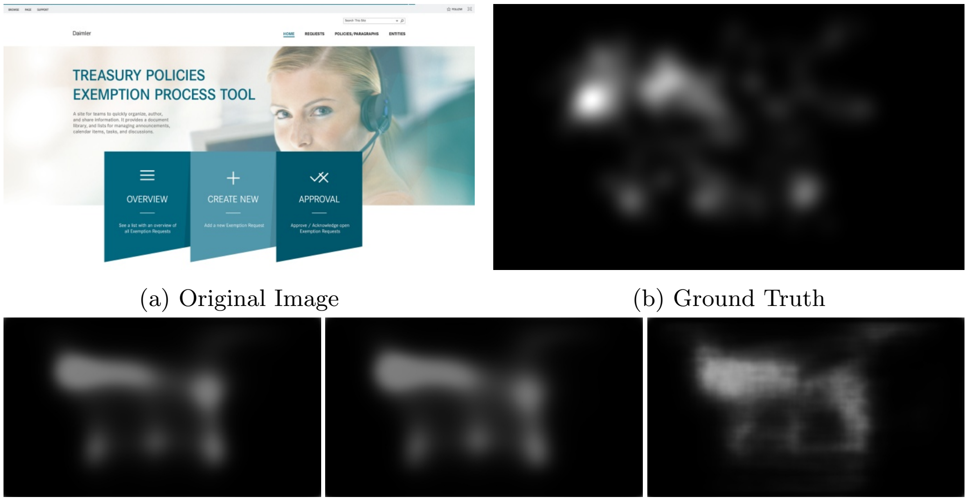
Fig. 7.6: Comparison of the SalGAN variants on a layout dense of images and text.

In Figure 7.6, we can compare the quality of predictions. We took into consideration one of the images that we previously analyzed with TSGAN. The quality of the saliency map is higher than the one generated by the previous model. The highlighted areas include well those in the ground truth. The model that has the poorest performance is the one trained from scratch on our dataset. A large amount of noise is visible, which does not disappear even in advanced training epochs. Again the metrics highlight the fine tuned model as the one with the highest performance. If we compare the metrics calculated on this specific image, FineTuned and KeepTrain are very close to each other with a difference of a few decimals to the advantage of the former. In general, SalGAN does not mark very specific areas of the image but rather larger portions of the layout.

In Figure 7.7, we study other examples of saliency maps. More than before, SalGAN favors wider predicted saliency areas in spite of the probability intensity. All three models perform very similar, by introducing a lot of false positive areas. However, all the parts of the ground truths are predicted correctly. In the following subsection, we will compare the two models directly to estabilish weaknesses and strengths of each of them.

7.3.3 Comparison of TSGAN and SalGAN

In the last two subsections, we limited ourselves in comparing the performances of different versions of the same model, without directly comparing TSGAN and SalGAN architectures. It is very clear how different the predictions are: SalGAN



(a) Original Image (b) Ground Truth

(c) FineTuned model (d) KeepTrain model (e) FromScratch model

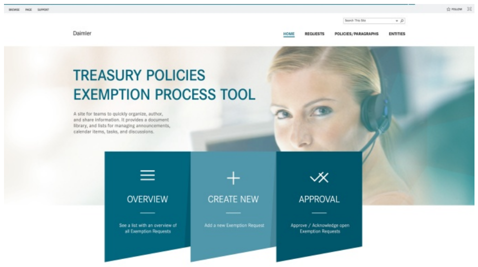
Fig. 7.7: Comparison of the SalGAN variants on a layout rich of images

predicts wider areas with lower probability, while TSGAN tries to be as specific as it can when predicting the intensity of each area of the saliency map. As we can observe from Tables 7.1 and 7.2, the fine tuned version of SalGAN outperforms every version of TSGAN we have analyzed. But how can we compare them qualitatively?

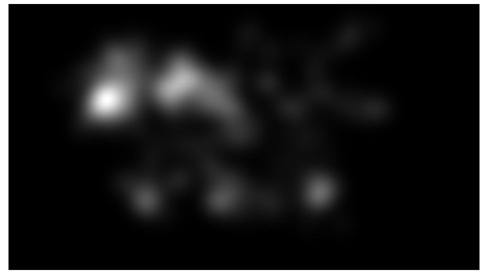
In Figure 7.8, a representation of how the two architectures perform is reported. Although models score similar metrics on this specific image, SalGAN is actually not performing very well, compared to TSGAN. Many areas to the left of the image are marked as false positives. In addition, the most salient areas are highlighted as one large homogeneous area. On the other hand, TSGAN achieves a better performance as it minimizes the defects found in SalGAN. If we just analyze this image we might think that TSGAN actually performs better. The situation changes dramatically when layouts with rich textual information and images are considered. In the previous sections we have identified this as TSGAN's weak point.

The Figure 7.9 shows how SalGAN is able to offer a much higher quality prediction in this case. TSGAN struggles to identify the salient parts producing an almost completely wrong map. If we compare the metrics calculated on this image, we obtain very distant values that, once again, benefit SalGAN. The metrics that most highlight this difference are NSS and AUC-Borji: TSGAN scores 1.02 and 0.72 respectively, while SalGAN scores 1.25 and 0.81. The difference in AUC-Borji value confirms how TSGAN struggles to find the salient areas, introducing many false positives.

As a consequence, we can state that SalGAN performs generally better than TSGAN on a wider variety of layouts. TSGAN suffers when working with pages rich of information, where every single element could be a highlight. On the other hand, SalGAN is not able to provide too detailed information about salient areas and merely identifies large areas that are equally likely. TSGAN generally proves to be better in all those layouts where there is a scattering of information, ensuring better



(a) Original Image



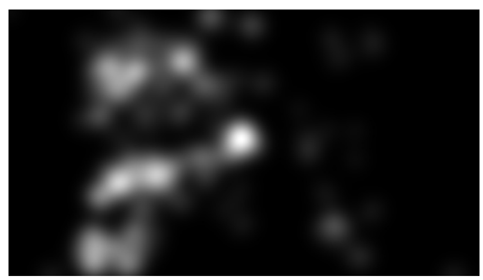
(b) Ground Truth

(c) Ground Truth

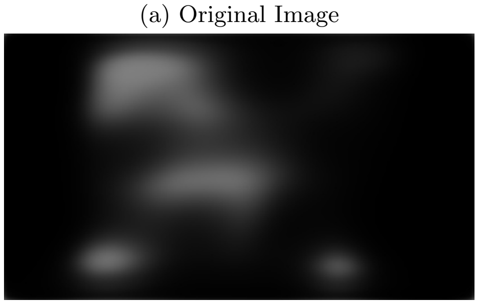
(d) Ground Truth

Fig. 7.8: Comparison of the prediction between the best performing model variants of SalGAN and TSGAN on a layout rich of images

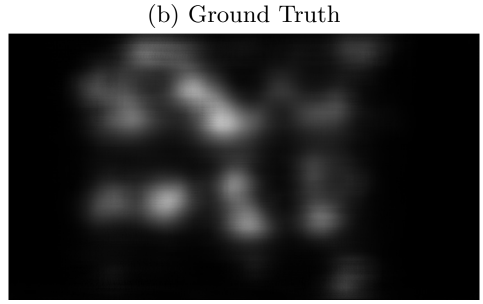
(a) Original Image



(b) Ground Truth



(c) SalGAN Fine Tuned



(d) TSGAN Reference

Fig. 7.9: Comparison of predictions between the best performing model variants of SalGAN and TSGAN when they operate on a layout dense of images and text

detail. As SalGAN is already trained on a larger dataset, it is able to generalize better. Our limited dataset does not allow TSGAN to work well with every type of website, sometimes producing wrong outputs. Therefore, after having compared metrics and output quality, we believe that SalGAN is the preferred model in most situations.

8

Gaze Path Prediction

In this chapter, we discuss on how it is possible to predict the gaze path of users looking at web pages. We propose a model able to perform this task taking inspiration from what is available in literature. Then, we evaluate the results obtained using metrics applicable to this domain.

8.1 Reference Architecture: PathGAN

Scientific research, in the field of eye movement prediction, did not achieve many important results yet. The lack of annotated datasets does make the task of creating new models not easy: no data very often leads to the impossibility of performing a successful training. All the solutions proposed so far apply to the prediction of gaze on natural images. Among the limited studies in this field, a Generative Adversarial Network, called PathGAN, stands out for its results. PathGAN [4] predicts the visual scanpath of people observing images, both in normal and in 360° format. The quality of its results places it as one of the best performing models in this domain.

In the following subsections, we examine this architecture in more detail and we focus on which are the major changes we have introduced. Then, we describe how the model behaves during the training procedure.

8.1.1 PathGAN architecture

As already mentioned, PathGAN is a Generative Adversarial Network. It belongs to the Conditional GANs family, due to its particular structure. Figure 8.1 represents PathGAN's architecture.

The first part of the network is a generator: an input image is fed to obtain a gaze path. This output represents the route of the eyes of a potential user who observes that specific input image. The generated path is a sequence of 63 fixations. Each fixation consists of a tuple of four elements: a x-coordinate, a y-coordinate, a timestamp and an end of path probability. This last element allows the generation of paths of variable length: a threshold is set on it to determine which fixations

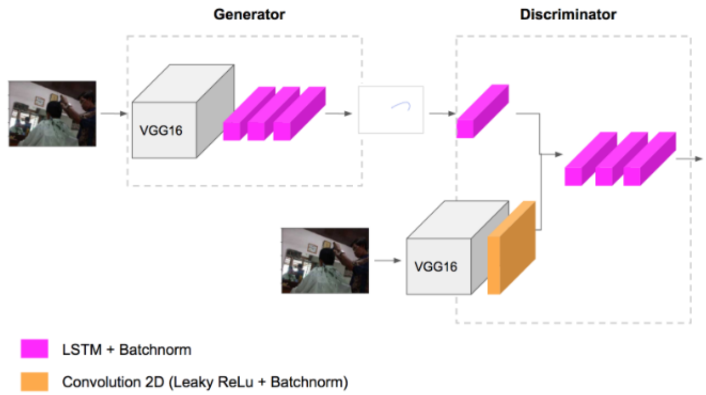


Fig. 8.1: PathGAN architecture

shouldn't be included in the final prediction. As expected, the first three values of each tuple include information on both position and duration of every fixations.

The first part of the generator has the same architecture of a VGG16. Pre-trained weights are used to improve convergence and lower the total number of variables to train. The output of this first section is fed to a set of LSTM layers. Thanks to them the network is able to generate sequences of data and work with the temporal dimension. The generator output is fed to the discriminator, together with the original image on which the prediction was made. The discriminator must determine whether the prediction is realistic or not. No univocal ground truth is available, because every user looks at an image differently. The network has to understand how a realistic gaze path looks like, in order to output valid samples. To manage the two different inputs, the discriminator is divided into two different branches. The first branch deals with processing the path, while the second one deals with the image. The branch that processes the path consists of a single LSTM layer. The second branch is a pre-trained VGG16 model that extracts features from the input image. The two branches join together merging the outputs of each one. Finally, the data crosses additional LSTM layers that return a final value ranging between 0 and 1 and evaluating the quality of the generated path. If the output is higher than 0.5, then the path is realistic otherwise it's not.

Since it is a conditional GAN, the loss is slightly different from the normal one. The structure is very similar to that already seen in the models predicting saliency maps. One term is dedicated to minimize the adversarial loss while the other one aims at minimizing the content loss.

$$L_{cGAN}(G, D) = \mathbb{E}_{x,y}[logD(x, y)] + \mathbb{E}_{x,z}[log(1 - D(x, G(x, z)))] \quad [8.1]$$

$$L_{L^2}(G) = \mathbb{E}_{x,y,z}[\left\|y - G(x, z)\right\|^2] \quad [8.2]$$

$$L = L_{cGAN}(G, D) + L_{L^2}(G) \quad [8.3]$$

The formula of cGAN loss is represented in Equation [8.1]. In this case x indicates the image, z denotes the noise vector and y is the predicted path. Equation [8.2], instead, reports the content loss. It is equal to the L2-norm calculated between the network output and the ground truth. The combination of the two losses is reported in Equation [8.3]. The content loss is multiplied by a constant α in order to normalize its weight within the objective function. Thanks to this multi-target loss the training is more stable and the network provides better predictions.

8.1.2 PathGAN training procedure

PathGAN paper also describes the procedure used to train the network. A train/test split of 80%/20% is used to train and test the model. In order to facilitate the convergence, an unbalanced training approach has been used: the generator and the discriminator update their weights a different number of times during every training step. During the first 5 epochs only the generator is trained, so that it already has a solid knowledge to compete against the discriminator. The only loss used in this first phase is the content loss. Starting from the sixth epoch the discriminator training also starts. For each batch of images the generator is updated 8 times while the discriminator 16. This training strategy avoids the generator becoming too good. In this phase both content loss and adversarial loss are used. The content loss is multiplied by an α equal to 0.05, ensuring that the two terms have the right weight within the target function. These expedients used by the authors allow them to limit the mode collapse and reduce the chances of a diverging training. In the next sections, we expose how we have modified the network to fit our case study.

8.2 PathGAN problems in the GUI domain

The authors of PathGAN have partially released the source code of the architecture used. All the code concerning the network structure is present, while the training algorithm has not been published. During the re-implementation of the latter, we followed the details reported inside the paper. The first tests with the network did not give encouraging results in the GUI domain. We identified several problems that needed to be resolved to improve the performances. As we have already reported in the previous section the generator and discriminator weights are not updated with the same frequency. The choice to make more updates on the discriminator, rather than on the generator, did not lead to improvements in performance. Moreover, assigning a very low weight to the content loss ($\alpha = 0.05$) makes the discriminator very strong, compared to the generator. Training the generator for the first 5 epochs alone was still not sufficient to prevent this phenomenon. In addition, the number of values to predict complicates the problem too much. As mentioned in the previous subsection, to allow variable length path predictions the last element of each fixation tuple is an end of path probability. We noticed that the training did not manage this extra variable adequately, resulting in either very short or very long paths. Overall we experienced completely wrong predictions that, in the long run, during the training, led to mode collapse. In Figure 8.2 we can see two examples.

During mode collapse the generator tends to predict outputs that match the edge of the image, completely ignoring the original ground truth. The triggering cause of this phenomenon has been identified in the combination of several elements.

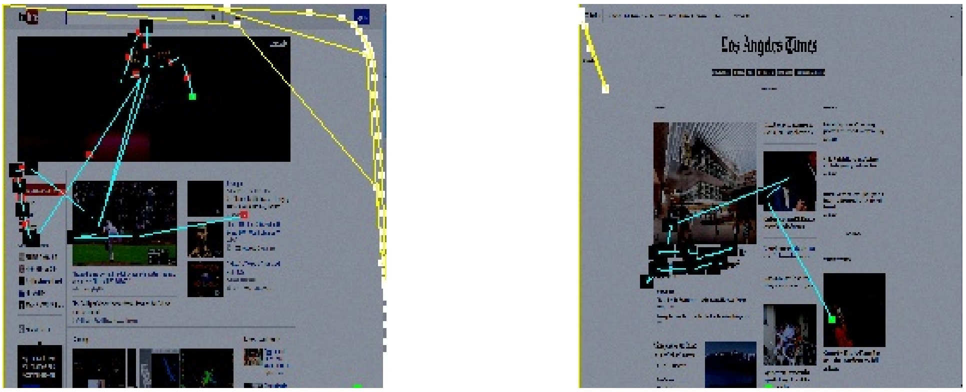


Fig. 8.2: Two examples of mode collapse

The discriminator becomes too strong, compared to the generator, which can no longer make realistic predictions. The lack of data does not help in this regard. The discriminator clearly overfits on the training data after several epochs. Changing the number of times the generator and discriminator weights are updated does not prevent this. An explanatory graph of this phenomenon is visible in Figure 8.3.

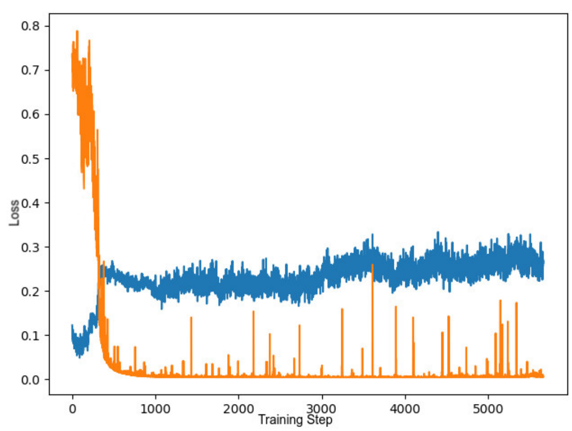


Fig. 8.3: Graph representing generator (blue) and discriminator (orange) loss after the discriminator overfits the training set

The generator (blue line) is very strong in the early stages of training because it has been trained alone for the first 5 epochs. In the first steps the discriminator loss starts to decrease gradually. It suddenly experiences a huge drop that matches with a degradation of the generator's loss score. From that point onward, the predictions start to be totally wrong. It is clear that the network needs some modifications in order to be applied to our domain. In the next section, we will report how we have adapted this architecture to work in our case study.

8.3 PathGAN enhancements

All the improvements we have introduced to the network try to mitigate the problems that have arisen by applying the reference architecture to the Graphical User Interface domain. Since we do not have a dataset as large as the one used by the authors, we have chosen to reduce the complexity of the problem. Instead of predicting a sequence of tuples of four elements (x-coordinate, y-coordinate, timestamp, end of path probability) we chose to remove a variable. Our goal was to predict paths of different lengths with only three variables, instead of four. We removed the end of path probability to force the network to predict paths of the same length. Since not all the paths of our dataset are of the same size, we opted to introduce dummy nodes on the arcs connecting two fixations. This solution makes sure that also our dataset has 63 fixation long paths. The dummy nodes have been added on the arcs by linear interpolation. Since, in reality, they do not actually correspond to a fixation, the timestamp is increased by a negligible constant. This allows us to distinguish which are the real fixations with respect to the fictitious ones introduced. After post processing the predictions, we are able to generate a sequence of real fixations. Each timestamp is transformed into a duration allowing us to better distinguish the real ones from the dummy ones. Predicted fixations with duration below the threshold are considered dummy and, therefore, removed from the path. This procedure preserves the generation of paths of variable length without the need of the end of path probability. After this modification, the results are improved, but further changes were necessary.

The next step was to introduce improvements in the way the network is trained. At the same time we also changed the weight assigned to content and adversarial loss. The only way to make the discriminator weaker is to update the generator weights more often.

At first we tried to keep the weights assigned to the various parts of the loss unchanged. Initially, we tried to tune the number of weight updates for every training step of both generator and discriminator. After several attempts we realized that we were just postponing the moment when the discriminator would overfit. Therefore, it has proved essential to also modify the weights of the various parts of the loss. We saw positive effects when we decreased the weight given to the adversarial loss within the objective function. A higher content loss weight prevented the discriminator from taking over. The quality of the samples generated increased dramatically, allowing the network training to be completed successfully. In conclusion, we have chosen to multiply the adversarial loss by a constant equal to 0.35, and the content loss by a constant equal to 1. Moreover, we found the right number of weight updates for every part of the network: we update 16 times the generator weights every step, limiting the discriminator to only 4 times.

We introduced other modifications aimed at avoiding overfitting. We took our cue from saliency prediction models and added noise on the images passed to the discriminator. Also in this case, the qualitative evaluations of the output, together with the loss trend, were fundamental; in fact, we could immediately detect any overfitting and mode collapse. Further attempts to improve the results were undertaken without success. For example, we tried to modify the network to receive a saliency map input. Both generator and discriminator should have benefited as there is a match between saliency map and path. We noticed that there were no tangible benefits; on the other side, the complexity of the architecture increased. We also tried to modify the path preprocessing. Instead of adding dummy nodes on the arcs, we tried to superimpose them on existing nodes. This should have brought more precision in predicting fixations. Again, we have not noticed any improvement. We believe that further attempts can be made using these techniques. Due to time constraints we did not have the opportunity to test all possibilities.

Finally, we have modified the network making it a simple WGAN. The training process was modified to respect the characteristics of a WGAN. The weights of the discriminator and the generator are updated with different frequencies. The discriminator is now updated more often because weight clipping has been introduced. The discriminator is updated 5 times while the generator is updated only once. We have also reset the weights of the various terms of the loss to their original values; in particular, we have set content loss at 0.05 and adversarial loss at 1. This change has brought interesting results which we will evaluate in the following sections.

8.4 Models evaluation

In this section, we are going to compare the two variants of PathGAN we have created. The models differ in loss and training procedure used. Both models predict fixed length paths made of sequences of three elements tuples, as previously introduced. The first model (hereafter, Normal) is a conditional GAN, while the second one (hereafter, WGAN) uses the structure of a conditional Wasserstein GAN. The two models have different weights assigned to the terms of the objective function. The Normal model uses a content loss weight equal to 1.0, while the WGAN model uses a content loss weight equal to 0.05. The former has an adversarial loss set to 0.2 while the latter has an adversarial loss set to 1.0. Each model advantages one term of the objective function over the other; this is due to the different combination of weight updates between generator and discriminator that requires a different parameter tuning.

Jarodzka's metrics calculate 5 similarity values for the following characteristics: vector shape, vector length (saccadic amplitude), vector position, vector direction and fixation. We will compare each model evaluating its performance both quantitatively and qualitatively. Keep in mind that each layout in our dataset has been observed by 11 or 12 different users. Because of this, there is no single ground truth for each image: every path collected in the dataset is considered as a ground truth to calculate the metrics. The prediction is compared with each ground truth of the image, and then the average performance is calculated. Finally, the average of all the images is calculated to evaluate the model as a whole. Table 8.1 shows the results of the two models.

The values express some important information. The shape of the vectors and the path length are predicted correctly by the models. The similarity is very high

	Vector	Shape	Direction	Length	Position	Duration
Normal	0.992	0.693	0.991	0.836	0.29
WGAN	**0.993**	**0.699**	**0.992**	**0.840**	**0.310**

Table 8.1: Model performance with no threshold on the duration of fixations

for all model variants considered. The position of the fixations is also quite good compared to ground truths. The direction similarity of the path, on the other hand, decreases considerably; however, it remains within an acceptable range. All models struggle to determine the duration of each fixation. The duration similarity is by far the lowest similarity metric, highlighting the poor performance in this regard. The similarity values reward WGAN as the best performing model. It achieves the best score in all of the 5 metrics. The Normal model performs also well, falling behind WGAN for just some decimal points in every metric. The metrics highlight that both models have the same strengths and weaknesses since they perform well and poorly in the same metrics.

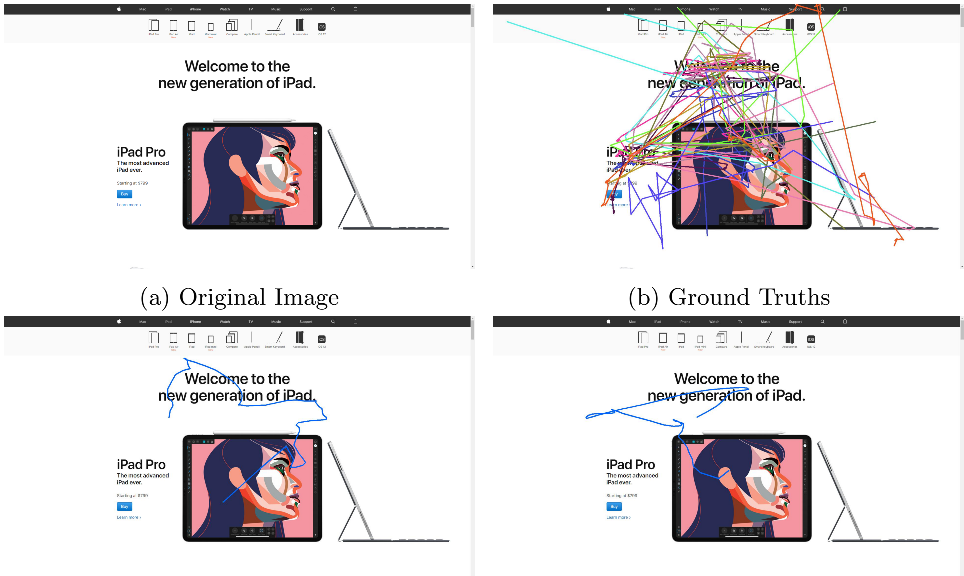
Before moving to a qualitative analysis of the results, we try setting a threshold on the duration of the fixations to improve the results. The goal is to eliminate the dummy fixations in the prediction generated by the network to return a path 63 fixations long. Table 8.2 shows the results.

	Vector Shape	Direction	Length	Position	Duration
Normal	0.992	**0.699**	0.991	0.838	0.308
WGAN	**0.993**	0.698	**0.992**	**0.840**	**0.326**

Table 8.2: Model performance with a threshold equal to 0.0027 on the duration of fixations

We have set a threshold equal to 0.0027; all fixations with a shorter duration are not considered as such. Since the durations are normalized between 0 and 1, and the predicted paths have a total duration of 4 seconds, we can calculate the corresponding threshold expressed in seconds. 0.0027 is around 10 milliseconds, one order of magnitude smaller than the average fixation duration [31]. We chose this threshold to avoid losing important fixations in the final prediction. This change has introduced a very slight improvement in results. The similarity values regarding duration remain modest, even if a very small improvement is visible. All the models obtain an increase in performance but too limited to affect the predictions positively. Once again WGAN is confirmed as the best model, even if Normal manages to shorten the distance in most of the metrics. Normal manages also to overtake WGAN on direction similarity for just a thousandth. Let's now compare the paths generated by the models from a qualitative perspective. Figure 8.4 contains an example of a prediction for each model considered.

The two predictions focus on the same areas of the image. The ground truths highlight that the most important areas on which the gaze falls are the two text sections and the central image. None of the two models can predict a path that touches all three portions of the image. The one that comes closest is WGAN model



(a) Original Image (b) Ground Truths

(c) Normal (d) WGAN

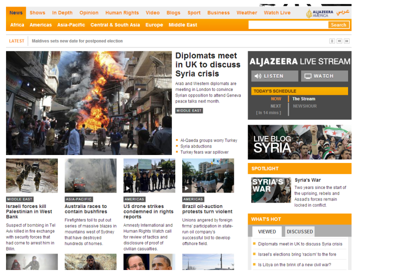
Fig. 8.4: PathGAN variants. Each picture represents a prediction

which follows in a more natural way the direction of the most important points of the layout. The metrics calculated on the individual images confirm this consideration. Both the Normal model and the WGAN model have very similar metrics in terms of length and vector shape. The direction similarity is better in WGAN with a value of 0.643 against the value of 0.608 of the other model. The initial movement to the left probably improves this similarity. In terms of position and duration the two models are very similar. In this specific image the better behavior of WGAN is confirmed by the metrics.

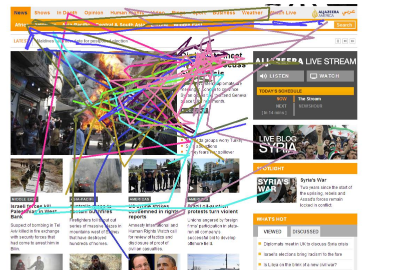
The next layout we analyze has more complex features. Unlike the previous image, the website is an online newspaper: a lot of information is present inside the page and the user's gaze is not easily predictable because many elements capture his attention. The figure of interest is reported in Figure 8.5.

In this scenario all models perform worse. The one that best encloses, although with many limitations, the set of ground truths is Normal. It is the one that best encapsulates the movement of the eyes towards the image, but leaving aside the focus on the central title, despite getting very close. Even the metrics detect this fact: Normal scores 0.74 in direction similarity while WGAN scores only 0.69. Again, we have very high values of length and shape for both models, with above-average performance also in terms of position. Duration similarity is still low in both models. The Normal model has a better overall performance in this layout.

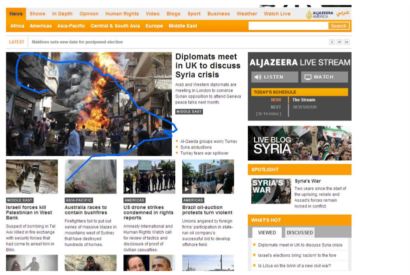
The last layout we are going to analyze is the one depicting a dashboard. The structure of the page is totally different from those already seen, with a very specific organization of information. In Figure 8.6 we can observe the predictions.



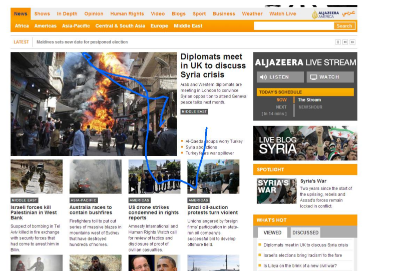
(a) Original Image



(b) Ground Truths

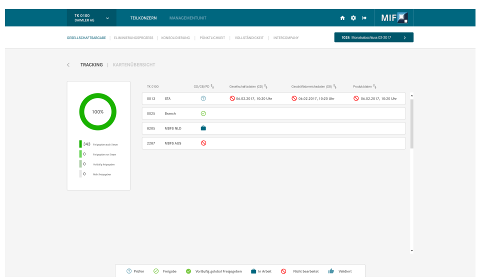


(c) Normal

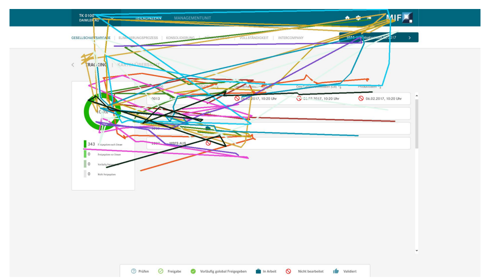


(d) WGAN

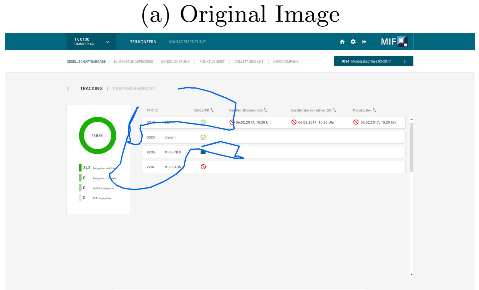
Fig. 8.5: PathGAN variants. Each picture represents a prediction



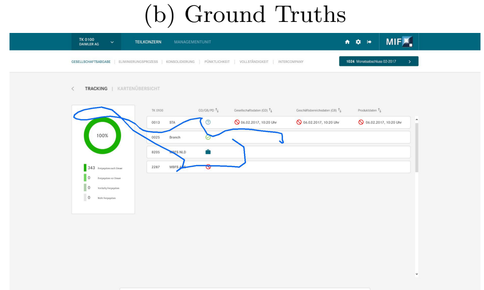
(a) Original Image



(b) Ground Truths



(c) Normal



(d) WGAN

Fig. 8.6: PathGAN variants. Each picture represents a prediction

We can immediately observe how WGAN behaves better in this case. The path collects most of the relevant areas of the page and is also more natural. The Normal model also outputs a realistic prediction, achieving a good result. The metrics calculated on this precise image confirm again our impression. The similarity of direction in WGAN model is higher than in the Normal model with a score of 0.74 against the score of 0.7. All the other metrics are also higher in value. The values of the position similarity are very good in that it reaches the value of 0.87. Also in this case the duration similarity does not record satisfactory values.

Overall PathGAN variants show the same strengths and weaknesses. In every analyzed image the models perform according to this model mean. The WGAN model behaves generally better than the Normal one, even if most of the time the competition is a close call. Wasserstein training improves final results almost in every aspect.

Our models may benefit from exploring new training procedures such as WGAN-GP. Weight clipping used in simple WGAN might have slowed down learning in the later training steps. We feel that the Normal model has reached its maximum performance with this amount of data. In fact, many attempts in tuning the parameters of the network have been performed. Further training can be attempted on the simple WGAN model doing a better tuning of the loss terms and the number of weight updates in each training step. Further research should be done to improve the duration estimation. A modification of the network can lead to better results.

9

Saliency and Gaze prediction Tool

In this chapter we introduce the tool we created. It collects all the functionalities of the neural networks we have exposed in the previous chapters, giving designers the possibility to analyze interfaces through a simple web interface.

9.1 Tool Overview

In order to make our research results usable, we have chosen to create a simple web application. The tool is a demo limited to basic functionalities. Designers can utilize the application to predict how the layout of a web page will be observed by users. A saliency map or a gaze path, representing the behavior of the average user, can be predicted.

This tool was created using the Django framework, as it easily interfaces with libraries written in Python. All the neural networks we have developed were written in this programming language; frameworks such as Keras and Tensorflow were employed. We chose a page structure simple and easy to use, where functionalities are easy to find. Using our network we also checked whether the focus was going on the correct parts of the layout. We wanted the user to start the uploading procedure as soon as possible. The saliency map corresponding to the main page of our tool is shown in Figure 9.4b.

(a) Normal layout

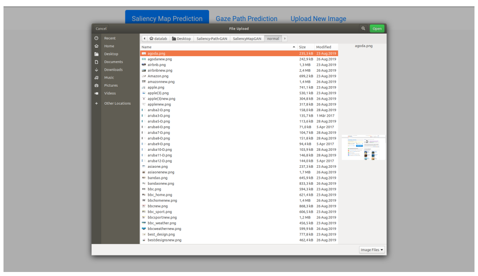
(b) Layout Heatmap

Fig. 9.1: Main page of our tool

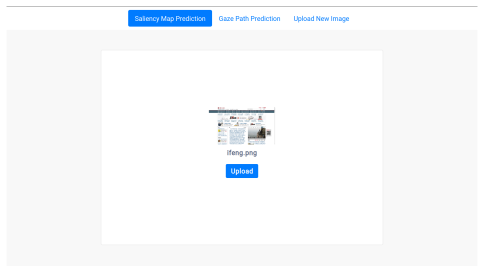
In the next subsections, we cover in more detail every functionality available in our web app. We focus on providing a step by step guide to use our tool.

9.1.1 Saliency Map Prediction

The first page appearing in our web application is dedicated to load layout images (Figure 9.4a). The user will have to select an image from his computer by clicking on the “Select a file” button. A selection window opens and the user can select the page he wants to analyze (Figure 9.2).



(a) Image selection



(b) Image preview

Fig. 9.2: Upload image layouts

In the top navbar, the user can choose whether to generate a saliency map or the complete path of the gaze. In this subsection, we analyze the saliency map generation functionality. We verify that the button “Saliency Map Prediction” is selected in the navbar. Once this is done, we notice that the screen shows a thumbnail of the uploaded image and the “Upload” button below it. Press the button to start the analysis. The total time taken to complete the prediction is variable depending on the computing power of the computer on which the tool is running. Four different predictions are generated, one for each available model variant.

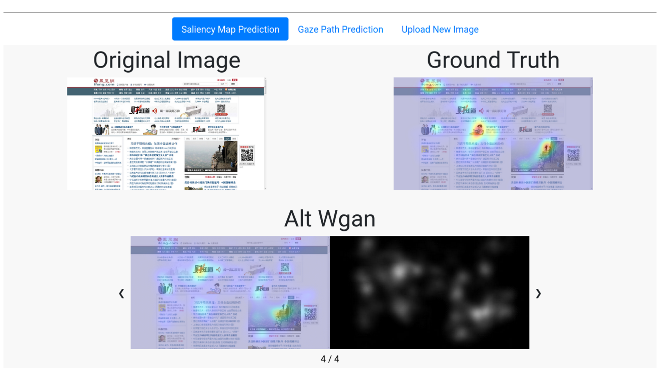


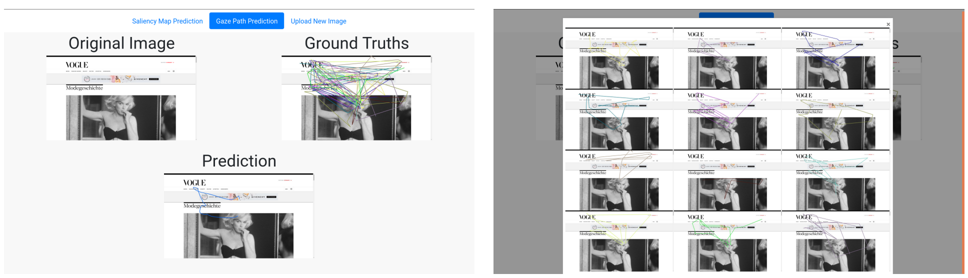
Fig. 9.3: Saliency map prediction screen

Once the analysis is complete, the results are displayed (Figure 9.3). At the top section, the original image and the corresponding heatmap of the ground truth are shown. At the bottom, a slideshow with the predictions of all models can be browsed. For each model the black and white saliency map and the corresponding heatmap, superimposed on the original image, are displayed. This facilitates to visualize which are the actual areas marked in the grayscale saliency map.

The user can perform another analysis by selecting the “Upload New Image” button; in this case, he will be redirected to the main screen where he can upload a new image.

9.1.2 Gaze Path prediction

As for the saliency prediction function, we need to load an image through the main screen of the web application to predict the gaze path (Figure 9.4a). The steps to perform this task are the same as those reported for the saliency map prediction (Figure 9.2). Make sure to select the “Gaze Path Prediction” button in order to use the correct functionality. The processing time is lower because only one model is used. The screen opened after having completed the analysis presents a summary of the results (Figure 9.4).



(a) Path prediction screen (b) Single ground truth preview popup

Fig. 9.4: Path prediction section

In the upper section, the original image is shown, together with the same image overlapped with the set of ground truths. It is also possible to click on the latter to view the individual paths in more detail through a popup. The lower section contains the prediction drawn on the original image. The overall layout is very similar to the one of the saliency map prediction section already analyzed.

10

Conclusion

In this thesis, we have proposed two approaches to predict the user's gaze behavior while looking at a web page. A first approach is based on the prediction of saliency maps to identify in advance which are the most attention-grabbing areas of a web page. The second approach tries to predict the complete path of the glance in the first four seconds of observation.

We explained how and why the results obtained help designers to speed up their design prototyping process. We provide the possibility to know in advance the behavior of the users who will view the newly designed layouts. Less meetings with the final users are necessary, saving a huge amount of time and money. Artificial Intelligence enables us to achieve our goals, particularly by exploiting generative models, such as GANs.

We have repeatedly pointed out that research in this field is very limited due to the lack of data. Saliency map prediction in the Graphical User Interface domain was not totally unexplored. The research work that inspired us is very recent and uses relatively new techniques. To the best of our knowledge, no research tried to create models for gaze prediction in the domain of Graphical User Interfaces. Gaze prediction is a very new research problem, that has been addressed only in the natural image domain. In addition to these considerations, Generative Adversarial Networks are still a young type of neural networks that research can still improve.

The results obtained in this thesis reach the state of the art in both prediction of saliency maps and gaze paths in the domain of web GUIs. The limited research in this field allowed us to explore new models and to take inspiration from what has already been achieved for the natural image domain.

Not all the results obtained are excellent, but they still look promising: better solutions can be built on top of ours. Our achievements must be considered as a starting point for research. Our work has always based on what scientific literature has produced so far, adapting solutions for other domains to ours.

Many further improvements can be designed to boost the results of this research. For instance, it could be possible: *(i)* to expand the amount of data available to train the models; *(ii)* to introduce new precautions to avoid model divergence and mode collapse; *(iii)* to develop new architectures from scratch specifically designed for the web page domain; *(iiii)* to investigate the possibility of introducing reinforcement learning for better performance. Many other small changes can be thought to help increase the quality of results. The world of Artificial Intelligence is in constant evolution to produce better performing models.

References

1. C. Shen and Q. Zhao. Webpage Saliency. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 33–46, Cham, 2014. Springer International Publishing.
2. Y. Li and Y. Zhang. Webpage Saliency Prediction with Two-stage Generative Adversarial Networks. *ArXiv*, abs/1805.11374, 2018.
3. J. Pan, C. Canton-Ferrer, K. McGuinness, N. E. O'Connor, J. Torres, E. Sayrol, and X. Giró. SalGAN: Visual Saliency Prediction with Generative Adversarial Networks. *ArXiv*, abs/1701.01081, 2017.
4. M. Assens, X. Giro i Nieto, K. McGuinness, and N. E. O'Connor. PathGAN: Visual Scanpath Prediction with Generative Adversarial Networks. 2018.
5. F. Rosenblatt. The Perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 1958.
6. D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *J. Physiol.*, pages 215–243, 1968.
7. Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, Nov 1998.
8. Sepp Hochreiter and Jürgen Schmidhuber. Long Short-term Memory. *Neural computation*, 9:1735–80, 12 1997.
9. A. Theodorou, R. H. Wortham, and J. J. Bryson. Designing and implementing transparency for real time inspection of autonomous robots. *Connection Science*, 29(3):230–241, 2017.
10. G. E. Hinton, T. J. Sejnowski, and D. H. Ackley. Boltzmann Machines: Constraint Satisfaction Networks that Learn. Technical Report CMS-CS-84-119, CMU Computer Science Department, may 1984.
11. G. E. Hinton. A Practical Guide to Training Restricted Boltzmann Machines. In Grégoire Montavon, Genevieve B. Orr, and Klaus-Robert Müller, editors, *Neural Networks: Tricks of the Trade (2nd ed.)*, volume 7700 of *Lecture Notes in Computer Science*, pages 599–619. Springer, 2012.
12. D. P. Kingma and M. Welling. Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16, 2014, Conference Track Proceedings*, 2014.
13. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative Adversarial Nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc., 2014.

14. A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from Simulated and Unsupervised Images through Adversarial Training. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2242–2251, 2016.
15. M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein GAN. *ArXiv*, abs/1701.07875, 2017.
16. I. Gulrajani, F. Ahmed, M. Arjovsky, V. Dumoulin, and A. Courville. Improved Training of Wasserstein GANs, 2017.
17. M. Mirza and S. Osindero. Conditional Generative Adversarial Nets. *ArXiv*, abs/1411.1784, 2014.
18. A. Borji and L. Itti. State-of-the-Art in Visual Attention Modeling. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):185–207, Jan 2013.
19. L. Itti and C. Koch. A saliency-based mechanism for overt and covert shifts of visual attention. Vision Res. 40, 1489-1506. *Vision research*, 40:1489–506, 02 2000.
20. J. Harel, C. Koch, and P. Perona. Graph-Based Visual Saliency. In *Proceedings of the 19th International Conference on Neural Information Processing Systems*, pages 545–552, Cambridge, MA, USA, 2006. MIT Press.
21. K. Simonyan, A. Vedaldi, and A. Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *CoRR*, abs/1312.6034, 2013.
22. N. Liu and J. Han. A Deep Spatial Contextual Long-Term Recurrent Convolutional Network for Saliency Detection. *IEEE Transactions on Image Processing*, 27:3264–3274, 2016.
23. S. Jia. EML-NET: An Expandable Multi-Layer NETwork for Saliency Prediction. *ArXiv*, abs/1805.01047, 2018.
24. M. Kummerer, T. S. A. Wallis, L. A. Gatys, and M. Bethge. Understanding Low- and High-Level Contributions to Fixation Prediction. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.
25. J. Chang, Y. Zhang, and Y. Wang. An Element Sensitive Saliency Model with Position Prior Learning for Web Pages. *ArXiv*, abs/1804.10361, 2018.
26. Z. Chen and W. Sun. Scanpath Prediction for Visual Attention using IOR-ROI LSTM. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, pages 642–648. International Joint Conferences on Artificial Intelligence Organization, 7 2018.
27. P. Xu, K. A. Ehinger, Y. Zhang, A. Finkelstein, S. R. Kulkarni, and J. Xiao. TurkerGaze: Crowdsourcing Saliency with Webcam based Eye Tracking. *ArXiv*, abs/1504.06755, 2015.
28. O. Le Meur and T. Baccino. Methods for comparing scanpaths and saliency maps: strengths and weaknesses. *Behavior Research Methods*, 45(1):251–266. 10.3758/s13428-012-0226-9.
29. Z. Bylinskii, T. Judd, A. Oliva, A. Torralba, and F. Durand. What do different evaluation metrics tell us about saliency models? *arXiv preprint arXiv:1604.03605*, 2016.
30. H. Jarodzka, K. Holmqvist, and M. Nyström. A vector-based, multidimensional scanpath similarity measure. pages 211–218, 01 2010.
31. M. Castelhano and K. Rayner. Eye movements during reading, visual search, and scene perception: An overview. 01 2008.
32. P. Ralph and Y. Wand. A proposal for a formal definition of the design concept. *Design Requirements Workshop (LNBIP 14)*, pages 103–136, 2009.

Acknowledgements

I would like to thank all those who made the development of this thesis possible.

A very special thanks to my supervisors Prof. Ursino and Eng. Porcino, that during these 6 months helped me to achieve these results. Their door was always open to assist me whenever I had the need. They've put me on the right track to overcome any difficulty, sometimes sacrificing their free time.

I would also like to thank the whole Datalab team who were able to give me valuable advice and made my stay in Germany very pleasant. They're all extraordinary people who always want to help and are passionate about what they do.

I would like to thank Daimler for providing me with the necessary tools to successfully complete this research work. Meeting the best experts inside this company, enriched my knowledge, making it possible to achieve these results.

Finally, I must express my very profound gratitude to my family, my girlfriend Diletta and all my friends for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis. This accomplishment would not have been possible without them.

Thank you.

Alessandro Scopelliti