UNIVERSITAT POLITÈCNICA DE VALÈNCIA

centralelille
ÉCOLE CENTRALE DE LILLE

TRABAJO DE FIN DE MASTER

STUDY AND IMPLEMENTATION OF A REAL-TIME ARTISTIC STYLE TRANSFER SYSTEM FOR AN IMMERSIVE ART EXPERIENCE

AUTHOR: Tanguy Jeanneau

TUTORS: Raúl Esteve Bosch

Roberto Palacio Paredes

ACADEMIC YEAR: 2018-19

MIARFID
Máster en Inteligencia Artificial, Reconocimiento de Formas e Imagen Digital

DSIC
DEPARTAMENTO DE SISTEMAS INFORMÁTICOS Y COMPUTACIÓN

ESCUELA TÉCNICA SUPERIOR INGENIEROS INDUSTRIALES VALENCIA

DISCLAIMER

This work has been conducted during my Erasmus semester at the *Universitat Politècnica de València* (UPV)

For administrative reasons, it was attached to *the Escuela Técnica Superior de Ingenieros Industrials* (ETSII) from the UPV, even if its content is linked *to Departamento de Sistemas Informáticos y Computación* (DSIIC) and to the classes I followed in the *Máster en Inteligencia Artificial, Reconocitiento de Formas e Imagen Digital* (MIARFID)

Finally, this work is not equivalent to a full Master Thesis at my home university *the Ecole Centrale de Lille* (ECL), it is therefore the first part of a longer work and will be carried at the ECL next year, following its requirements

ABSTRACT

Convolutional Neural Networks have been proven to be extremely powerful tool to solve image recognition tasks, outperforming previous models by far. Using the internal image representation of some models, it is possible to automatically perform style transfer, i.e extract the style of a target image and apply it to a content image.

The goal of this work is the creation of a screen-camera system that allows the user to see in real time the artistic style of a work of art. This system could be displayed in a museum allowing the visitors to immerse themselves in their favorite pieces of art.

This Real-time Video Style Transfer task will be solved by implementing the Network-optimization-based technique proposed by H. Huang *et al* [0], where a feed-forward transformation network is optimized to minimize a hybrid loss of content, style and temporal coherence.

This project is divided into 3 parts:

1) Study of the state of the art of style transfer applied to images and videos;

2) Description of the most relevant model: a feed-forward Image Transformation Network that can be optimized to perform style transfer for a given style, a module that calculates the temporal loss using dense optical flow, and a pre-trained fixed Loss Network that computes content and style losses;

3) Model implementation and prototype creation.

RESUMEN

En el último decenio, las Redes Neuronales a Convolución han mostrado sobresaliente capacidades de reconocimiento de patrones en imágenes. Esta capacidad se puede utilizar para realizar transferencias de estilo, i.e extraer el estilo de una imagen objetivo y aplicarlo a una imagen de contenido.

La meta de este trabajo es la creación de un sistema pantalla-cámara que le permite al usuario verse en tiempo real con el estilo artístico de una obra de arte. Este sistema podría estar en un museo y le permitir al visitante del museo sumergirse en sus obras de arte favoritas.

Para resolver esta tarea de Transferencia de Estilo para video en tiempo real, implementé un modelo propuesto por H. Huang *et al* [0] que consiste en una red convolucional residual *feed-forward* que se optimiza para minimizar una función perdida híbrida de contenido, estilo y coherencia temporal.

Este proyecto se divide en 3 partes:

1) Estudio del estado del arte de la transferencia de estilo aplicada a imágenes y videos.

2) Implementación del modelo más relevante: una red convolucional residual *feed-forward* que desempeña la transferencia de un estilo elegido, un módulo que computa la de coherencia temporal usando el flujo óptico denso y una red de perdida fijada) que calcula las pérdidas de estilo y de contenido

3) Implementación del modelo y creación de un prototipo pantalla-cámara.

INDEX

THESIS

0. Introduction

0.1. Context

In the past few years, Artificial Intelligence (A.I) has been gaining a lot of momentum, proving itself capable of solving numerous problems that were previously considered impossible and is now achieving state of the art results in a variety of tasks.

A.I can be used to optimize, predict, detect, generate based on data, and its adaptability make it usable in many situations. A.I is affecting almost every domain: Finance with algorithms of prediction on stocks, Insurance with risk assessment, Transportation with the rise of self-driving vehicles, industry with the optimization of processes and [smart maintenance], design with computer-generated design, computer vision, etc. But we haven't seen it all yet, and many more applications are yet to come.

One of the most fascinating aspects of A.I is the capacity of these algorithms to grasp complex patterns and, for some of them, to generate content following these patterns.

Art, as a domain, is especially being challenged by the new possibilities given by these algorithms. New York's Christies saw in October 2018 the first auction of an AI-generated piece of art, that was sold for more than 400,000€ - 10 times more than its predicted sale value! Art and creativity used to be the privilege of humans, but this is redefining the boundaries between human and machine.

What is the value of an art piece without author? Can this be considered as a form of creativity and art? Where is the value of an art piece that was automatically generated? These questions and many more remain open.

Fig. 1: "Edmond de Bellamy", by Obvious. First ever AI-generated piece of art to be sold at auction.

0.2 Definitions

Before entering further in the subject, it is necessary to define the terms we will be using.

First, let's address the elephant in the room: What is Artificial Intelligence?

In March 2019, a study from London venture capital firm MMC found out that on the 2,830 European *AI startups* studied, a whopping 40% of them did not actually used Artificial Intelligence [1]. This in one of the many signals that shows the blurriness of the definition of A.I. Especially since it became *trendy*, anything more or less related to computer science can end up being called A.I. It became a *buzzword* often misused in the media, or by the companies themselves to make a product look more appealing or raise more funds.

Moreover, there is a deeper reason for the inconsistency of the definition of *Artificial Intelligence*, which is the range of definitions of *Intelligence*. How can we define Artificial Intelligence if there isn't one universal definition for Intelligence?

0.2.1 Intelligence

Some definitions of human intelligence revolve around its capacity of rational reasoning, as in the Cambridge Dictionary: "Intelligence: the ability to learn, understand, and make judgments or have opinion that are based on reason" [3]

Other argues that there is no such thing as one human intelligence, but several *intelligences*. This theory was built by Howard Gardner in the 80s, mainly developed in his book "Frames of Mind: The Theory of Multiple Intelligences" [2]. Among the 7 intelligences he defines in this book we can name Logical-Mathematic intelligence, Verbal-Linguistic intelligence, Intrapersonal intelligence, or Interpersonal intelligence. While the first two can be included in Cambridge Dictionary's definition, the two last describe emotional processes and implies a capacity of self-consciousness. Like Daniel Goleman's controversial Emotional Intelligence [4], these intelligences go beyond rational reasoning and revolve around emotions and feelings.

The link between computers and emotions is quite a controversial subject, and we will here stick to Cambridge Dictionary's definition and focus on the rational definition of intelligence.

0.2.2 Artificial Intelligence

Artificial Intelligence is the kind of phrase that spark curiosity and fear, amp up fantasies and almost have a mythological aspect. Let's adopt here a more rational and pragmatical viewpoint.

The history of A.I is tightly linked to the history of computer, the latter being the support of the former.

Ada Lovelace (1815-1852) is considered by the historians as the first programmer [7] with her work on Baggage's Machine, but the first modern computer (ENIAC) only appears in 1946. Alan Turing (1912-1954) is considered to be the father of computer science and developed in 1937 the concept of Turing Machine and in 1950 the Turing Test, among other major contributions.

Fig 2. Left: Photography of Alan Turing. Right: Painting of Ada Lovelace.

The development of computer science allowed the creation of more and more complex algorithms. During the 70s and the 80s, the focus was on expert systems: systems where human knowledge were embedded as rules to allow human-like decision-making. One of the most important expert system in MYCIN, developed at Stanford University in the 70s, that was able to achieve expert level of blood infection identification [8]

An "AI Winter", i.e a period of drastic loss of interest on the fied of AI, followed from the late 80s to the early 90s

Then, one of the major milestones in the history of A.I is the victory of IBM's computer Deep Blue against Garry Kasparov, n.1 rated world chess champion at the time, in 1997.

From the late 90s to now, A.I gained back momentum and is now more popular than ever. We are now in an AI era dominated by Machine Learning, the field of study of algorithms

that can learn patterns from data. This “revolution” is powered by the ever-growing quantities of data and processing power.

We’ll chose here the definition of A.I of Cambridge Dictionary [6]: “the use of computer programs that have some of the qualities of the human mind, such as the ability to understand language, recognize pictures, and learn from experience.”

This definition rules out the notions of feelings and consciousness often linked to the notion of intelligence to focus on the imitation of human-like rational reasoning.

Another point to demystify is the dissociation between weak AI and strong AI. The former describes an algorithm that can reach super-human level at one predefined task (say, translate sentences between languages). The latter, also called Artificial General Intelligence, is the theoretical concept of an A.I that could do everything that a human does including its capacity of self-consciousness for instance. While strong AI only exists in science-fiction, all A.I are actually weak ones. The term A.I will only refer to weak A.I here.

To sum it up in Britannica’s terms [5],

> “Psychologists generally do not characterize human intelligence by just one trait but by the combination of many diverse abilities. Research in AI has focused chiefly on the following components of intelligence: learning, reasoning, problem solving, perception, and using language”

0.2.3 Machine Learning

Nowadays, the main category of A.I is Machine Learning: “a type of artificial intelligence in which computers use huge amounts of data to learn how to do tasks rather than being programmed to do them » (Oxford Dictionary [9])

The knowledge doesn’t need to be encoded by human experts, but the algorithm will explore possibilities and find out which set of parameters gives the best output.

One sub-category of Machine Learning is Deep Learning: Machine Learning using deep neural networks. Deep learning algorithms are capable of learning more complex patterns, generally producing better results, but with the cost of a worse interpretability. This is why it is often pictured as an advanced black box.

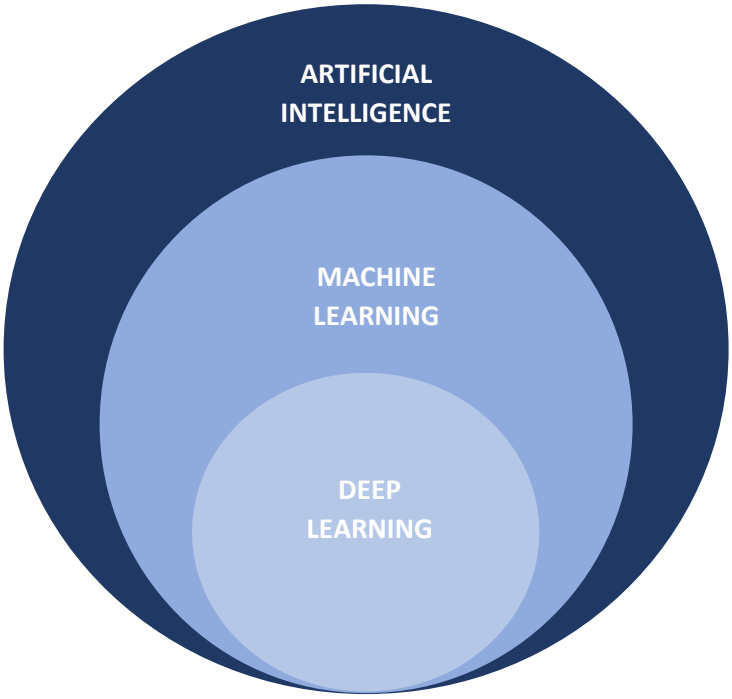


Fig 3. Venn Diagram showing the semantic inclusion of the phrases Artificial Intelligence, Machine Learning and Deep Learning.

0.2.4 Computer Vision

Computer Vision is a branch of computer science whose goal is to enable computers to process visual data (i.e images, video) to perform some of the tasks that the human visual system does and outperform it. These tasks include pattern recognition, object detection, trajectory and velocity estimation, image understanding, building a 3D representation from 2D images, etc.

I. Computer vision and Machine Learning

1.1 A brief history of Computer Vision and Convolutional Neural Networks

To understand why Machine Learning is so widely used in image processing, let's go back a few years in the past to understand how it changed the computer vision landscape.

1.1.1 From 1957 to 1989

In 1957, Russel Kirsch and his team were wondering "what would happen if computers could see the world the way we see it?". This question led them to create the first ever digital image, a 176x176 pixel scan of Kirsch's baby's picture. This is often considered to be the beginning of Computer Vision as a science field. However, it wasn't until 1975 that the first digital camera was created by Steven Sasson, a Kodak engineer.

Fig 4. Left: First ever digitized image, R. Kirsch, 1957. Right: First ever digital photo displayed on a monitor, next to the analogous image that served as model.

In 1960, Larry Roberts started studying the extraction of 3D information from 2D views at the MIT

In 1979, K. Fukushima proposed the Neocognitron, an early version of CNN. This was inspired by Nobel prizes D. Hubel & T. Wiesel's work on the visual system, as they discovered two types of cells in the visual primary cortex (namely simple cell and complex cell) responding to different kind of stimuli. The Neocognitron has been used for pattern recognition and Optical Character Recognition (OCR)

In 1982, David Marr published another influential paper, "Vision: A computational investigation into the human representation and processing of visual information". [17]

1.1.2 From 1989 to today

In 1989, Yann LeCun published "Backpropagation applied to handwritten zip code recognition" [20] where he used back-propagation to find the optimal parameters of a neural network to perform OCR on digits for ZIP code recognition.

Later on 1998, Yann LeCun proposed LeNet5, a novel 7-level Convolutional Neural Network in "Gradient-based learning applied to document recognition". [19]

CNN started gaining more momentum after AlexNet[14], a CNN proposed by Alex Krizhevsky in 2012, vastly outperformed its concurrence during the ImageNet Large Scale Visual Recognition Competition (ILSVRC). During this international competition, the participants must create object recognition software that are trained and evaluated on ImageNet, a large database of more than 14,000,000 images spread across more than 20,000 categories. AlexNet scored a top-5 error of 15.3% at ILSVRC 2012, which was more than 10 points lower than the second best of this year. This enormous increase of performances contributed to attract public attention on AI and computer vision.

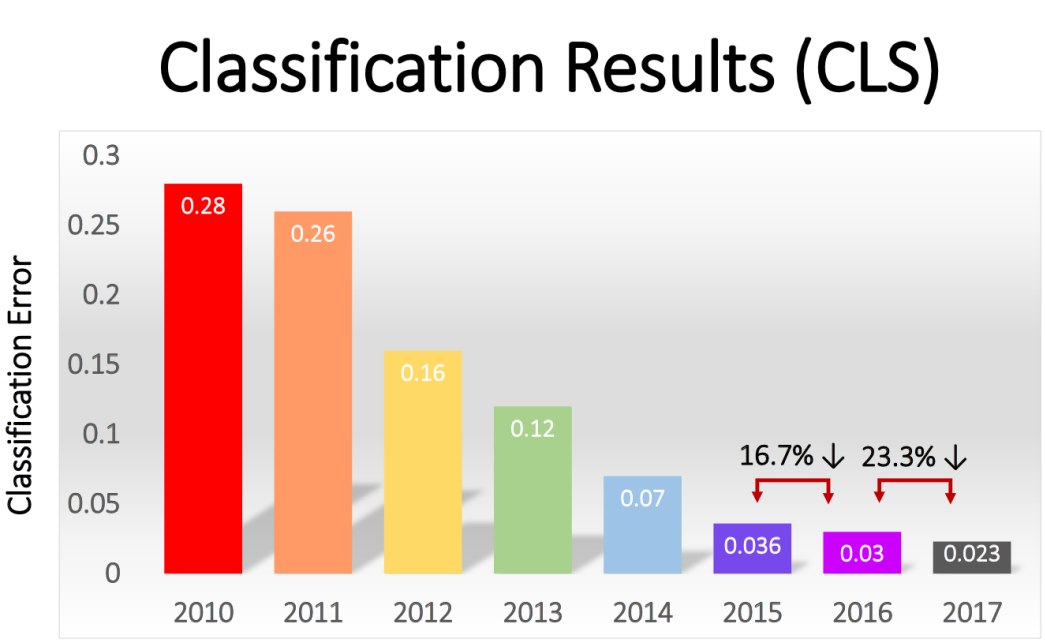


Fig 5. Best (lower) Classification Error at the ILSRVC competition, by year.

Since then many architectures have been proposed, some of them using specific layers or techniques. We can name for instance VGG that was runner-up for ILSRVC 14, often used for its simplicity and its performance but quite heavy (138 million parameters). GoogLeNet (a.k.a Inception V1) won ILSRCV 14 and introduced the inception module, that reduced the total number of parameter to about 4 millions. ResNet won ILSRCV 15 and introduced the notion of skip connection that allowed a better propagation of the gradients and networks to go deeper and improve performances. A visual comparison of these architectures is included after the introduction of the building blocks of CNNs

Through time, CNN are getting more and more complex and deeper, which is made possible by the improvements in processing power and GPU-based computing.

Let's get more into the details and see how Neural Networks works.

1.2 (Regular) Neural Networks

Nature is a great source of inspiration in science, especially living beings since the millions of years of evolution life has been through allowed the design of performant solutions. Many great discoveries were made trying to replicate natural phenomenon, and biomimetics is for instance at the origin of Leonardo Da Vinci's flying machine (the first flying device designed), hydrophobic surfaces, Velcro Tape (inspired by bur fruit), etc.

The human brain being considered as the most advanced processing unit in nature, its inner working inspired researchers when trying to artificially replicate intelligence. In 1943, Warren McCulloch and Walter Pitts tried to modelize a brain neuron, and their model got refined during the next decades to produce today the artificial neural network model.

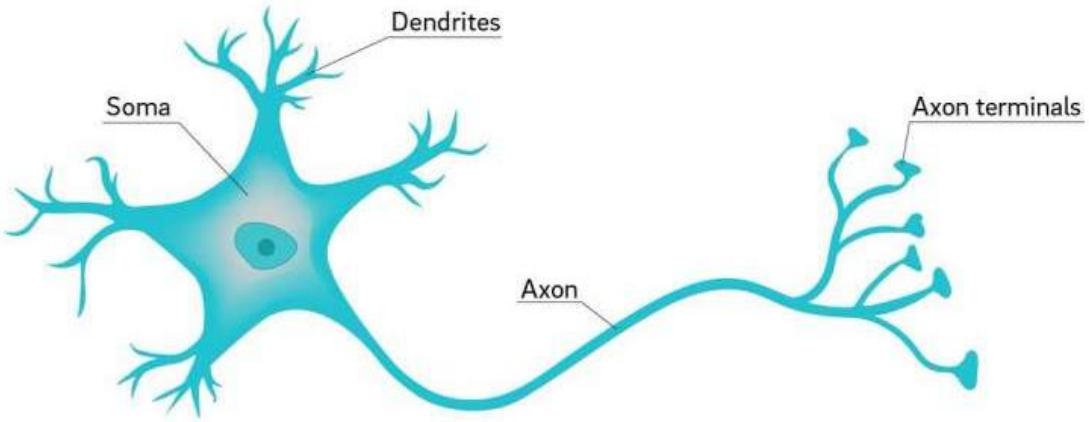


Fig. 6: Simplified representation of a biological neuron

Neurons works as following: Dendrites that convey information from other neurons as an electrical signal. They converge into a cell body (soma) that gather the electrical information and, if the combination of the inputs signals is strong enough, release an output signal to the axon that will convey the information to other neurons.

The model of an artificial neuron is the following:

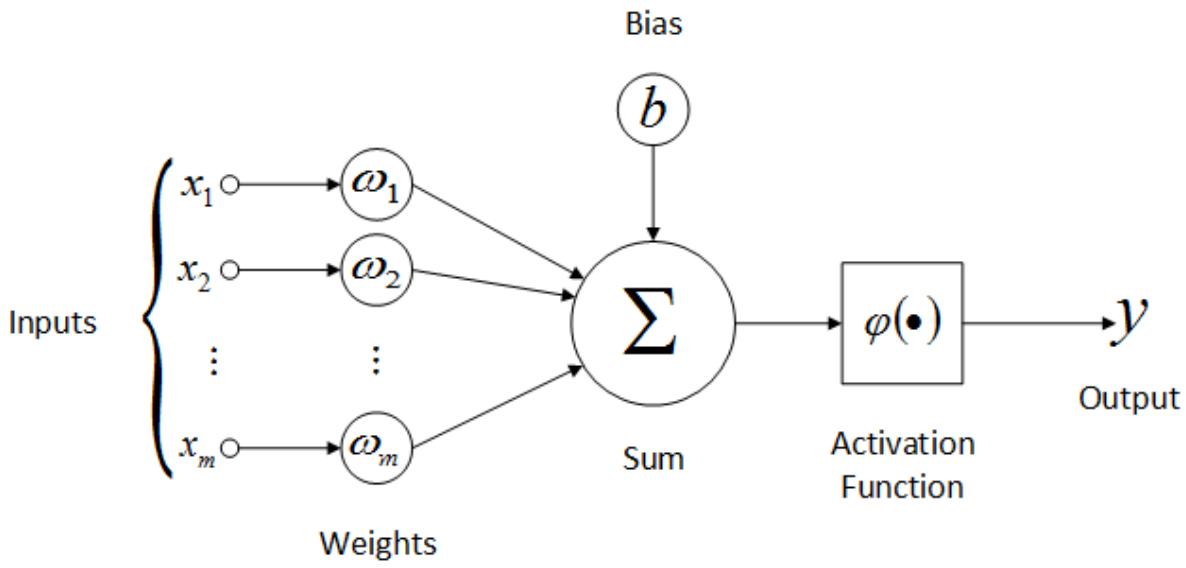


Fig 7. Representation of an artificial neuron

The electrical signal is represented as a real number in [0, 1], allowing to represent not only on and off signal but also in between states. The cell processes step is represented by a weighted sum, whose result is passed through an activation function to bring back the signal in [0, 1].

The output of a neuron is:

$$o = \varphi\left(\sum_{i=1}^{N}(w_i * x_i) + b\right) = \varphi(\mathrm{WX} + \mathrm{b})$$

With N the number of inputs, X=(x1-xN) the input vector, W=(w1-wN) the weights vector for this neuron, b the bias of the neuron, and σ the activation function.

The sigmoid function is often used as the activation function, which is defined as following:

$$\varphi(\mathrm{x}) = \frac{1}{1 + e^{-x}}$$

Then we build a layer of neurons in which every of them takes the same inputs and have they own weights. A layer has as many output dimensions as it has neurons.

$$O = \varphi(A^T\mathrm{X} + \mathrm{B})$$

With A = ($w_{i,j}$) the matrices of every weight of every neuron of the layer, B the matrix of biases, X=(x1-xN), and by extending φ to matrices

Finally, a Network can be built by having multiple layers and plugging the outputs of one to the inputs of the next one. The result is a Neural Network with one input layer X = (x1 – xN), one output layer and one or several hidden layers (the layers that are not input nor output)

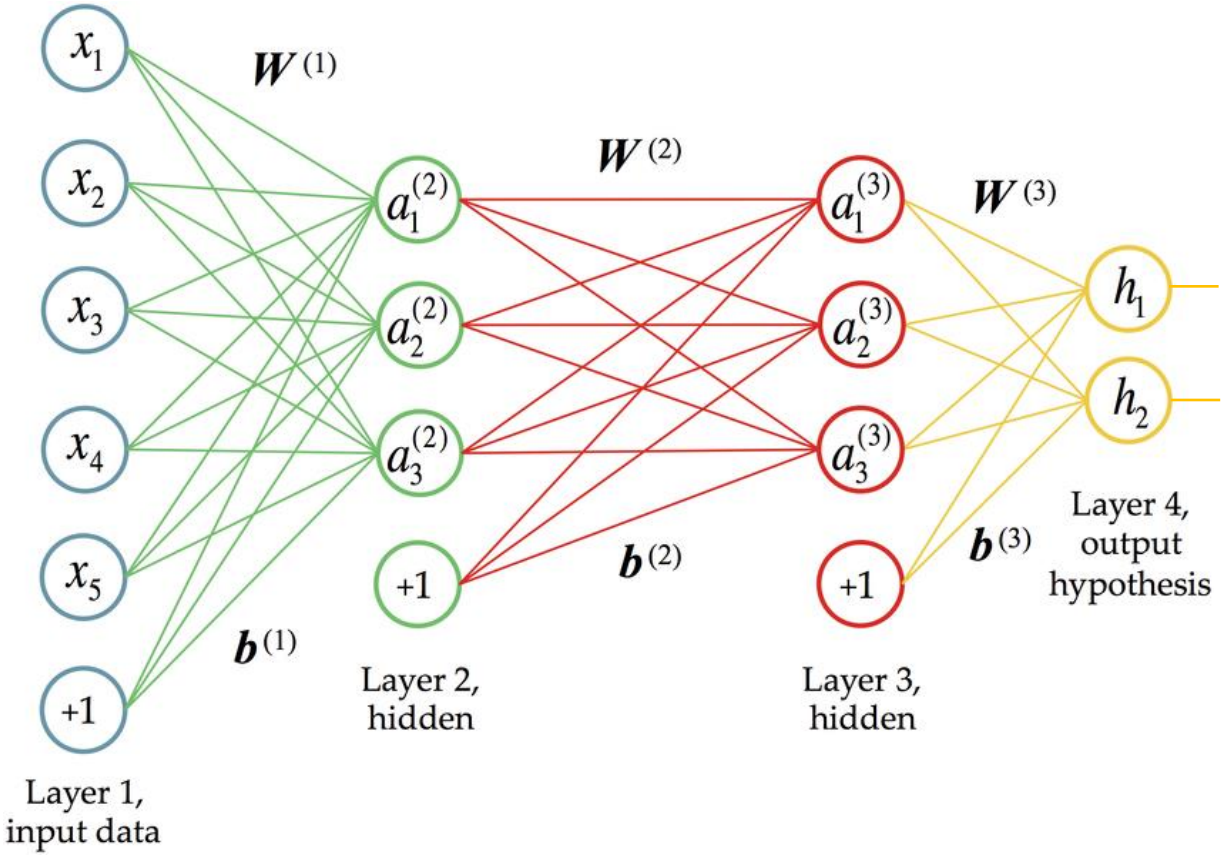


Fig. 8: An Artificial Neural Network with two hidden layers

The efficiency of neural networks has been theorized and proved under certain circumstances. For instance, in 1989 G. Cybenko proved that a neural network with a single hidden layer containing a finite number of neurons and using sigmoid as activation function can approximate continuous functions on compact subsets of R^n.

However, this theorical result does not provide the actual number of neurons nor the coefficients of the network to reach the needed precision.

In order to obtain the most optimized parameters (weights and biases), we need a cost function that would evaluate how well is the network performing, and an optimization algorithm that would iteratively modify the weights to minimize the cost function. Ideally the cost function should be convex, so that the optimization algorithm can converge to only one minimum.

One common algorithm for optimizing the network's parameters is the Gradient Descent. An input (or a batch of inputs) is passed through the network, the loss function is calculated, then the gradients of the loss function with respect to each parameter is computed. The error is then backpropagated to modify each parameter.

However, Regular Neural Networks are not well suited for image processing since they are shift-dependent, could only take a fixed-size input and are computationally expensive.

1.3 Convolutional Neural Networks

1.3.1 Convolution operation

In image processing, convolutions are one of the most important operations. A convolution is defined by a kernel (or filter), a matrix of smaller dimension than the input image. To obtain the output matrix, sub-matrices of the size of the kernel are extracted from the input image, and the element-wise product of these two matrices is summed. The kernel is shifted throughout the input image and this operation is repeated to obtain a complete matrix.

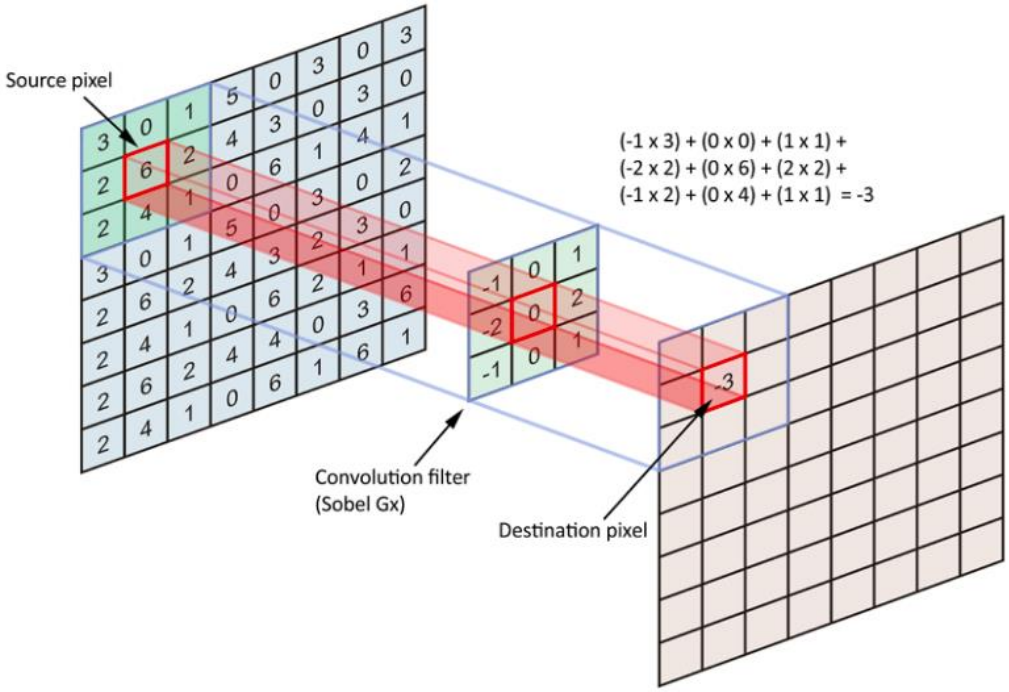


Fig. 9: A Convolution with Sobel filter along x axis

Convolution filter can be used for variety of tasks depending on the kernel's values: Edge detection (Sobel filters), box blur (normalized Ones matrix), gaussian blur, sharpen (Laplacian filter), etc.

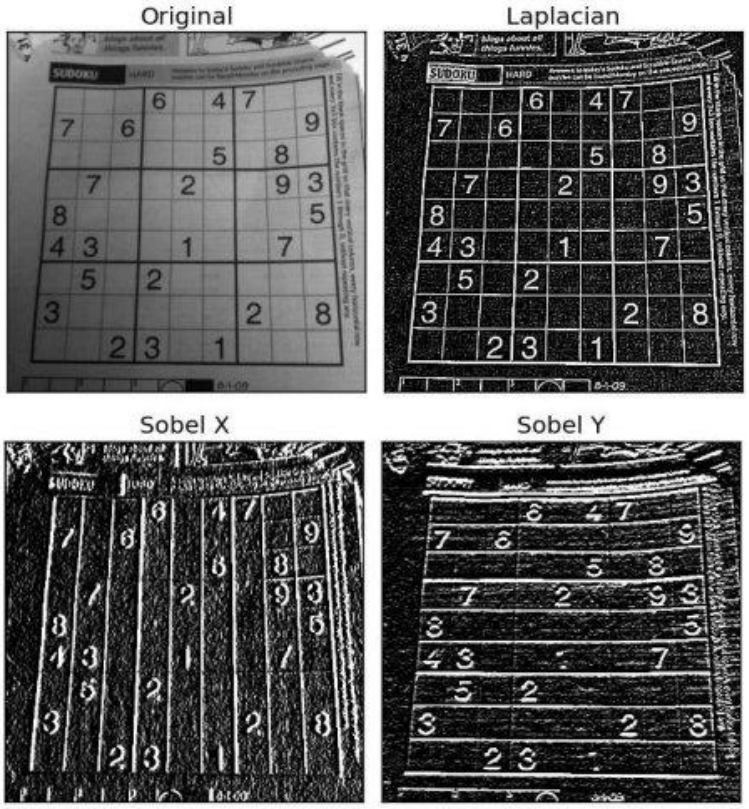


Fig. 10: Image before and after being processed by various convolution filters, whose kernels are detailed below.

$$Laplacian\ filter: \begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix}$$

$$Sobel\ x: \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$$

$$Sobel\ y: \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

1.3.2 Convolution Layer

The main element of Convolutional Neural Networks is the convolution layer. A convolutional layer is a group of N convolutions whose kernel's weights are the parameters. It is also important to note that since the input objects are 3-dimensional (images have a given height, width and a depth of three for colored images), the kernels have 3 dimensions.

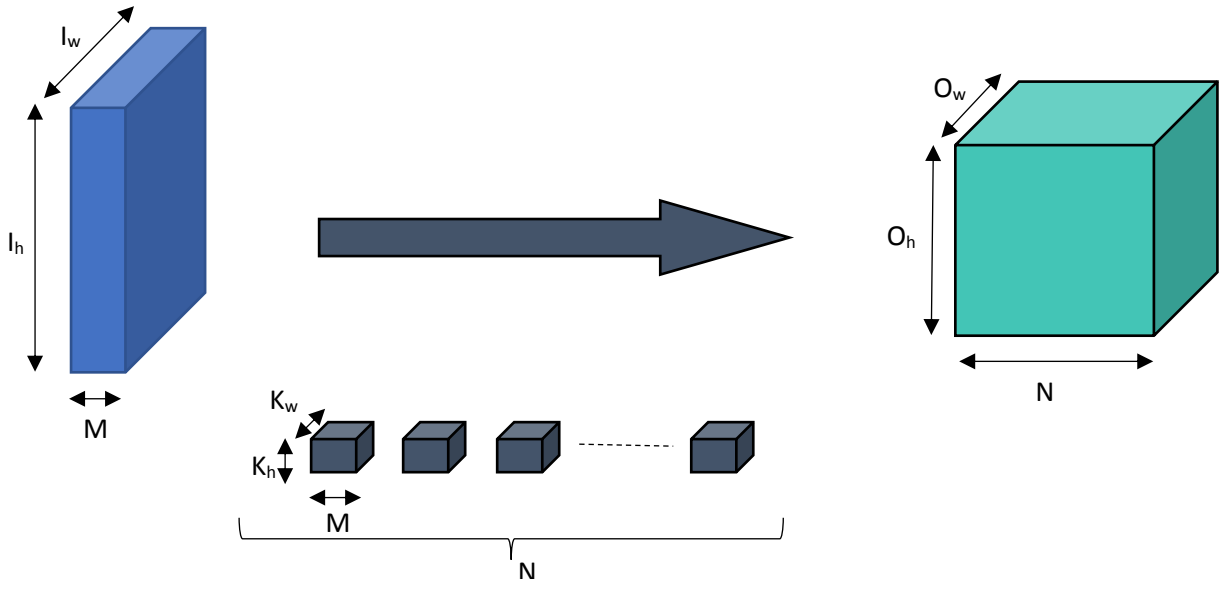


Fig 11. A Convolutional layer and its parameters

With M the input depth, N the number of kernels of the layer and therefore the output depth, I_h and I_w the height and width of the input, O_h and O_w those of the output and K_h and K_w those of the kernels.

Also, during the training phase not only one input is given but a batch of inputs, which adds one dimension to the inputs and outputs of the layer.

Several important factor can have an impact on the output size O_h (resp. O_w):

- The input size I_h (resp. I_w)
- The kernel size K_h (resp. K_w)
- The stride (distance between two consecutive frames of the kernel) along h s_h (resp. s_w along w)
- The zero padding (number of zeros added at the beginning and at the end of the input) along h p_h (resp. p_w along w)

Stride and padding modify the way the convolution kernel is applied to the input. By adding stride, the kernel won't slide across all the input element by element, but the kernel will be applied every s_h (resp. s_w) element. This will reduce the output size. By adding padding, p_h (resp. p_w) zeros are concatenated along the respective axis and included in the input. This will augment the output size.

The general formula relationship between the output size and the given parameters is:

$$O = \left\lfloor \frac{I + k - 2p}{s} \right\rfloor + 1$$

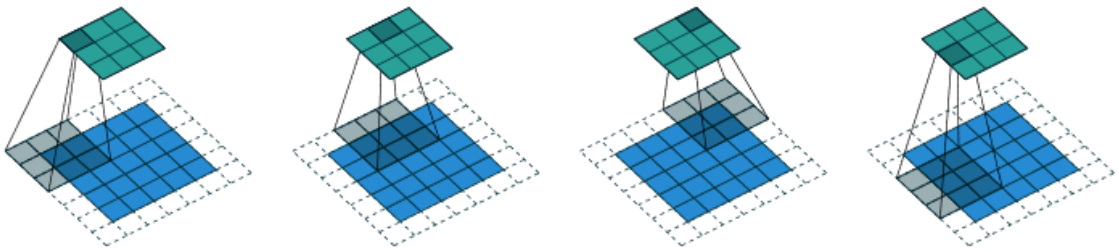


Fig 12: Convolution of a 3x3 kernel (gray) over a 5x5 input (blue) with a 1x1 zero padding, a 2x2 strides to produce a 3x3 output (green). More visualizations of this kind are available in [22]

1.3.3 Activation functions

Like in regular Neural Networks, the output of a convolutional layer is passed through an activation function. In the case of CNN, sigmoid function I rarely used, the Rectified Linear Unit (or ReLU) is more frequent, along with the hyperbolical tangent function. This function is defined as following:

$$ReLU(x) = \begin{cases} x \; if \; x \geq 0 \\ 0 \; if \; x < 0 \end{cases}$$

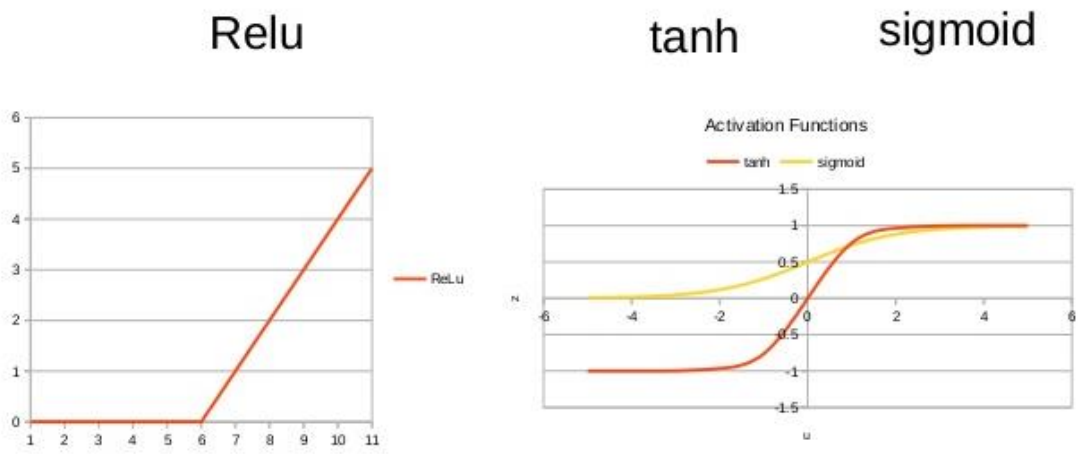


Fig. 13: Activation functions. Left: ReLU. Right: Tanh and Sigmoid.

1.3.4 Pooling layers

The goal of a pooling layer is to reduce the size of the input to reduce the computational cost. Pooling layers operate by segmenting the input in squares of a pre-defined size and returning one value per segment.

The two common pooling layers are max pooling and average pooling. As their names suggests, the first returns the maximum value of each segmented region, and the second returns its average.

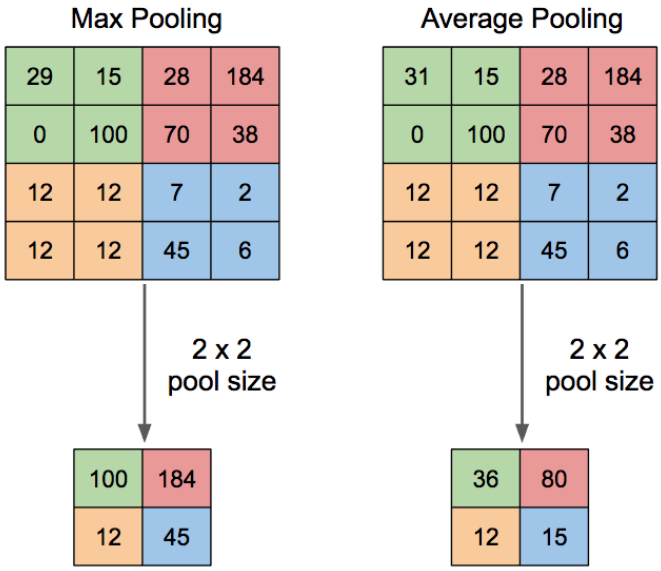


Fig. 14: Max pooling and Average pooling layers. [23]

1.3.5 Fully connected layer

This layer is just a regular neural network layer. If the input is a volume, then it is flattened and this vector is passed through the neural network layer. It is often found at the end of the networks to bridge the gap between the features extracted by the convolution and pooling layers and the targets (categories to recognize for instance)

1.3.6 Feature extraction capacities

Convolutional Neural Networks are very performant tools when it comes to object detection and patter recognition and state-of-the-art performance on these problems are mainly achieved by CNNs. They build over the training iterations feature identification capabilities, each layer building on the previous one to grasp more complex patterns.

As a picture is worth a thousand words, here is the beautiful feature visualization made by the Google Brain Team [12]. The interactive version available at https://distill.pub/2017/feature-visualization/ is worth giving a look. They wanted to see which kind of pattern the different layers of GoogLeNet were the most sensitive to. To do so, they optimized the input image to maximize the activation of various layers using gradient descent on the pixels space. As we can see here, the deeper the layer, the more complex were the patterns identified.

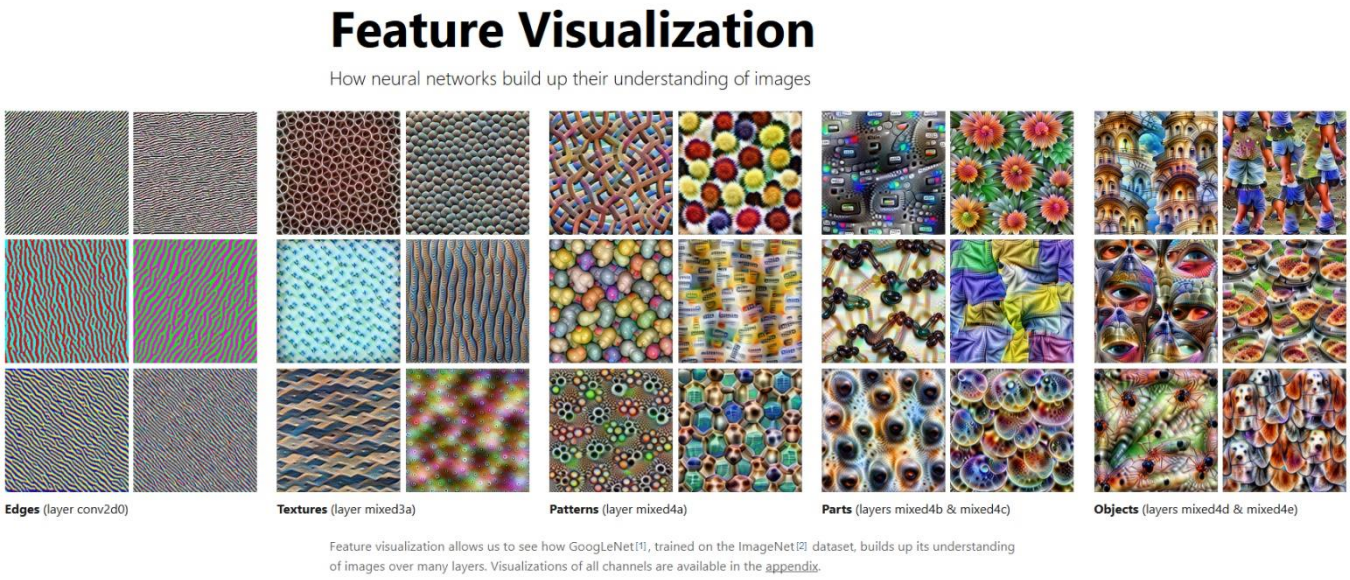


Fig. 15: Visualization of the feature representation in GoogLeNet

1.3.7 Famous architectures

As explained in 1.1 A brief history of computer science and convolutional neural networks, one of the breakthroughs that allowed CNN to surpass previous models were the skyrocketing performances of AlexNet in the ILSRVC 12 competition. After this model was created, numerous novel architectures were proposed the following years and we will review here some of the most performant.

Among other, the Residual block was introduced by Kaiming He *et al* in 2015[47] in ResNet, the winner of ILSRVC 15.

Counter intuitively, a deeper network won't necessarily give better results than a shallower one. This in partially due to the fact that when overpassing a certain depth, the gradients will tend to vanish, and the network won't be able to properly learn. The proposed architecture with *skip connections* aims at fixing this issue.

The ResNet that won ILSRVC 15 had 152 hidden layers, compared to 22 for GoogLeNet who won the previous year's competition.

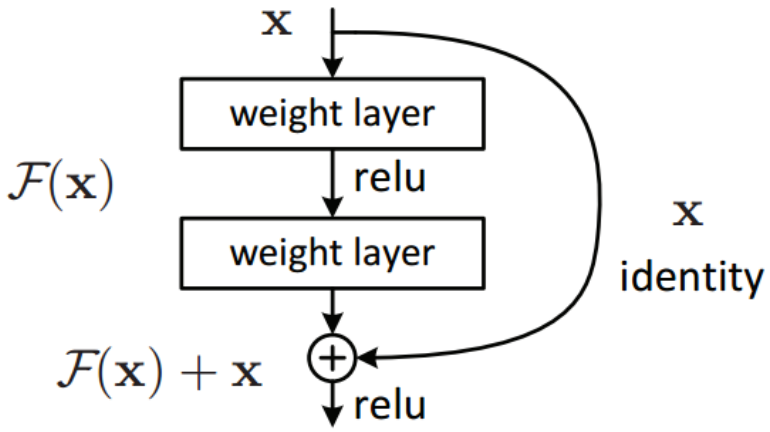


Fig. 16: A Residual block. The first weight layer block is often Convolution – Batch Normalization – ReLU, while the second is Convolution – Batch Normalization.

At ILSRVC 14, two models were especially remembered: The runner-up VGG for its simplicity and its feature representation that makes him the default choice for feature extraction nowadays; and the winner GoogLeNet for its performances and the Inception block it introduced.

At this point, it seems important to notice that, even though the first neural networks were biomimetics, they got further away from the initial intuitions while improving and are more to be considered as mathematical and informatics models than attempts to recreate or simulate the human brain. For instance, back-propagation is a key process in neural network's training, but there is no such mechanism in nature.

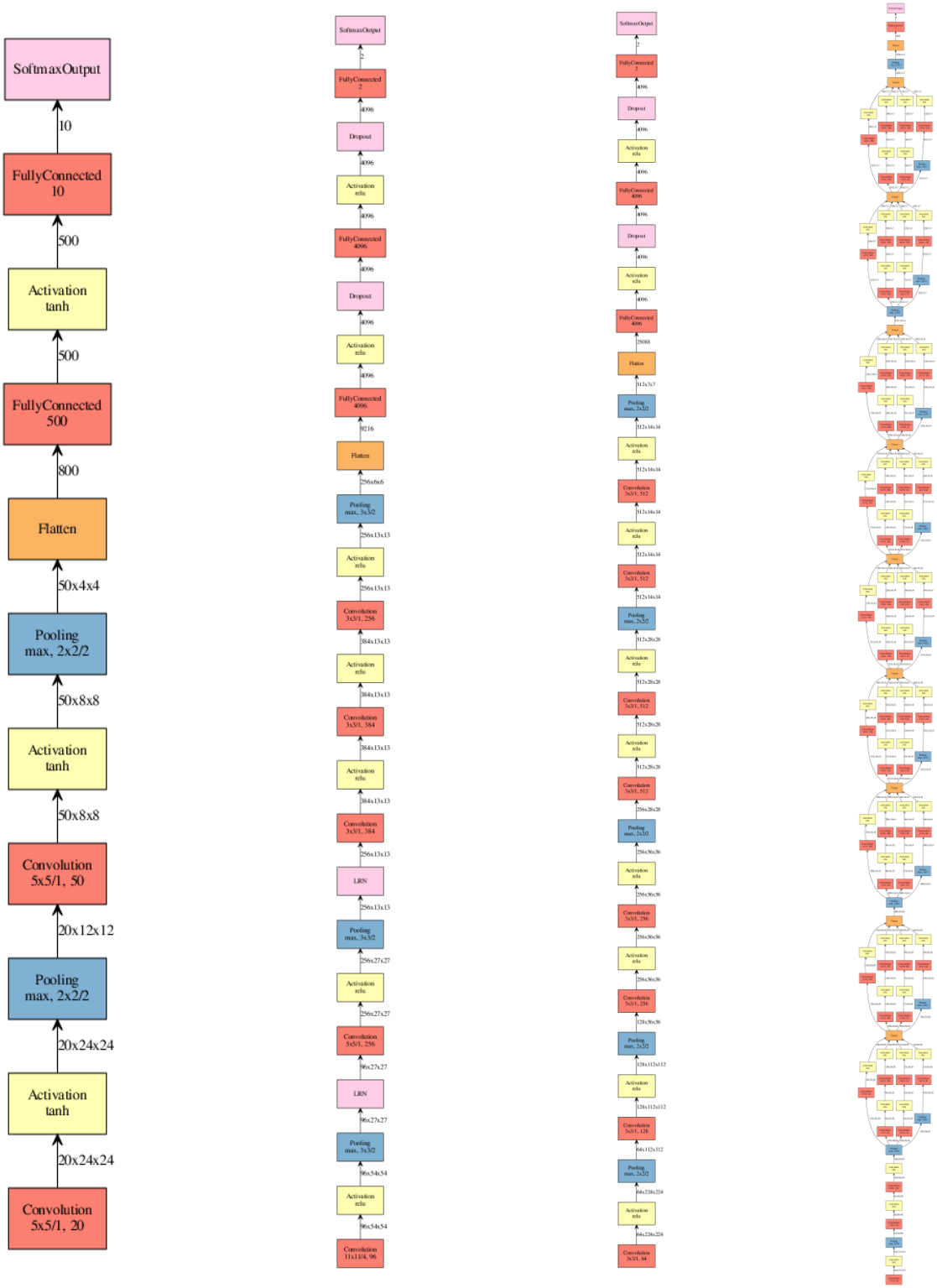


Fig. 17: Comparison of several famous CNN architectures. From left to right: LeNet5, LeCun et al, 1998. AlexNet, Krizhevsky et al, 2012. VGG, Simonyan et al, 2014. GoogLeNet, Szegedy et al, 2014. ResNet, K. He et al, 2015. Source: [21]

Note: Each colored block can represent several occurrences of the same layer. For instance, the VGG model includes 2 to 3 layers of convolution in a row for each convolution block in the schema. Also ResNet wasn't included in the comparison because if it were at the same scale as LeNet5, it would have been 13 pages long, and a reduced-size image wouldn't have been readable.

II. State of the art in Style Transfer

2.1 Image style transfer

2.1.1 Pre-neural style transfer

Before the use of neural networks, it was common to try to mimic the painting process of the artist. For instance, Litwinowicz *et al* [10] would repaint images by generating brush strokes to copy the style of the painter. The style obtained would depend on predefined parameters like the thickness or the quantity of strokes.

This method was particularly great at reproducing impressionist and pointillist painting styles, it was however limited to paintings as not every style can be recreated by brush strokes.

Color Plate 1. An original image. Plates 1-8 are 640x480 pixels.

Color Plate 2. Processed image using no brush stroke clipping and a constant base stroke orientation of 45°.

Color Plate 3. Technique of Color Plate 2 is modified so that brush strokes are cliped to edges detected in the original image.

Color Plate 4. Technique of Color Plate 3 is modified to orient strokes using a gradient-based technique.

Fig. 18: Illustration of the brush stroke technique developed by Litwinowicz et al to recreate an impressionist style

2.1.2 Image-optimization-based Neural Style Transfer

One of the most important paper on neural style transfer is probably *A Neural Algorithm of Artistic Style,* Gatys *et al*. 2015[11], the first paper published on this subject. It showed incredible results and opened the way to many more developments and variations of the idea of Neural Style Transfer.

The idea is to use a CNN pre-trained on an object detection task to extract the style of the target style image, the content of the target content image, and to optimize a new image to minimize the difference between its style (resp. content) and the one of the target style image (resp. target content image)

Some examples of this method are exposed in Fig. 19

2.1.2.1 content and style features extraction

One may ask how it is possible to quantify and extract the artistic style of an image.

As explained previously, CNN's convolution layers can extract features from the input image and the deeper the layer the more complex are the grasped patterns. To extract the style of an image, Gatys *et al*. used a CNN pretrained on object detection task (VGG19 trained on the ImageNet dataset), passed the image through the network and extracted the outputs of various layers across the network. Altogether, these tensors represent the artistic style, the shallower layers capture more of the texture and colors while the deeper layers capture higher-level information.

The authors have chosen the layers {'conv1_1', 'conv2_1', 'conv3_1', 'conv4_1', 'conv5_1'} (e) as the feature space for style representation.

In order to capture the content of the image, the authors used the deeper layers since they carry information about the content of the image and the disposition of the object but not about the exact pixel values. In their paper, they used the layer 'conv4_2' to capture the content of the images.

2.1.2.2 Image Optimization

Once the content and the style features are extracted from the target content image and the target style image, the output image is initialized with white noise. It is then fed into the VGG19 network to extract content features and style features in order to compute the content loss and the style loss. Finally, the derivative of the losses with respect to the activation layers are calculated, from which the gradients with respect to the input image are computed using standard error backpropagation. The input image can be modified and by iterating the process, until the desired result is obtained.

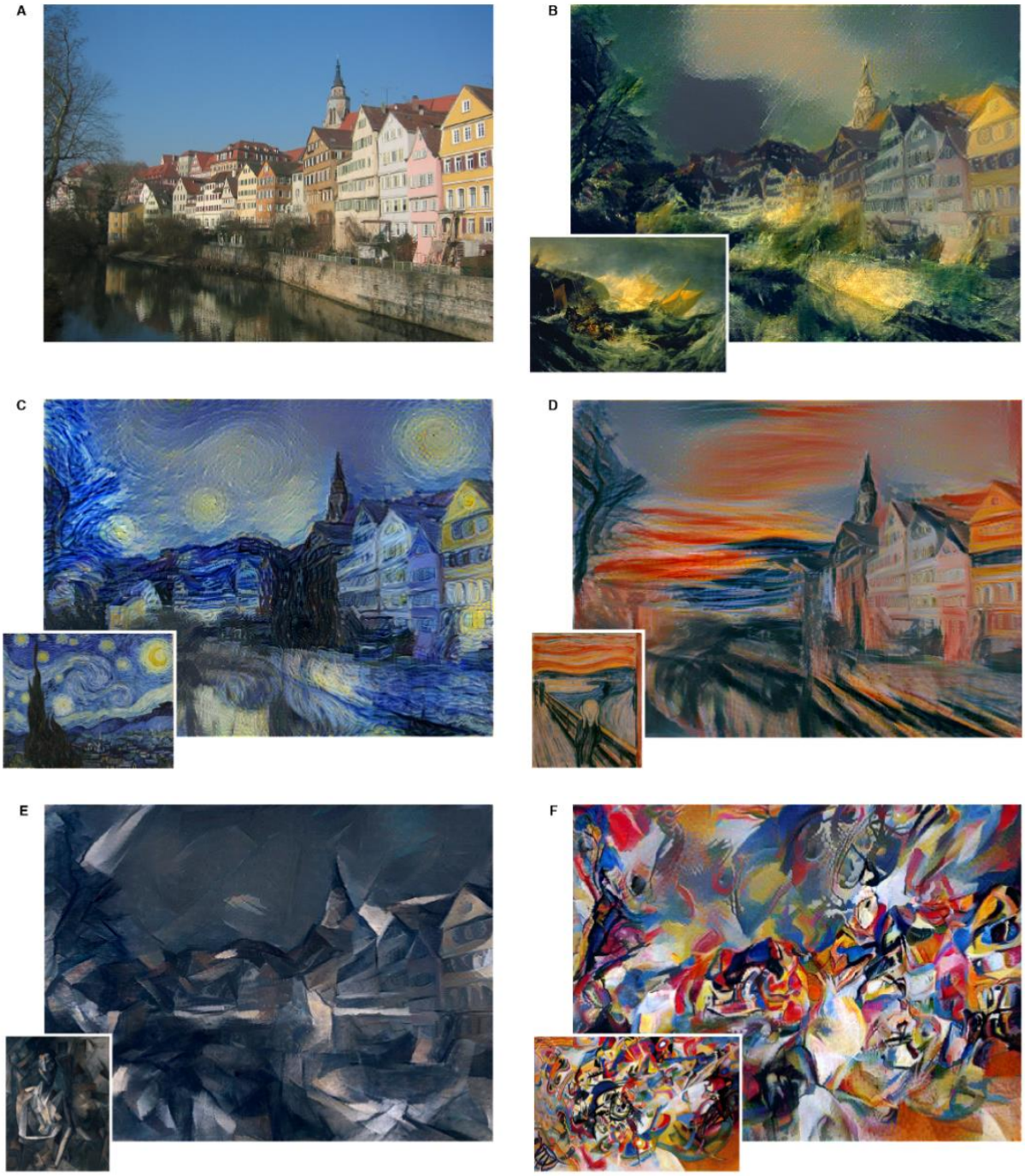


Fig. 19: Images that combine the content of a photograph with the style of several well-known artworks. The images were created by finding an image that simultaneously matches the content representation of the photograph and the style representation of the artwork (see Methods). The original photograph depicting the Neckarfront in Tübingen, Germany, is shown in **A** (Photo: Andreas Praefcke). The painting that provided the style for the respective generated image is shown in the bottom left corner of each panel. **B** *The Shipwreck of the Minotaur* by J.M.W. Turner, 1805. **C** *The Starry Night* by Vincent van Gogh, 1889. **D** *Der Schrei* by Edvard Munch, 1893. **E** *Femme nue assise* by Pablo Picasso, 1910. **F** *Composition VII* by Wassily Kandinsky, 1913.

Fig. 20: Detailed results for the style of the painting *Composition VII* by Wassily Kandinsky. The rows show the result of matching the style representation of increasing subsets of the CNN layers (see Methods). We find that the local image structures captured by the style representation increase in size and complexity when including style features from higher layers of the network. This can be explained by the increasing receptive field sizes and feature complexity along the network's processing hierarchy. The columns show different relative weightings between the content and style reconstruction. The number above each column indicates the ratio α/β between the emphasis on matching the content of the photograph and the style of the artwork (see Methods).

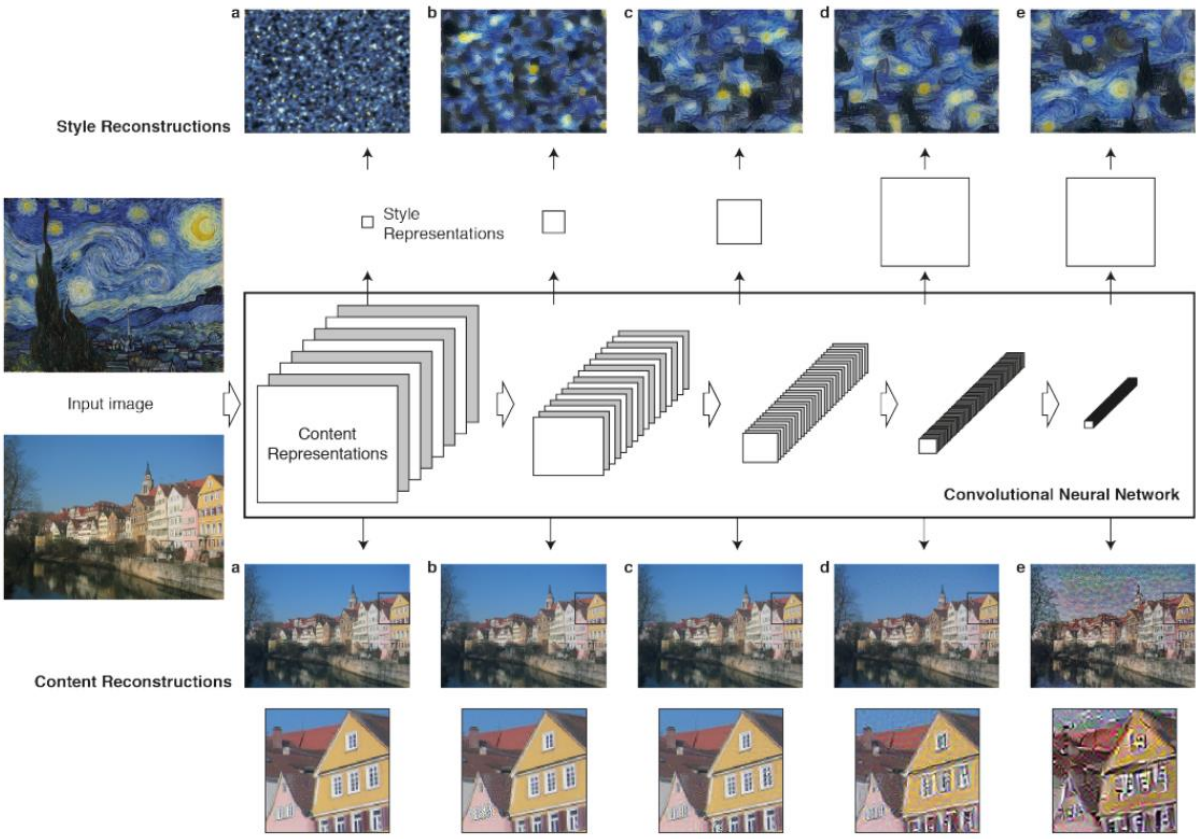


Fig. 21: Style reconstruction: a, b, c, d and e are the reconstruction of the style of the input image based on the layers {'conv1_1'}, {'conv1_1', 'conv'2_1'}, {'conv1_1', 'conv2_1', 'conv3_1'}, etc.
Content reconstruction: a, b, c, d and e are the reconstruction of the content of the input image based on the layers {'conv1_1'}, {'conv'2_1'}, {'conv3_1'}, etc.

Content Loss

Let L be a layer with N_L filters of size H_L x W_L. The output will be a map of shape N_L x H_L x W_L which can be reshaped as a N_L x M_L matrix with $M_L = H_L$ x W_L. Let $\vec{c}$ and $\vec{x}$ be the content image and the input image, and C^L and F^L the activation of their respective feature representation in L. The content loss is defined as

$$L_{content}(\vec{c}, \vec{x}) = \frac{1}{2}\sum_{i,j}(F^L{}_{i,j} - C^L{}_{i,j})^2$$

The derivative of the content loss with respect to $F^L{}_{i,j}$ is

$$\frac{\partial L_{content}}{\partial F^L{}_{i,j}} = \begin{cases} (F^L - C^L)_{i,j} & if\ F^L(X)_{i,j} > \\ 0 & if\ F^L(X)_{i,j} < \end{cases}$$

Style Loss

On top of the extracted features, the authors compute the correlation between the different filter's responses of a same layer. To do so, they compute the Gram matrix of the feature matrix, defined as following:

$$G^L_{i,j} = \langle F^L_i | F^L_j \rangle = \sum_k F^L_{i,k} F^L_{j,k}$$

Keeping the same notations as for the content loss, let $\vec{s}$ be the style image and S^L the Gram matrix of the activation of its feature representation in the layer L. Let U be the ensemble of the layers selected to extract the style features from.

The style loss is then defined as

$$L_{style}(\vec{s}, \vec{x}) = \sum_{L \in U} w_L E_L(\vec{x})$$

Where w_L are the weights of each layer in the total loss and $E_L(\vec{x})$ is the contribution of the layer L in the total style loss of $\vec{x}$,

$$E_L(\vec{x}) = \frac{1}{4N_L^2 M_L^2} \sum_{i,j} (G^L_{i,j} - S^L_{i,j})^2$$

The derivative of $E_L(\vec{x})$ with respect to $F^L{}_{i,j}$ is

$$\frac{\partial E_L(\vec{x})}{\partial F^L{}_{i,j}} = \begin{cases} \frac{1}{N_L^2 M_L^2} ((F^L)^T (G^L - S^L)_{j,i} & if\ F^L(X)_{i,j} > \\ 0 & if\ F^L(X)_{i,j} < \end{cases}$$

Total Loss

Finally, the total loss is defined as

$$L_{total}(\vec{c}, \vec{s}, \vec{x}) = \alpha L_{content}(\vec{c}, \vec{x}) + \beta L_{style}(\vec{s}, \vec{x})$$

With α and β the weights for content and style reconstruction. These parameters control the tradeoff between the content similarity and the style similarity of the result to the targets. The impact of the ratio α / β is demonstrated in Fig. 20

After Gatys *et al*'s first publication on neural style transfer for images, many variations of this idea were proposed, and some are worth mentioning.

Chuan Li and Michael Wand combined CNN with Markov Random Fields (MRF) [25]. This combination allowed a better consistency and achieve both photorealistic and abstract style transfer.

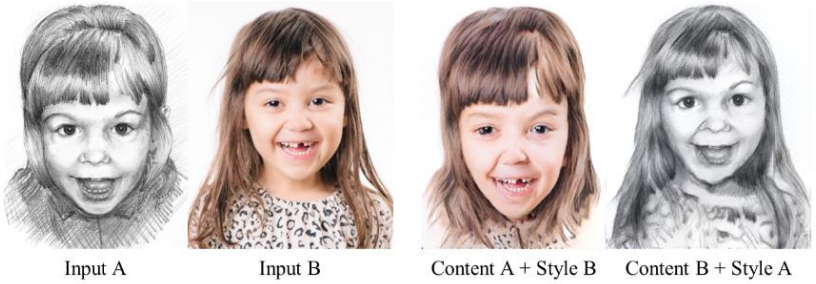


Fig. 22: Example of style transfer performed by Li and Wand [25]

Also, Funjun *et al* [24] proposed to add a photorealism regularization term to the total loss to allow Photo style transfer and remove the distortion and artifacts produced by Gatys *et al*'s method. This method makes it possible to switch a photo from daytime to nighttime

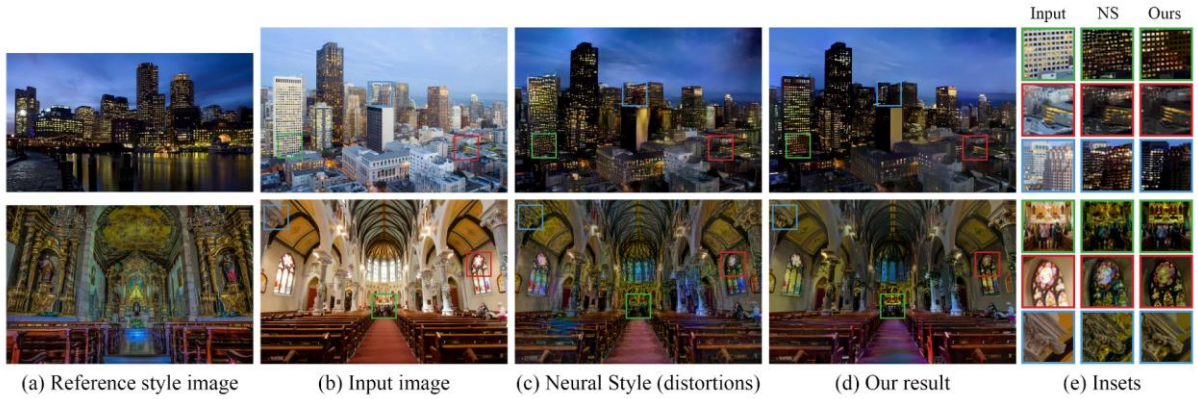


Fig 23: Example of style transfer performed by Funjun et al [24]

Champanard *et al* [26] proposed a semantic-map-based style transfer method that allowed to create visually appealing images from sketches, or to perform style transfer between two semantically close pictures. This technique allows a greater degree of control on the semantic maps and on the end results

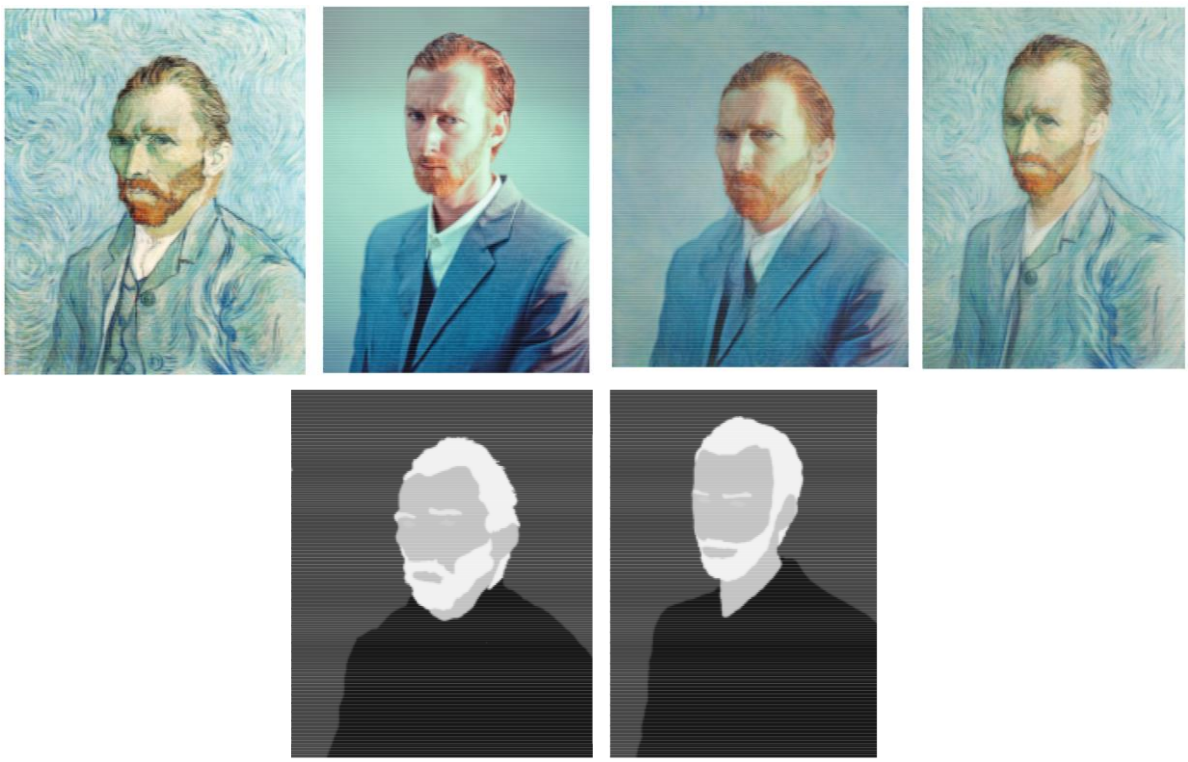


Fig. 24: Example of semantic style transfer performed by Champanard et al [26]

2.1.3 Network-optimization-based Neural Style transfer

The main limitation of the technique we reviewed is their speed. For Gatys *et al*'s method, producing one image of 512x512 pixels by 100 iterations (a reasonable number of iterations) could take more than 10 seconds on a high-quality GPU [27]. This could reach more than 2 minutes on the same hardware for a 1024x1024 image and 300 iterations.

Such low speed would make this mode unusable for real-time processing. The next model we will explore achieved a speed up of 3 orders of magnitude, taking only 0.05s to process a 512x512 image instead of 54,85s for Gatys *et al*'s model with 500 iterations. This breakthrough happened without perceptible loss of quality and this method can be used to perform state-of-the-art image super-resolution.

In "Perceptual Losses for Real-Time Style Transfer and Super-Resolution", Johnson *et al* proposed a novel method where a feed-forward CNN is trained to translate input images into a pre-defined style, and an auxiliary loss network (a fixed VGG 16) is used to compute style and content losses. The computational burden is therefore transferred on the training phase and the style transfer only takes one pass forward through the network. However, one network can only be trained on one style, which means that performing style transfer with a new style requires the training of a new CNN which takes a few hours.

Here is an overview of the model:

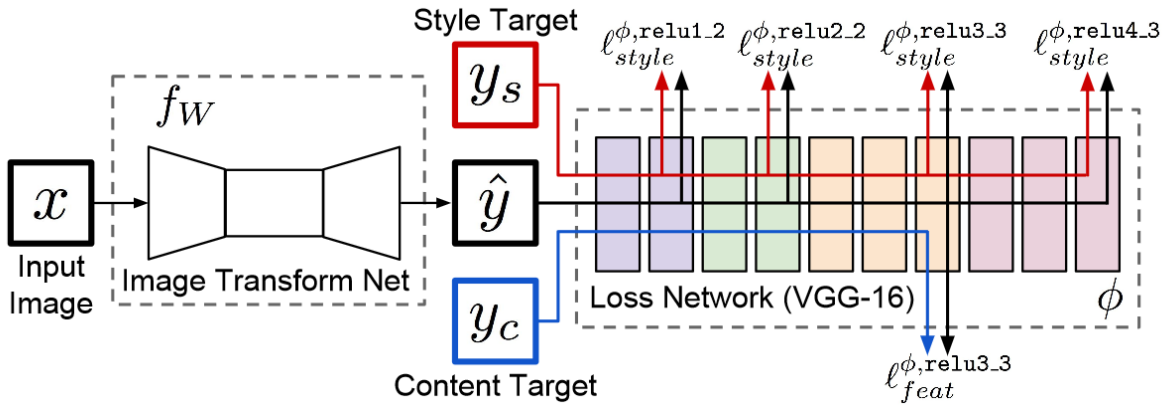


Fig. 25: Overview of Johnson et al's system for fast style transfer

The input image x is fed into an Image Transform Net, a deep residual convolutional neural network, that transforms it into the output image $\hat{y}$. Once the network is trained, nothing more needs to be done and the output image $\hat{y}$ will be the stylized version of x with the style of the style of the target. During the training process, the Loss network is used to compute the style and content losses in the same manner as in Gatys *et al*'s method [11], with slightly different loss functions. The style target y_s, the content target y_c and the output image are fed into the network, the activations of the content layer are extracted to compute the content loss, the activations of the style layers are extracted to compute the style loss, and the total loss is backpropagated through the Image Transform Net to adjust its weights. Here the Image Transform Net is optimized, not the input image nor the Loss Network.

Image Transform Net

This feed-forward network consists of two 2-strided convolutional layers for downsampling, each followed by a batch normalization and ReLU activation, residual blocks as the core of the network, and two fractionally-strided convolutional layer of stride ½.

This downsampling – upsampling architecture reduces the computational cost of the residual layers, which allow the use of a bigger network for the same cost. A convolutional layer with 3x3 kernels with C filters and an input of size $C * H * W$ requires 3^2HWC^2 multiplications, which is the same amount as a convolutional layer of size 3x3 kernels with DC filters and the same input downsampled by a factor D, $DC * \frac{H}{D} * \frac{W}{D}$.

This also increase the Effective Receptive Field (ERF) of the convolutional layers of the residual blocks, i.e the size of the input region that affect a layer's activations.

Content loss

Named Feature Reconstruction Loss by the authors, this content loss is the normalized squared Euclidian distance between the feature representation of the target and output image, extracted from the content layer of the Loss Network.

Taking the same notation as in 2.1.2.2 – Content Loss, let y_c and $\hat{y}$ be the content target and the output of the Image Transform Net, and C^L and F^L the activation of their respective feature representation in the layer L of the Loss Network. The content loss is defined as

$$L_{content}(\hat{y}, y_c) = \frac{1}{N_L H_L W_L} \|F^L - C^L\|_2^2 = \frac{1}{N_L H_L W_L} \sum_{i,j} ({F^L}_{i,j} - {C^L}_{i,j})^2$$

Which is the content loss used by Gatys *et al* [11], normalized.

Style Loss

Named Style Reconstruction Loss by the authors, it is quite similar to the Style Loss introduced by Gatys *et al*.

Taking the same notations as in 2.1.2.2 – Style loss, the normalized Gram matrix is defined as

$$G^L_{i,j} = \frac{1}{N_L H_L W_L} \langle F^L_i | F^L_j \rangle = \frac{1}{N_L H_L W_L} \sum_k F^L_{i,k} F^L_{j,k}$$

Then, let y_s be the style image and S^L the Gram matrix of the activation of its feature representation in the layer L of the Loss Network. The style reconstruction loss associated with the layer L of the Loss Network is defined as the squared Frobenius Norm if the differences between the Gram matrices of the style image and the output from the Image transform Network. The total style loss is the sum of the style reconstruction losses associated with the layer L for each $L \in U$

$$L_{style}(y_s, \hat{y}) = \sum_{L \in U} \|G^L - S^L\|_F^2$$

Total Variation Regularization

To encourage the smoothness of the output image, another loss is added whose goal is to minimize the Euclidian distance between two adjacent pixels. We define the Total Variation Regularization of an image y as

$$L_{TVR}(y) = \sum_{i,j} ((x_{i,j} - x_{i,j+1})^2 + (x_{i,j} - x_{i+1,j})^2)^{\frac{1}{2}}$$

The Image Transform Network is then trained using Stochastic Gradient Descent to minimize a linear combination of the loss functions with respect to the input image x and the fixed content and style target images y_c and y_s

$$W^* = \arg\min_W E_{x,y_c,y_s} [\lambda_c L_{content}(\hat{y}, y_c) + \lambda_s L_{style}(\hat{y}, y_s) + \lambda_{TV} L_{TVR}(\hat{y})]$$

This model performs orders of magnitude faster than Gatys *et al*'s model, as showed in the benchmark below ran on a GTX Titan X GPU.

	Gatys *et al*				Speedup		
Image Size	100	300	500	Ours	100	300	500
256 × 256	3.17	9.52s	15.86s	**0.015s**	212x	636x	**1060x**
512 × 512	10.97	32.91s	54.85s	**0.05s**	205x	615x	**1026x**
1024 × 1024	42.89	128.66s	214.44s	**0.21s**	208x	625x	**1042x**

Fig. 26: Benchmark conducted by Johnson et al

Fig. 27: Qualitative comparison of Johnson et al's model and Gatys et al's model

Fig. 28: Detail of the previous figure. Left, Gatys et al. Right, Johnson et al.

While the authors stated that the quality of the results of their method is qualitatively similar to those of Gatys *et al*'s image-optimization-based method, the following papers tended to notice a lower quality [31]. This is noticeable in the comparison given by the authors on Composition VII by Wassily Kandinsky. This loss of quality for the benefit of speed has been the subject of several follow-up papers [30, 31] whose goal was to modify the proposed method to get a higher quality output for the same computational cost, among others.

Another approach was developed by, Ulyanov *et al* [30] for texture transfer and texture synthesis based on their proposed Texture Network architecture. As suggested by the name, this model performs well for texture transfer and texture synthesis but lack higher-level consistency when it comes to style transfer.

Their method consists on a feed-forward generator network g that can be trained to produce a texture sample $g(z)$ from a noise sample z that resembles to the texture of a given texture target image y_s. This texture generation method can then be extended to perform texture transfer by taking as input a noise sample z and a content image y_c. See Fig. 29 below for a graphical overview.

The performances of the generator network are evaluated by a descriptor network (namely VGG19) as in Johnson *et al*'s method, uses the same content loss and style loss (named texture loss by the author) and the same training method (SGD).

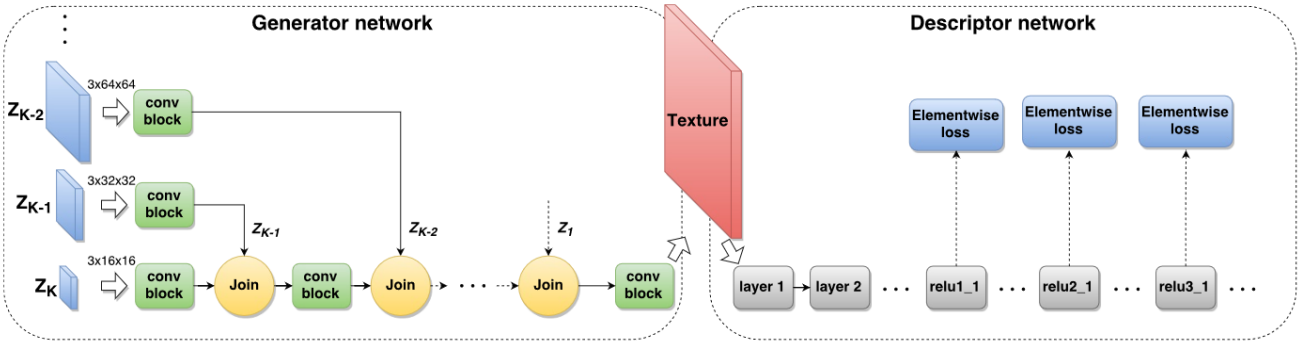


Fig. 29: Overview of the proposed architecture (texture networks). We train a *generator network* (left) using a powerful loss based on the correlation statistics inside a fixed pre-trained *descriptor network* (right). Of the two networks, only the generator is updated and later used for texture or image synthesis. The `conv` block contains multiple convolutional layers and non-linear activations and the `join` block upsampling and channel-wise concatenation. Different branches of the generator network operate at different resolutions and are excited by noise tensors z_i of different sizes.

As we can see in Fig 30, and Fig 31, this method is able to grasp and reproduce textures at state-of-the-art level but gives poorer results for style transfer, as the produced image is quite repetitive and lacks semantic understanding.

Fig. 30: Model comparison for texture generation.

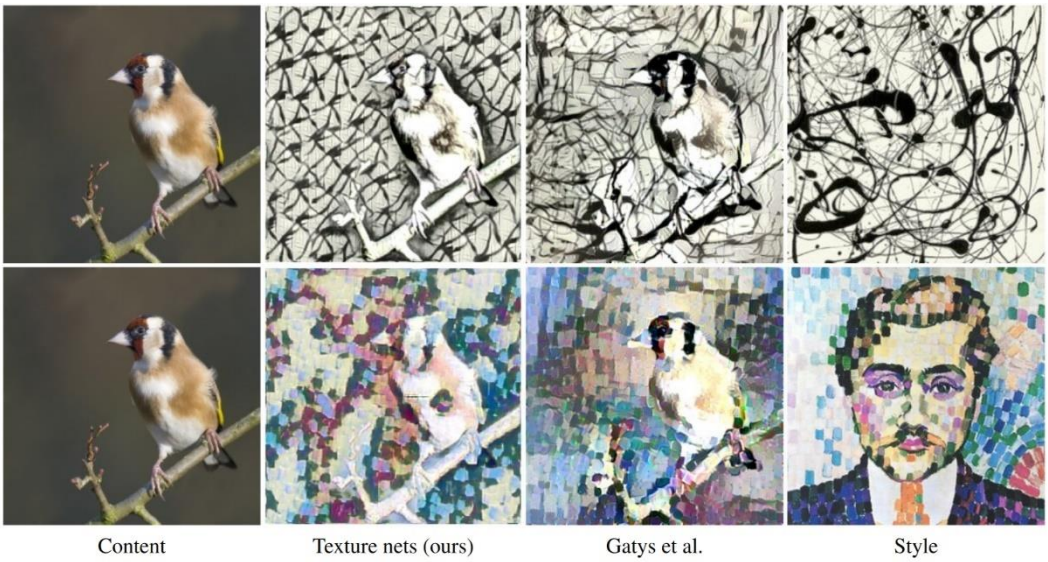


Fig. 31: Model comparison for style transfer

2.1.4 Instance Normalization

A notable incremental improvement from the previous method is the replacement of Batch Normalization (BN) modules by Instance Normalization (IN) module.

Ulyanov *et al* published an improved version [31] of their texture network released one year earlier [30] that we described earlier, building on the residual Image Transform Net proposed by Johnson *et al* [27]. One of the main contributions of this paper was the introduction of Instance Normalization, although quite similar to the *layer normalization* introduced by J. L. Ba *et al* [34]. Given an input tensor $X = (x_{n,i,j,k})$ of shape $N * C * H * W$, the Instance Normalization output $Y = (y_{n,i,j,k})$ is defined as

$$y_{n,i,j,k} = \frac{x_{n,i,j,k} - \mu_{n,i}}{\sqrt{\sigma_{n,i}^2 + \varepsilon}}$$

With

$$\mu_{n,i} = \frac{1}{HW}\sum_{j,k} x_{n,i,j,k}$$

$$\sigma_{n,i}^2 = \frac{1}{HW}\sum_{j,k}(x_{n,i,j,k} - \mu_{n,i})^2$$

Instance Normalization differs from the Batch Normalization because the former performs the normalization to each channel of each element of the batch, while the latter performs the normalization to each channel for the entire batch:

$$y_{n,i,j,k} = \frac{x_{n,i,j,k} - \mu_i}{\sqrt{\sigma_i^2 + \varepsilon}}$$

With

$$\mu_i = \frac{1}{HW}\sum_{n,j,k} x_{n,i,j,k}$$

$$\sigma_i^2 = \frac{1}{HW}\sum_{n,j,k}(x_{n,i,j,k} - \mu_i)^2$$

This minor difference has great consequences on the output quality (see fig.32) without impacting negatively the computational cost of the model and reduces the convergence time. The reason for this evolution is, according to the authors, that the result of a style transfer should not depend on the contrast of the image but on the content, and that applying normalization to each channel of the batch individually instead of at the batch level would discard the unnecessary contrast information.

This paper also proposes new methods for texture synthesis, but this is separated from the contributions on style transfer.

Fig 32. Comparison of network-optimization-based style transfer with Batch Normalization, Instance Normalization, and image-optimization-based style transfer. StyleNet refers to the Image Transform Net introduced by Johnson *et al* [27]

2.1.5 Network-optimization-based Multi-style Neural Style transfer

Even though the network-optimization-based approaches proposed by Johnson *et al*[27] and improved Ulyanov *et al*[30, 31] allows a near-real-time processing speed once the network is trained on a style, one network can only learn one style. This has been the subject of further research by various universities, and several improvements have been proposed to enable network-optimization-based arbitrary style transfer, i.e models able to perform style transfer on with any given style image (and content image) in real-time or almost.

To my knowledge, the first paper published on multi-style network-optimization-based was "A Learned Representation for Artistic Style" by Dumoulin *et al.*, 2017 [29]. They introduced a Conditional Instance Normalization module (CIN) inspired by Ulyanov *et al*'s Instance Normalization [31] that allowed a single network to learn multiple pre-defined styles at once. Although this model didn't permit to perform *arbitrary* style transfer, it was a step in the right direction.

The losses and the learning process are identical to the ones proposed by Johnson *et al* [27], and the feed-forward transformation network is almost identical as Ulyanov *et al*'s StyleNet, the only difference being the normalization modules.

Taking the same notations as in 2.1.4, the authors added two parameters β and γ, matrices of shape $N_s * C$ with N_s the number of styles to be learnt.

$$CIN(x,s) = \gamma_S \left(\frac{x - \mu(x)}{\sigma(x)}\right) + \beta_S$$

With μ and σ x's mean and standard deviation across H and W the spatial dimensions, and γ_S and β_S are the rows of γ and β corresponding to the style s. The CIN module is summarized in fig 33 below.

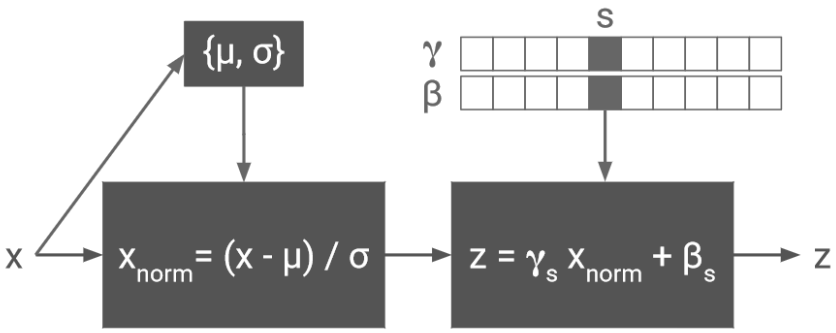


Fig. 33: Schematic representation of the Conditional Instance Normalization.

The intuition behind this modification is that between two different trained single-style feed-forward model, many of the weight will be similar, therefore it should be possible to combine the layers of one network to make it recreate the style that the second learnt.

This method requires the parameters γ_s and β_s to be trained for each specific style, which means that it can only transfer styles that has been trained on and cannot perform style transfer for a previously unseen style. However, it is possible to quickly fine-tune an existing model and learn new γ_s and β_s for a new style. As we can see in fig 34, a new style can be learned in 5000 steps instead of 40000 when training a new single-style model.

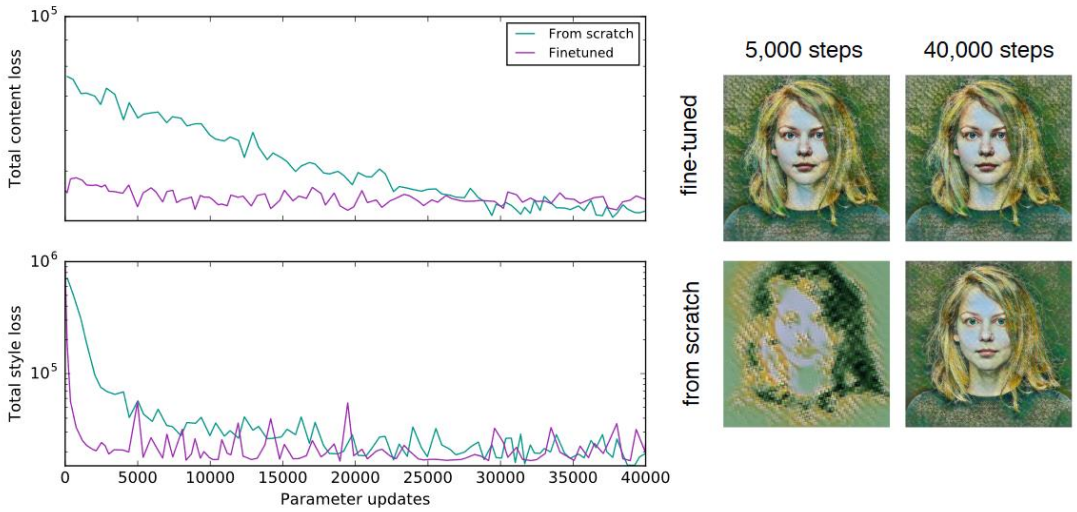


Fig. 34: The trained network is efficient at learning new styles. (Left column) Learning γ and β from a trained style transfer network converges much faster than training a model from scratch. (Right) Learning γ and β for 5,000 steps from a trained style transfer network produces pastiches comparable to that of a single network trained from scratch for 40,000 steps. Conversely, 5,000 step of training from scratch produces leads to a poor pastiche.

2.1.6 Network-optimization-based Arbitrary Neural Style transfer

Finally, X. Huang and S. Belongie proposed in 2017 a method [33] to perform real-time arbitrary style transfer, i.e a model that could learn to transfer the style of any target image to any content image. To do so, they introduced a new type of normalization module called Adaptative Instance Normalization (AdaIN), inspired from Ulyanov *et al*'s IN and Dumoulin *et al*'s CIN.

The idea, simple but smart, of the authors, was to align the channel-wise mean and variance of the input image x to those of the style target y_s

Taking the same notations as in 2.1.5, the Adaptative Instance Normalization of

$$AdaIN(x, y_s) = \sigma(y_s)\left(\frac{x - \mu(x)}{\sigma(x)}\right) + \mu(y_s)$$

Feed-forward network architecture

Unlike in the previous paper studied, this method uses a simpler encoder-decoder structure. The encoder f is a fixed pre-trained VGG19 network truncated at the layer relu4_1, with reflection padding to avoid border artifacts. Then, after encoding both content and style images through f, both are fed into the AdaIN module to align the mean and variance of the content image to those of style image, producing the target features

$$t = AdaIN(f(y_c), f(y_s))$$

Then the decoder is built to mirror the encoder, with some variations: all pooling layers are replaced by upsampling ('nearest' mode) to avoid checkerboard effects, reflection padding is used to avoid border artifacts and *no normalization* is used (not BN nor IN) as it would decrease performances.

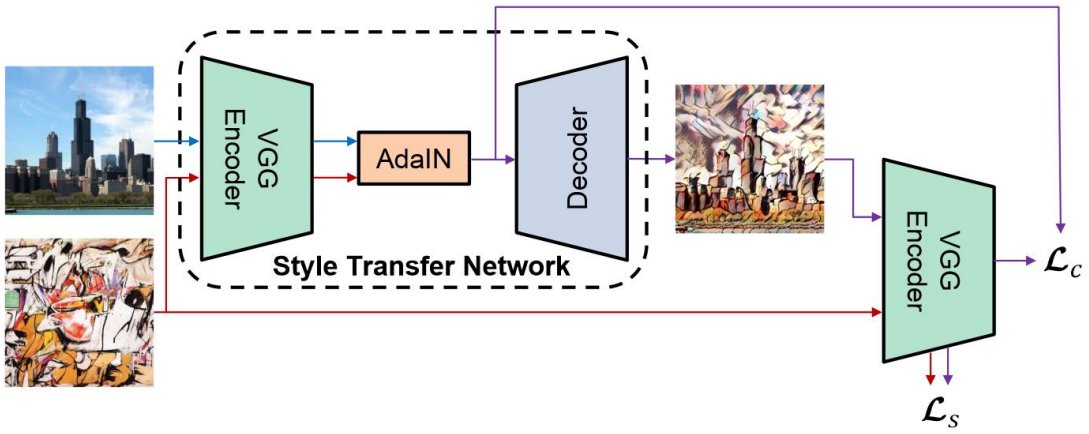


Fig 35. Architecture summary

Losses

The content loss is defined as the Euclidian distance between the target features and the output image features. The authors used t as the target features.

$$L_{content} = \left\| f(g(t)) - t \right\|_2$$

For the style loss, the authors used the style loss defined by Li *et al* [35]:

$$L_{style} = \sum_{L \in U} \left\| \mu\left(F^L(g(t))\right) - \mu(F^L(y_s)) \right\|_2 + \sum_{L \in U} \left\| \sigma\left(F^L(g(t))\right) - \sigma(F^L(y_s)) \right\|_2$$

This system is then trained with an Adam optimizer on the MS-COCO dataset for the content images and on a dataset made of images collected from WikiArt. It is important to note that only the decoder's weights are being modified

Performances

The authors conducted a benchmark of their method compared to other popular methods. Not only can it perform arbitrary style transfer, but this can be done in real-time.

Method	Time (256px)	Time (512px)	# Styles
Gatys *et al.*	14.17 (14.19)	46.75 (46.79)	∞
Chen and Schmidt	0.171 (0.407)	3.214 (4.144)	∞
Ulyanov *et al.*	**0.011** (N/A)	**0.038** (N/A)	1
Dumoulin *et al.*	**0.011** (N/A)	**0.038** (N/A)	32
Ours	**0.018** (0.027)	**0.065** (0.098)	∞

Fig 36. Benchmark conducted by the authors on a Pascal Titan X GPU

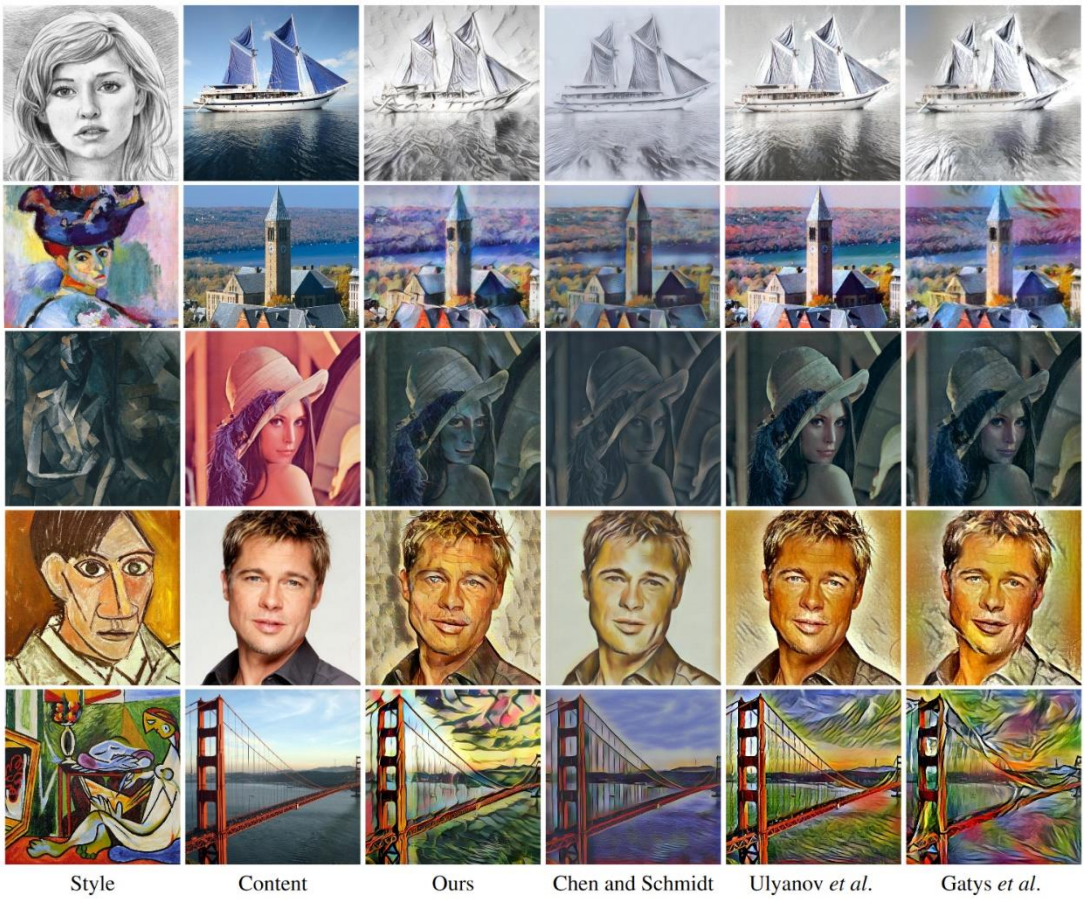


Fig 37. Comparison of the Style Transfer results with other popular methods. It is important to note that the style images were not present in the training set and are therefore previously unseen by the network.

2.2 Video Style Transfer

Although the previous method is able to run in real-time and produce quality images, it could not guarantee a quality result for style transfer in videos. The reason is that by processing a video frame by frame with image style transfer methods, two adjacent frames are likely to have significantly different aspect since a slight change in the image noise can have a great impact on the style transfer result. A flickering effect would appear, which is not desirable.

To add this desired temporal coherency, the main idea exploited in different ways is to add a temporal loss that would ensure that the learning process takes into account the previous frames to produce a smoother output.

2.2.1 Video-optimization-based video style transfer

The first paper going in this direction has been published by Ruder *et al* in 2016[41], which propose an adaptation of Gatys *et al*'s image-optimization-based method adding a temporal constraint to enforce smooth transition between frames. This temporal constraint uses the optical flow between frames to penalize deviations between frames along the point trajectories.

Optical Flow

Optical flow is the pattern of motion of objects and shapes between two consecutive frames, i.e the distribution of apparent velocities of movement of brightness pattern. While the term has been introduced by the psychologist James J. Gibson, it is widely used in computer vision and robotics to understand motions in real-life environments for instance.

Optical flow can be sparse or dense. The former refers to a method where optical flow is computed at some locations only, on a given grid or at points of interest for instance. The latter refers to a method where optical flow is computed for each pixel of the image, which is more precise but computationally more expensive.

Several methods for optical flow computation have been proposed by Fleet and Weiss [36], but the authors used deep learning-based techniques for optical flow estimation as DeepFlow [37] or EpicFlow [9]

Losses

We are working with a video input, i.e a tensor of size $L * C * H * W$ with L the length of the video, C the number of channels (C=3 for RGB encoding), and H and W the frames height and width. We will note $x^{(i)}$ the i-th frame of the video that will serve as content image, and $y^{(i)}$ the corresponding stylized output

The content and style losses are similar to the previous ones. The content loss is the mean square error between the content feature representations of the output $y^{(i)}$ and the content image $x^{(i)}$. Taking the same notation as in 2.1.3, we have:

$$L_{content}(x^{(i)}, y^{(i)}) = \frac{1}{N_L H_L W_L} \sum_{i,j} ({F^L}_{i,j} - {C^L}_{i,j})^2$$

The style loss is the mean squared error (not Frobenius norm as previously) between the Gram matrices of the feature representations of the output $y^{(i)}$ and the style image s, summed across the layers $L \in U$

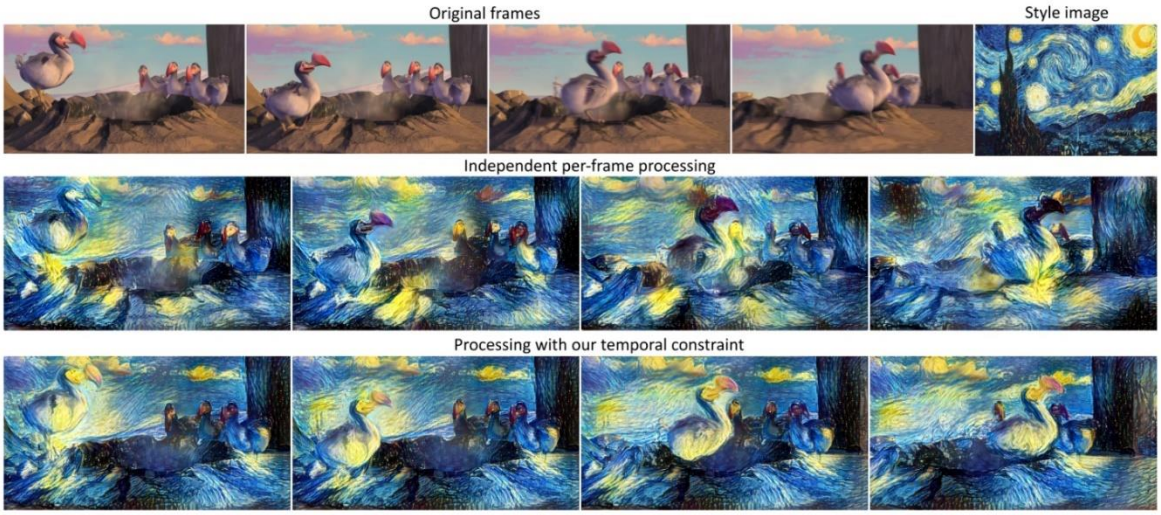


Fig. 38: Scene from *Ice Age* (2002) processed in the style of *The Starry Night*. Comparing independent per-frame processing to our time consistent approach, the latter is clearly preferable. Best observed in the supplemental video, see section 8.1.

$$L_{style}(s, y^{(i)}) = \sum_{L \in U} \left(\frac{1}{{N_L}^2 {H_L}^2 {W_L}^2} \sum_{i,j} ({G^L}_{i,j} - {S^L}_{i,j})^2 \right)$$

The temporal loss is slightly more complex. Its goal is to penalize deviation between the output image i and the previous output warped with forward optical flow, where this optical flow is estimated with high confidence. The confidence of the optical flow is defined pixel-wise as 0 in disoccluded regions (regions occluded in frame $x^{(i-1)}$but visible in frame $x^{(i)}$) and motion boundaries, and 1 elsewhere.

Let x and y be the first and second frame, $w(y^{(i-1)}, y^{(i)}) = (u, v)$ be the optical flow in the forward direction, $\hat{w}(y^{(i-1)}, y^{(i)}) = (\hat{u}, \hat{v})$ the optical flow in the backward direction, and $\tilde{w}$ the forward optical flow warped to the second image:

$$\tilde{w}\left(y^{(i-1)}, y^{(i)}\right) = w(\left(y^{(i-1)}, y^{(i)}\right) + \hat{w}\left(y^{(i-1)}, y^{(i)}\right))$$

Occlusions can be detected by comparing the forward and the backward optical flow, as detailed by Sundaram *et al* [39]. For non-occluded regions, the forward optical flow of a given pixel should be the opposite of its backward optical flow. If this is not the case, the point is either being occluded at the frame t+1 or the optical flow was not correctly computed, which are two valid reasons not to include this point in the temporal loss computing. Therefore, an area is flagged as disoccluded if:

$$|\hat{w} + \tilde{w}|^2 > 0.01(|\tilde{w}|^2 + |\hat{w}|^2) + 0.5$$

Also, the exact location of the motion boundariy is subject to fluctuation, which can lower the quality of the optical flow estimation in these areas. Motion boundaries are detected using the following inequality:

$$|\nabla \hat{u}|^2 + |\nabla \hat{v}|^2 > 0.01|\hat{w}|^2 + 0.002$$

Using the previous two inequalities, a confidence mask between the frames i-1 and I is defined such as $c^{(i-1,i)} \in \{0,1\}^{C*H*W}$ is 0 in disocclued regions and motion boundaries, and 1 elsewhere.

The temporal loss is defined as:

$$L_{temporal}(y,w,c) = \frac{1}{CHW}\sum_{n,i,j} c_{n,i,j}(y_{n,i,j} - w_{n,i,j})^2$$

Finally, the total loss is:

$$L_{tot}\left(x^{(i)}, s, y^{(i-1)}, y^{(i)}\right) = \alpha L_{content}\left(x^{(i)}, y^{(i)}\right) + \beta L_{style}(s, y^{(i)}) + \gamma L_{temporal}\left(y^{(i)}, w_{i-1}^{i}(y^{(i-1)}), c^{(i-1,i)}\right)$$

With $w_{i-1}^{i}(x^{(i-1)})$ the frame $x^{(i-1)}$ warped with the optical flow between frames i-1 and i.

Also, the authors extended the temporal loss to give greater long-term coherency by summing the temporal losses associated with several frames gap. E.g. instead of comparing the input and output frames at step i and i-1, it is possible to compare the input and output frames at steps i and i-j for any given j. The total loss is modified as following:

$$L_{tot}\left(x^{(i)}, s, y^{(i-j)}, \dots, y^{(i)}\right) = \alpha L_{content}\left(x^{(i)}, y^{(i)}\right) + \beta L_{style}(s, y^{(i)}) + \gamma \sum_{j \in J : i-j \geq 1} L_{temporal}\left(y^{(i)}, w_{i-j}^{i}(y^{(i-j)}), c_{long}^{(i-j,i)}\right)$$

The definition of the confidence mask is also updated and defined as:

$$c_{long}^{(i-j,i)} = max\left(c^{(i-j,i)} - \sum_{k \in J : k < j} c^{(i-k,i)}, 0\right)$$

The effect of long-term temporal loss is visible in fig 39, as well as in the supplementary material released by the authors along with more demonstrations, available here: https://www.youtube.com/watch?v=vQk_Sfl7kSc

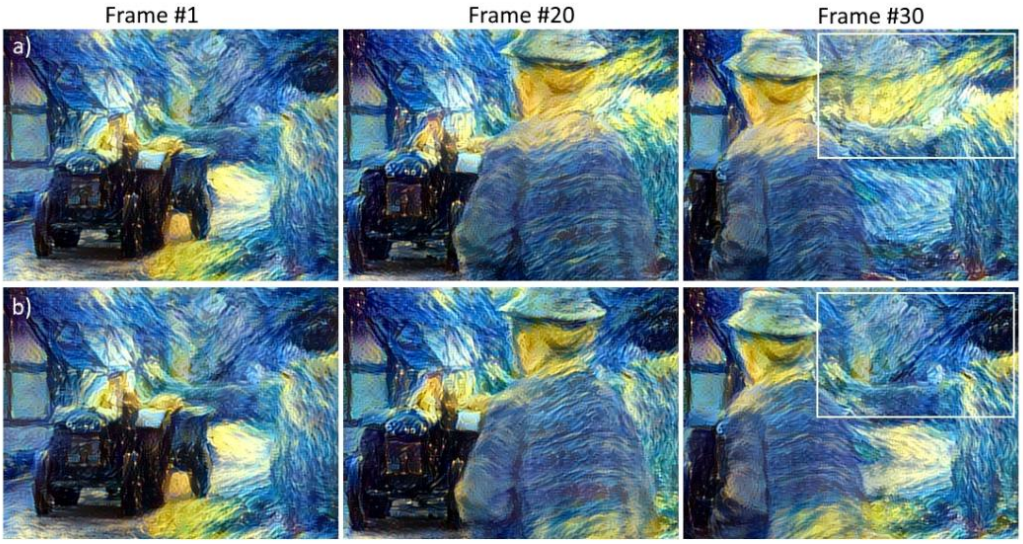


Fig. 39: Scene from Miss Marple, combined with The Starry Night painting. **a)** Short-term consistency only. **b)** Long-term consistency with $J = \{1, 10, 20, 40\}$. Corresponding video is linked in section 8.1.

Training process

The authors found out that with a regular training process such as proposed by Gatys *et al* [11] the output image tends to have less contrast than expected, and that combined with the temporal coherence this led to lower the quality of the output through time. In order to solve this, they proposed a multi-pass algorithm that processes the entire video sequence both forward and backward. Every pass contains less iterations and does not goes until full convergence so that the intermediate result obtained can propagate forward and backward through the video and avoid artifacts caused by the one-way information flow. See fig 40 below for a demonstration of the contribution of the multi-pass algorithm.

Fig. 40: The multi-pass algorithm applied to a scene from Miss Marple. With the default method, the image becomes notably brighter and loses contrast, while the multi-pass algorithm yields a more consistent image quality over time. Corresponding video is linked in section 8.1.

Performances

This method allowed high quality results with both short-term and long-term temporal coherence (see fig 38.), but it is not applicable for our use case. Not only is it way too slow since it is based on the initial image-optimization process proposed by Gatys *et al* [11], but it also requires a multi-pass process to produce better results. This is not desirable, since we require to process a video in real-time and therefore without having access to the next frames.

2.2.2 Network-optimization-based video style transfer

Two years after publishing their *Artistic style transfer for videos*, Ruder *et al* proposed a novel and much faster version of their previous algorithm [40], built upon Johnson *et al*'s Image Transform Network [27] and their previous temporal loss.

The content, style and temporal losses (either short-term or long-term) are the same here as previously [41]. We keep here the same notations. The authors extended Johnson *et al*'s Image Transform Network to take as input not only the content image, but also the stylized previous frame as *prior knowledge* and the per-pixel confidence mask defined previously. They also changed the upsampling layers from fractionally-strided convolutional layers to a combination of nearest-neighbor upsampling and regular convolutional layer, to reduce the checkerboard artifacts visible in fig 28. This follows the recommendations given by Odena *et al* [42] who investigated the reasons and the solutions to the checkerboard artifacts that were visible in CNN using deconvolution (or fractionally-strided convolution) layers.

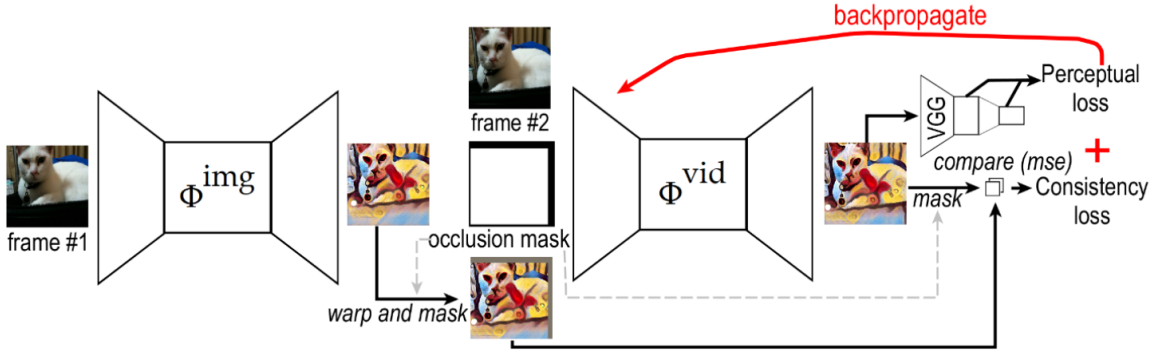


Fig 41. Mixed training

The authors propose two training methods. The first consist in mixing training examples that have a *prior knowledge* image, and some that don't. This aims at forcing the network to keep on refining the learning at every step instead of just lazily extract information from the prior knowledge image. The objective function is in this case:

$$W^* = \arg\min_W E_{x^{(1)},x^{(2)}}\left[L_{tot}(x^{(2)}, s, y^{(2)}, y^{(1)})\right] + E_p\left[L_{tot}(x^{(1)}, s, \Phi_W^{vid}(x^{(1)}, 0, 0), 0)\right]$$

With

$$y^{(2)} = \Phi_W^{vid}(x^{(2)}, w_1^2(y^{(1)}), c^{(1,2)})$$

And O a tensor filled with zeros of the shape of the input used for the initialization when no optical flow nor confidence mask can be computed.

The other method proposed, called multi-frame training, pass recursively the input frames $f^{(1)}$ to $f^{(N-1)}$, so that the training is performed on the potentially degenerated stylized image $y^{(N-1)}$as prior knowledge image. The network has to generate a non-degenerated stylized image $y^{(N)}$, which trains the network to correct degeneration. The objective function in this case is:

$$W^* = \arg\min_W E_{x^{(t)},x^{(t+1)}} \left[L_{tot}(x^{(t+1)}, s, y^{(t+1)}, y^{(t)}) \right]$$

With

$$y^{(t+1)} = \Phi_W^{vid}(x^{(t+1)}, w_t^{t+1}(y^{(t)}), c^{(t,t+1)})$$

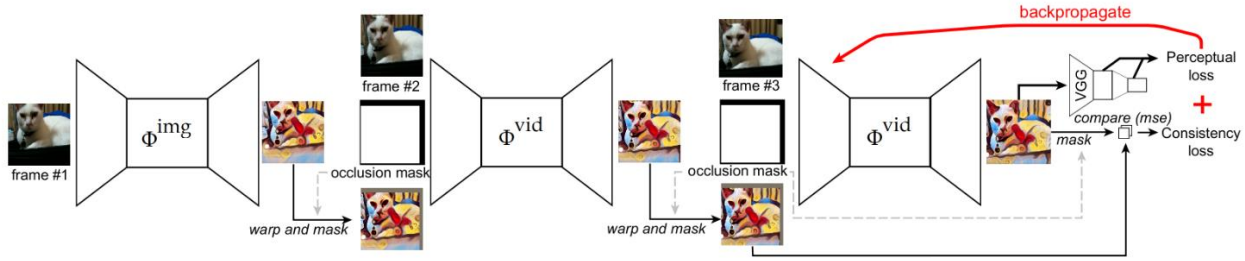


Fig. 42: Training procedure for our multi-frame approach, here shown with three frames.

When it comes to runtime, this method is computationally more expensive than Johnson *et al*'s network-optimization-based technique. The following benchmark has been conducted on a Nvidia Titan X GPU for a resolution of 1024x436 pixels. Although faster than image-optimization-based techniques, this model is too slow to produce real-time style transfer. Lowering the image quality would fasten the process, but if we suppose that this model is twice as slow as Johnson *et al*'s model, their benchmark in Fig. 26 suggests that a 512x512 pixel image would take 0.1 s to be computed, and 10 FPS is slightly too low for our use case.

Type	Method	alley_2	ambush_5	ambush_6	bandage_2	market_6	Runtime
Optimization	Random init	0.019	0.027	0.037	0.0180	0.023	540 s
Optimization	Prev frame init	0.010	0.018	0.028	0.0041	0.014	260 s
Optimization	Ours	**0.00061**	0.0062	0.012	**0.00084**	**0.0035**	180 s
Network	Per-frame	0.0062	0.011	0.016	0.0043	0.0089	0.2 s
Network	Ours	0.0016	**0.0042**	**0.0079**	0.0015	0.0039	0.4 s

Fig. 43: Short-term temporal consistency of video style transfer. We report average mean squared error over the test set (lower is better). Pixel values were between 0 and 1. Run time is reported in seconds per frame. We show the results of our optimization-based and network-based approaches, as well as three baselines: optimization-based stylization initialized with random noise or the previous frame, respectively, and independent per-frame network-based processing.

Here are qualitative results of this model. We can see that the temporal coherence is better in this model than in independent frame-per-frame models. Also, the authors compared the different training process and their combinations

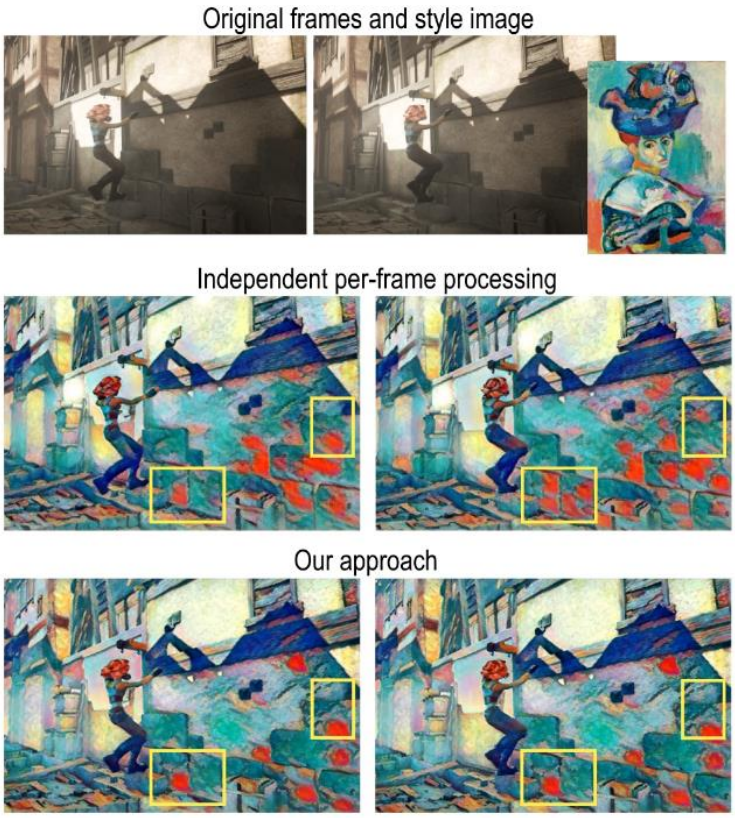


Fig. 44: Comparison of temporal consistency: Our method (multi-frame mixed) has less flickering than independent per-frame processing [12]. Rectangles indicate relevant differences. The corresponding video is available at `https://youtu.be/SKql5wkWz8E#t=0m42s`

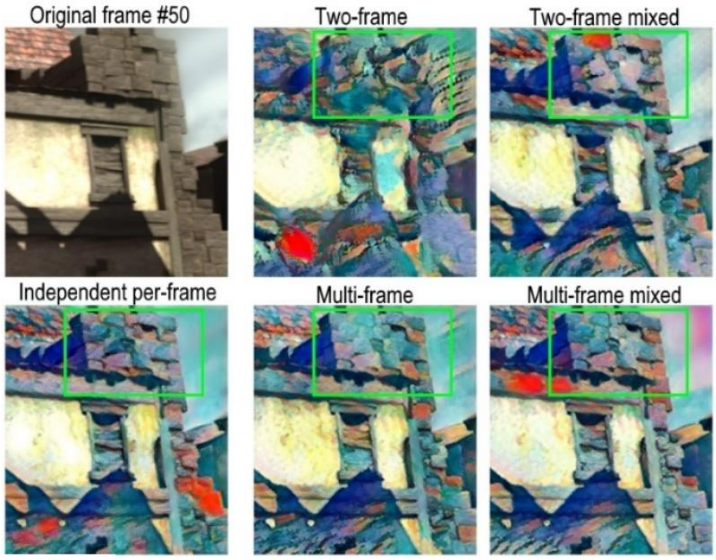


Fig. 45: Comparison of quality: Our advanced approaches retain visual quality and produce a result similar to [12] applied per-frame. The straightforward two-frame training, though, suffers from degeneration of quality. Rectangles indicate relevant differences. The corresponding video is available at `https://youtu.be/SKql5wkWz8E#t=2m47s`

Another architecture worth mentioning in real-time video style transfer is *ReCoNet: Real-time Coherent Video Style Transfer Network* proposed by Gao *et al*[43]. The main contributions of this paper are:

- the proposition of a luminance warping constraint to the temporal loss to reduce the negative impact of luminance variation from the input and stabilize output luminance;
- the proposition of a feature-map-level temporal loss;
- the creation of a feed-forward network that can achieve real-time and temporally coherent style transfer

The feed-forward network adopts a similar structure from Ruder *et al*'s feed forward network [40], inspired itsef by Johnson *et al*'s Image Transform Net [27], as described in the table below. However, it is explicitly separated in an encoder and a decoder. This allows the authors to extract feature before decoding and to apply a temporal loss at feature-map level.

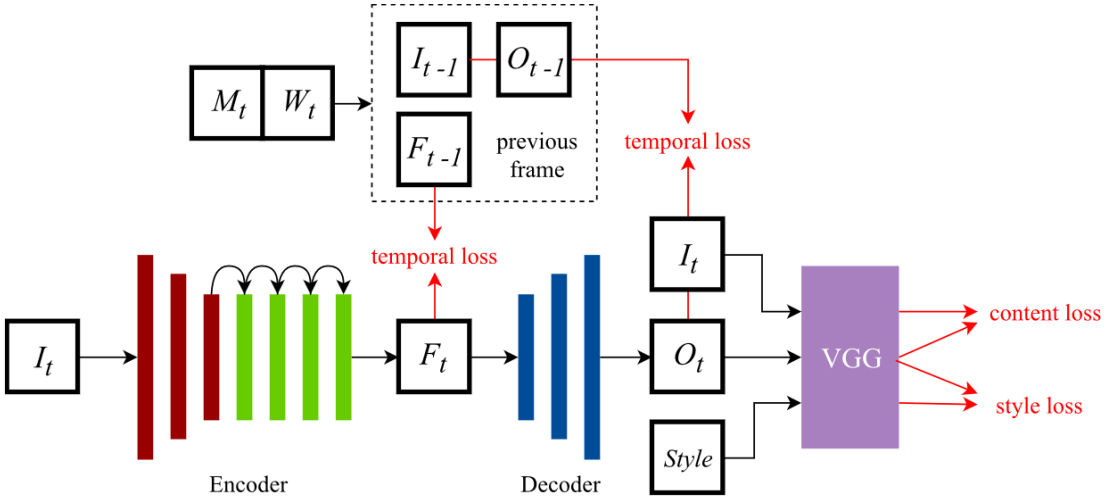


Fig. 46: The pipeline of ReCoNet. I_t, F_t, O_t denote the input image, encoded feature maps, and stylized output image at time frame t. M_t and W_t denote the occlusion mask and the optical flow between time frames $t-1$ and t. *Style* denotes the artistic style image. The dashed box represents the prediction results of the previous frame, which will only be used in the training process. Red arrows and texts denote loss functions

Layer	Layer Size	Stride	Output Size
Encoder			
Input			$3 \times 640 \times 360$
Conv + InsNorm + ReLU	$48 \times 9 \times 9$	1	$48 \times 640 \times 360$
Conv + InsNorm + ReLU	$96 \times 3 \times 3$	2	$96 \times 320 \times 180$
Conv + InsNorm + ReLU	$192 \times 3 \times 3$	2	$192 \times 160 \times 90$
(Res + InsNorm + ReLU) $\times 4$	$192 \times 3 \times 3$	1	$192 \times 160 \times 90$
Decoder			
Up-sample		1/2	$192 \times 320 \times 180$
Conv + InsNorm + ReLU	$96 \times 3 \times 3$	1	$96 \times 320 \times 180$
Up-sample		1/2	$96 \times 640 \times 360$
Conv + InsNorm + ReLU	$48 \times 3 \times 3$	1	$48 \times 640 \times 360$
Conv + Tanh	$3 \times 9 \times 9$	1	$3 \times 640 \times 360$

Fig. 47: Detailed architecture of ReCoNet

This model gives great results, both qualitatively and quantitatively. The output video seems temporally coherent and the output style matches the target style while the content is still perceivable, as visible in the comparison below (fig 48). Additional video material is available on YouTube here: https://www.youtube.com/watch?v=vhBRanZmdH0

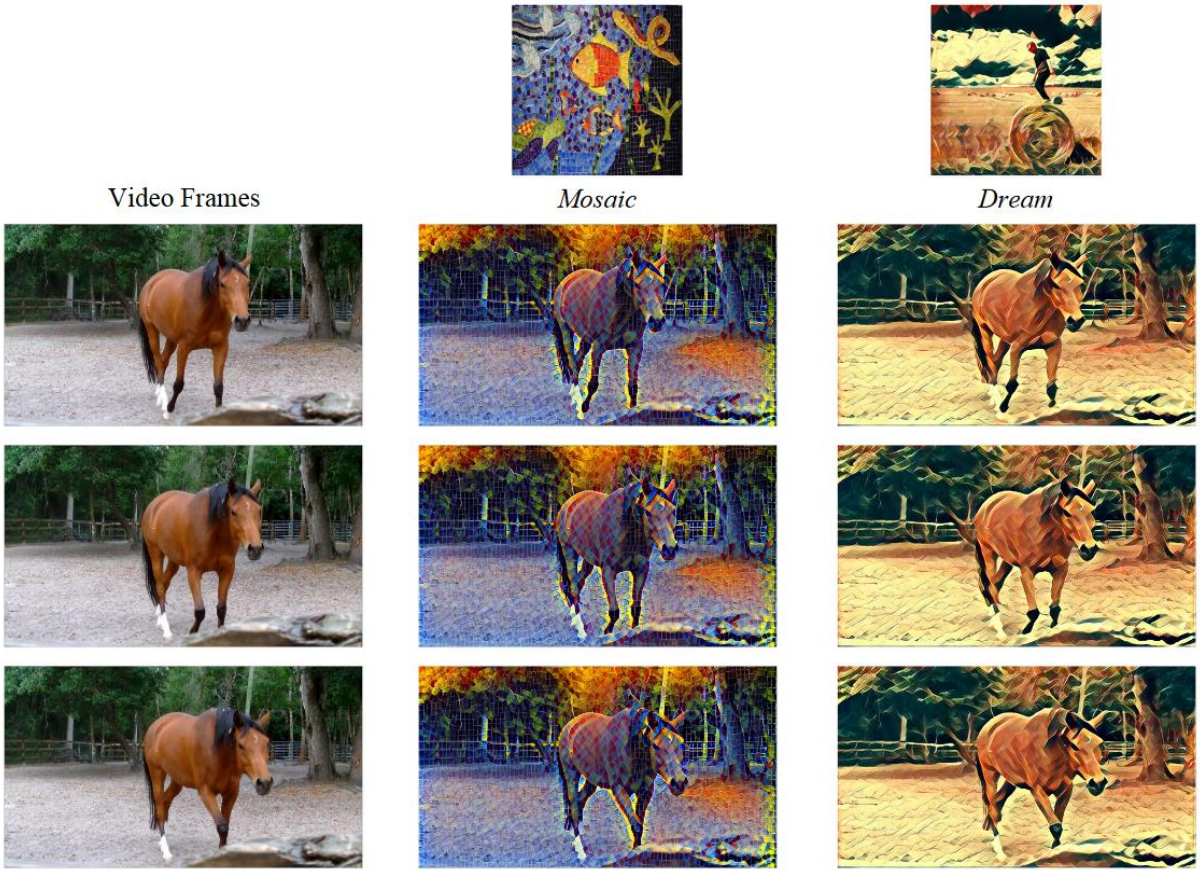


Fig. 48: Examples of coherent style transfer executed by ReCoNet for qualitative evaluation

Furthermore, this model is suitable for real-time. As exposed in the author's benchmark (fig 49), this model is capable to reach more than 200FPS on a modern GTX 1080 Ti GPU.

Model	Alley-2	Ambush-5	Bandage-2	Market-6	Temple-2	FPS
Chen *et al* [4]	0.0934	0.1352	0.0715	0.1030	0.1094	22.5
ReCoNet	0.0846	0.0819	0.0662	0.0862	0.0831	235.3
Huang *et al* [17]	0.0439	0.0675	0.0304	0.0553	0.0513	216.8
Ruder *et al* [27]	0.0252	0.0512	0.0195	0.0407	0.0361	0.8

Fig. 49: Temporal error and average FPS at inference stage with the same style on different models. Five unseen scenes from MPI Sintel Dataset are selected for validation.

III. Chosen model: Real Time Style Transfer for Videos, Huang *et al*, 2017

The ReCoNet architecture matches our criteria of style transfer quality, inference speed and temporal coherency and it could have been the chosen model for our use case.

However, as mentioned by the authors in fig 49, it is outperformed in terms of temporal error by another model proposed by H. Huang *et al* [0], with a temporal error on average 61% lower on this benchmark. It is also shown that both models reach a similar inference speed of more than 200FPS.

Although qualitatively different, the result of both models seems of good quality to me. On the example given by the authors of ReCoNet (which might be biased in favor of their model), Huang *et al*'s model seems smoother but ReCoNet's model seems to give more diverse an colorful results.

Fig. 50: Qualitative comparison of ReCoNet and Huang et al's model conducted by ReCoNet's authors on two given styles.

These two models are different, but both are based on the same papers. They both combine the works of Johnson *et al* [27] for the use and global design of the Image Transform Network (Downsampling – Residual blocks – Upsampling), Ruder *et al* [40] for the temporal loss, Ulyanov *et al* [31] for the use of Instance Normalization.

The majority of this model's characteristics have therefore already been described. It consists of a Stylizing network based on Johnson *et al*'s Image Transform Net, and a Loss Network (VGG19 pretrained on ImageNet classification task).

The Stylizing network is significantly smaller than the one used in ReCoNet (48 channels in the Residual blocks vs. 192 channels for ReCoNet). According to the authors, this allows a faster inference speed and a faster and better convergence, without impacting the quality of the output style. When comparing two models using 48 or 128 channels in the Residual Blocks (resp. 353-Res48 and 353-Res128), they found out that the temporal losses were quite similar, while 353-Res48 was more than twice as fast as 353-Res128 during inference time (see fig. 51)

Model	Alley_2	Ambush_5	Bandage_2	Market_6	Temple_2
353-Res128	0.0243	0.0408	0.0193	0.0330	0.0296
353-Res48	0.0244	0.0425	0.0195	0.0334	0.0302

Fig. 51: Temporal error with the same style on 353-Res48 and 353-Res128. Five unseen scenes from MPI Sintel Dataset are selected for validation.

The authors also show that the temporal coherency is learnt and encoded in the network directly, without the need for a long-term coherency loss as introduced by Ruder *et al* [40]

Another interesting factor is that compared to the method introduced by Ruder *et al*, this architecture does not require to compute optical flow during inference time which contributes to the increase in inference speed.

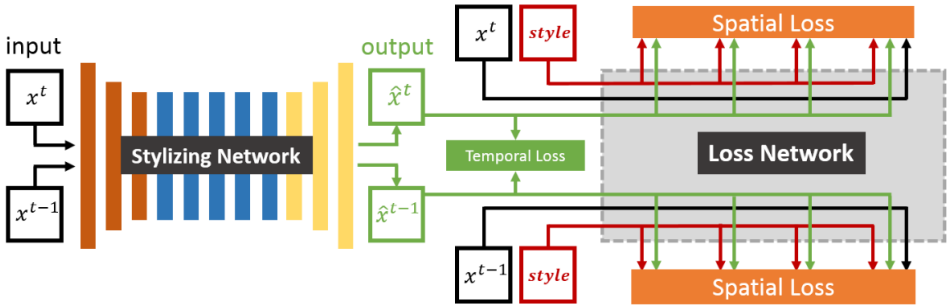


Fig 52. Overview of Hang et al's model.

	Layer	Size	Stride	Channel	Activation
Stylizing Network	Conv	3	1	16	ReLU
	Conv	3	2	32	ReLU
	Conv	3	2	48	ReLU
	Res × 5				
	Deconv	3	0.5	32	ReLU
	Deconv	3	0.5	16	ReLU
	Conv	3	1	3	Tanh
Res	Conv	3	1	48	ReLU
	Conv	3	1	48	

Fig 53. Detailed architecture of the Stylizing Network

IV. Implementation

4.1 Software-wise

My goal was to implement Huang *et al*'s model to reproduce their results. On top of this, I have made some modifications to get a better result.

4.1.1 Language and libraries

This project was done in Python, as it is one of the most commonly adopted language for machine learning and data science.

The main libraries I used were:

- PyTorch as my machine learning framework;
- Numpy, the traditional mathematic library;
- OpenCV 2, a computer vision library, for image manipulation and optical flow estimation;
- BeautifulSoup 4 to automatically scrap videos from videvo.net and easily get a large video dataset.

4.1.2 Training data

In order to get enough training data, I created a small web-scrapping tool that would automatically download free and royalty-free videos from the stock video website videvo.net. I gathered 500 video clips of 15s each, so approximately 180 000 frames in total. The videos were then converted to 640x360 pixels format to get uniform training data and reduce the dataset's weight.

See 5.2 for discussions on the training data quality

4.1.3 Upsampling layer

In their paper, Huang et al used Deconvolutional layers, also called Fractionally-strided Convolutional layer or Transposed Convolution. The first term is misleading, since there exists a different operation called Deconvolution, and that this is not the opposite of a regular convolution.

These layers have been widely used in CNNs when it was necessary to perform upsampling, as in Image Super-resolution tasks [27] or in Deep Convolutional Generative Adversarial Networks (DGAN) [44] to map latent space to higher-resolution images. However, it has been noticed that these layers were prone to generate checkerboard artifacts. As advised by Odena *et al* [42], I've replaced the Deconvolution layer (stride 2, kernel size 3, padding 1) by a 'nearest' Upsampling layer (upscaling factor 2) followed by a Convolutional layer (stride 1, kernel size 3, padding 1). This effectively led to an increase in the output's quality.

4.1.4 Optical Flow

In their paper, Huang et al used DeepFlow [46] to compute the optical flow, I used the Farneback [45] method. This choice was motivated by simplicity since there is an implementation of this method in OpenCV, that was therefore easily accessible to me. The result optical flow seems coherent and valid.

4.2 Hardware-wise

The training of the model was conducted on a Nvidia RTX 2080 GPU, thanks to the PRHLT Research Center from the UPV. The model was the evaluated either on this GPU (about 150 FPS) or on my personal computer's GPU, a Nvidia GeForce 940M (about 20 FPS)

The chosen webcam has a maximum frame rate of 30FPS, which does not limit the model's speed when the software is running on my PC's GPU, and a maximum definition of 1280x720 pixels, which is of a satisfying quality.

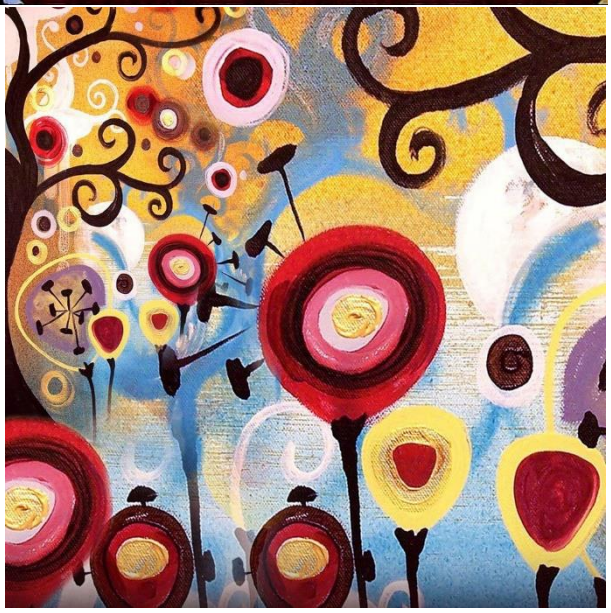
4.2 Results

This implementation is giving satisfactory results in terms of quality and diversity of the style (see fig.54). The parameters have been chosen to accentuate on the style more than on the content, because for this use case the fidelity of the style is more important than the accuracy of the content. However, the implementation of the temporal loss is not giving the expected result and the network is not yet capable of learning temporal coherency with the degree of precision expected (see fig. 55). This should be coming from the implementation more than the model, since Huang et al have demonstrated the capabilities of this architecture in terms of temporal coherency. Possible solutions for future improvement are exposed in 5.2.

Fig. 54: Style transfer performed by my implementation on Van Gogh's Starry Night.

Fig. 55: The network didn't learn to enforce temporal coherence as well as desired, some flickering effect remains.

Top row: frame 10. Middle row: frame 11.
Left column: input frame. Right column: stylized frame
Bottom image: style target



V. Conclusion

5.1 Conclusion of this project's goals

The final model is able to perform real-time video style transfer on the selected hardware. This proves the feasibility of such device.

However, the result lacks temporal coherence. This will be investigated in the next part of this work. As explained page 2, this is a first part of a work that started during my Erasmus Semester at the UPV as a *Trabajo de Fin de Master*, and that will be carried on during the next year at the *Ecole Centrale de Lille*.

5.2 Possible path for further improvement

To my knowledge, there is no architecture that allow *arbitrary* real-time coherent video style transfer. An interesting investigation direction would be to combine the proprieties of existing feed-forward networks that are capable of real-time arbitrary style transfer (Dumoulin *et al* [29], X. Huang *et al* [33]) with feed-forward networks that uses temporal losses to learn temporal coherence (Ruder *et al* [40], Gao *et al* [43], Huang *et al* [0])

On a more practical point of view, the temporal loss needs to be improved. There are several hypotheses that can explain the difference between my implementation and Huang et al's implementation. The first is that for simplicity reasons, I didn't used the same method to compute optical flows. Therefore, there might be a difference in the method that explain the gap in the temporal loss' behavior and the learning process. This might get into a cascade effect for the confidence mask computing, as a difference in the optical flow will trigger a difference in the computing of the disocclusion and motions boundary areas. Another hypothesis is that the network may need to get the exact same optical flow for a given pair of frames throughout the epochs. While Huang et al did computed all the optical flows once for all before the training process, I've chosen to compute the optical flows on the fly. These two differences between the author's implementation and mine might explain the difference in output quality.

Finally, the data quality could be improved. Although this made the data acquisition convenient and very time efficient, the quality of the data wasn't controlled. Some videos were very slow, comported very few movements, were too contrasted, blurred for esthetic reasons, or synthetic. This might have had a negative impact on the learning process if the content of the images was too far away from real life footage. It would be interesting to manually control every video of the input data to make sure that it corresponds to real-life footage with enough diversity, movements and colors.

VI. Bibliography

[0]: Huang, H., et al, Real-time neural style transfer for videos, http://openaccess.thecvf.com/content_cvpr_2017/papers/Huang_Real-Time_Neural_Style_CVPR_2017_paper.pdf.

[1]: The state of AI 2019, MMC Ventures, https://www.mmcventures.com/wp-content/uploads/2019/02/The-State-of-AI-2019-Divergence.pdf, visited on May 28, 2019.

[2]: Gardner, H., Frames of mind: the theory of multiple intelligences, 2011, http://www.pz.harvard.edu/resources/frames-of-mind-the-theory-of-multiple-intelligences, visited on May 28, 2019.

[3]: intelligence in Cambridge Dictionary Online https://dictionary.cambridge.org/dictionary/english/intelligence, visited on May 28, 2019.

[4]: Goleman, G., Emotional Intelligence: why it can matter more than IQ, Bantam Books, 2005, https://openlibrary.org/books/OL7826621M/Emotional_Intelligence, visited on May 28, 2019.

[5]: Copeland, B. J., Artificial intelligence in Encyclopaedia Britannica, Encyclopaedia Britannica inc., May 9, 2019, visited on July 9, 2019. https://www.britannica.com/technology/artificial-intelligence

[6]: artificial intelligence in Cambridge Dictionary Online https://dictionary.cambridge.org/dictionary/english/artificial-intelligence, visited on May 28, 2019.

[7]: Mijwel, M. M., History of Artificial Intelligence, 2015, https://www.researchgate.net/publication/322234922_History_of_Artificial_Intelligence, visited on May 29, 2019.

[8]: Buchanan, B., Shortliffe, E., Rule-based expert systems: the mycin experiments of the Stanford heuristic programming project, Addison-Wesley Publishing Company, n.d., http://www.aaai.org/Papers/Buchanan/Buchanan45.pdf.

[9]: machine-learning in Oxford Learner's Dictionaries Online, https://www.oxfordlearnersdictionaries.com/definition/english/machine-learning, visited on May 29, 2019.

[10]: Litwinowicz, P., Processing images and video for an impressionist effect in SIGGRAPHH '97 proceedings of the 24th annual conference on Computer graphics and interactive techniques, pp. 407-414, ACM Press/Addison-Wesley Publishing Company, New-York, 1997, https://dl.acm.org/citation.cfm?doid=258734.258893, visited on May 29, 2019.

[11]: Gatys, L., Ecker, A., Bethge, M., A neural algorithm of artistic style, 2015, https://arxiv.org/pdf/1508.06576.pdf.

[12]: Olah, C., Mordvinstev, A., Schubert, L., Feature visualization: how neural networks build up their understanding of images, Google inc., 2017, https://distill.pub/2017/feature-visualization/, visited on May 30, 2019. [13]: Park, E., et al, Large scale visual recognition challenge (ILSVRC) 2017, http://www.image-net.org/challenges/talks_2017/ILSVRC2017_overview.pdf.

[14]: Krizhevsky, A ., Sutskerver, I., Hinton, G., ImageNet classification with deep convolutional neural networks, n.d., https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.

[15]: Huang, T. S., Computer vision: evolution and promise, n.d., https://cds.cern.ch/record/400313/files/p21.pdf, visited on May 30, 2019.

[16]: Hubel, D. H., Wiesel, T.N., Receptive fields of single neurones in the cat's striate cortex in The journal of Physiology, October 1959, pp. 574-591, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1363130/, visited on May 30, 2019.

[17]: Marr, D., Vision: a computational investigation into the human representation and processing of visual information, MIT Press, 2010, https://direct.mit.edu/books/book/3299/visiona-computational-investigation-into-the-human, visited on May 30, 2019.

[18]: Fukushima, K., Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, K. Biol. Cybernetics, Springer-Verlag, 1980, https://link.springer.com/article/10.1007%2FBF00344251, visited on May 30, 2019.

[19]: LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., Gradient-based learning applied to document recognition, proc. of the IEEE, November 1998, http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf visited on May 30, 2019.

[20]: LeCun, Y., et al, Backpropagation applied to handwritten zip code recognition in Neural Computation 1, pp. 541-551, MIT, 1989

https://www.ics.uci.edu/~welling/teaching/273ASpring09/lecun-89e.pdf visited on May 31, 2019.

[21]: Cohen, J. P., Visualizing CNN architectures side by side with mxnet, 2016, https://josephpcohen.com/w/visualizing-cnn-architectures-side-by-side-with-mxnet/ visited on May 31, 2019.

[22]: Dumoulin, V., Visin, F., A guide to convolution arithmetic for deep learning, 2018, https://arxiv.org/pdf/1603.07285.pdf.

[23]: Deshpande, M., Introduction to convolutional neural networks for vision tasks, n.d., https://pythonmachinelearning.pro/introduction-to-convolutional-neural-networks-for-vision-tasks visited on May 31, 2019. [24]: Luan, F., et al, Deep photo style transfer, 2017 https://arxiv.org/pdf/1703.07511.pdf.

[25]: Li, C., Wand, M., Combining Markov random fields and convolutional neural networks for image synthesis, 2016, https://arxiv.org/pdf/1601.04589.pdf.

[26]: Champandard, A., Semantic style transfer and turning two-bit doodles into fine artwork, nucl.ai Conference 2016 Artificial intelligence in creative industries, July 18-20, Vienna/Austria, 2016 https://arxiv.org/pdf/1603.01768.pdf.

[27]: Johnson, J., Alahi, A., Fei-Fei, L., Perceptual losses for real-time style transfer and super-resolution, Stanford University, 2016 https://arxiv.org/pdf/1603.08155.pdf.

[28]: Radford, A., Metz, L., Chintala, S., Unsupervised representation learning with deep convolutional generative adversarial networks, 2017 https://arxiv.org/pdf/1511.06434.pdf.

[29]: Dumoulin, V., Shlens, J., Kudlur, M., A learned representation for artistic style, Google inc., 2017 https://arxiv.org/pdf/1610.07629.pdf.

[30]: Ulyanov, D., et al, Texture networks: feed-forward synthesis of textures and stylized images, 2016, https://arxiv.org/pdf/1603.03417.pdf.

[31]: Ulyanov, D., Vedaldi, A., Lempitsky, V., Improved texture networks: maximizing quality and diversity in feed-forward stylization and texture synthesis, 2017, https://arxiv.org/pdf/1701.02096.pdf.

[32]: Ulyanov, D., Vedaldi, A., Lempitsky, V., Instance normalization: the missing ingredient for fast stylization, 2017, https://arxiv.org/pdf/1607.08022.pdf.

[33]: Huang, X., Belongie, S., Arbitrary style transfer in real-time with adaptive instance normalization, Cornell University, 2017 https://arxiv.org/pdf/1703.06868.pdf.

[34] : Ba, J. L., Kiros, J. R., Hinton, G. E., Layer normalization, 2016 https://arxiv.org/pdf/1607.06450.pdf.

[35] :Li, Y., et al, Demystifying neural style transfer, 2017 https://arxiv.org/pdf/1701.01036.pdf.

[36]: Fleet, D., Weiss, Y., Optical flow estimation, n.d., http://www.cs.toronto.edu/~fleet/research/Papers/flowChapter05.pdf visited on June 1, 2019.

[37]: Weinzaepfel, P., Revaud, J., Harchaoui, Z., Schmid, C., DeepFlow: Large displacement optical flow with deep matching in ICCV 2013 - IEEE International Conference on Computer Vision. pp. 1385–1392. IEEE, Sydney, Australia (Dec 2013), https://hal.inria.fr/hal-00873592, visited on June 3, 2019.

[38] : Revaud, J., Weinzaepfel, P., Harchaoui, Z., Schmid, C., EpicFlow: Edge-Preserving Interpolation of Correspondences for Optical Flow in CVPR 2015 - IEEE Conference on Computer Vision & Pattern Recognition, Boston, United States (Jun2015), https://hal.inria.fr/hal-01142656, visited on June 3, 2019

[39]: Sundaram, N., Brox, T., Keutzer, K., Dense point trajectories by GPU-accelerated large displacement optical flow, Berkeley University, 2010 https://www2.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-104.pdf visited on June 3, 2019.

[40]: Ruder, M., Dosovitskiy, A., Brow, T., Artistic style transfer for videos and spherical images, 2018 https://arxiv.org/pdf/1708.04538.pdf.

[41]: Ruder, M., Dosovitskiy, A., Brow, T., Artistic style transfer for videos, University of Freiburg, 2016 https://arxiv.org/pdf/1604.08610.pdf.

[42]: Odena, A., et al., Deconvolution and Checkerboard Artifacts, Distill, 2016 http://doi.org/10.23915/distill.00003, visited on June 3, 2019.

[43] : Gao, C., et al, ReCoNet: real-time coherent video style transfer network, University of Hong-Kong, 2018 https://arxiv.org/pdf/1807.01197.pdf.

[44]: Radford, A., Metz, L., Chintala, S., unsupervised representation learning with deep convolutional generative adversarial networks, 2016 https://arxiv.org/pdf/1511.06434.pdf.

[45]: Farnebäck, G., Two-frame motion estimation based on polynomial expansion, Springer, Berlin, 2003 https://link.springer.com/chapter/10.1007%2F3-540-45103-X_50 visited on June 4, 2019.

[46]: Weinzaepfel, P., Revaud, J., Harchaoui, Z., Schmid, C., DeepFlow: large displacement ptical flow with deep matching, ICCV-IEEE International conference on computer vision, Dec 2013, Sydney, Australia, pp. 1385-1392 https://hal.inria.fr/hal-00873592/document/ visited on June 4, 2019.

[47]: He, K., Zhang, X., Ren, S., Sun, J., Deep residual learning for image recognition, Microsoft Research, 2015 https://arxiv.org/pdf/1512.03385.pdf.