# Converting night-time images to day-time images through a deep learning approach

Nicola Capece, Ugo Erra, Raffaele Scolamiero
Università della Basilicata, Dipartimento di Matematica, Informatica ed Economia
nicola.capece@unibas.it, ugo.erra@unibas.it, raffaele.scolamiero@gmail.com

***Abstract*—This paper examines the application of a deep learning approach to converting night-time images to day-time images. In particular, we show that a convolutional neural network enables the simulation of artificial and ambient light on images. In this paper, we illustrate the design of the deep neural network and some preliminary results on a real indoor environment and two virtual environments rendered with a 3D graphics engine. The experimental results are encouraging and confirm that a convolutional neural network is an interesting approach in the fields of photo-editing and digital image post-processing.**

***Keywords*-Image processing; Deep learning; Neural Networks; Image Filtering;**

## I. Introduction

Today, artificial intelligence (AI) is used in several information technology fields. Its aim is to recognize and make an intelligent decision based on data provided as input. There are several applications that use AI, such as self-driving cars [1], speech recognition [2], antispam filters [3], image recognition [4] [5]. Deep learning (DL) is getting closer to the concept of AI thanks to the evolution of GPU architectures and parallel computing. DL is often used in computer vision and image recognition. The aim of DL is to develop a computational model that consists of multiple levels of a process used to learn representations of data at multiple levels of abstraction [6]. DL uses complex structures, huge datasets and backpropagation algorithms to modify its internal parameters in order to improve the result of predictions. These parameters, which are called weights, are used to represent data from the previous level to the next one.

Any discussion of DL also indirectly refers to neural networks (NN), also called deep neural networks (DNN). A DNN is a neural network composed of several depth layers. A special type of DNN often used in image processing is called a convolutional neural network (CNN) and is structured to take advantage of the use of multidimensional data such as images. In general, a NN consists of several layers, and each one of them is composed of several neurons. Each neuron is connected to all the neurons of the subsequent layers through arches called synapses, defining a fully connected neural network (FcNN). Layers in a CNN are composed of data volumes called tensors. In this type of NN, each neuron is not connected to all neurons of the next layer as for FcNN, but only to a small group called the receptive field. In this case, neurons of a layer are connected to neurons of the subsequent layer with local connectivity.

In general, a NN analyzes the input and recognizes and classifies it using an automatic learning algorithm. There are several categories of this algorithm, each one focused on a different type of training. The most important two are: *(i)* Supervised learning, where the training data are composed of pairs of input and output data. For each input, the corresponding output is known and serves as a valuable example in the training process. After training, the system learns the hidden relationship that binds input to the output variable and is potentially able to make a prediction even where the output is not known a priori. *(ii)* Unsupervised learning, where the training data are composed of only input values. In this case, the task is performed through a search for hidden patterns inside the training data. In general, this type of learning is used when establishing the data's classes is a complex operation and especially when the nature of the data is not completely defined.

In this paper, we propose an implementation of a supervised learning algorithm that through a DNN is able to process an image and simulate a filter that applies artificial or natural lighting. The idea is to transform an image or a scene with a low ambient light into a fully illuminated image as if it has received light from a lamp or the sun. The training dataset consists of pairs of input–output values, where the input is composed of a set of images with low light, and to each of which is linked the same images but with greater brightness. The goal of our work is to be able to transform an image obtained in night-time ambient light into a day-time image.

The remainder of this paper is structured as follows. Section II provides an overview of related works. Section III provides some background information about the NN and software tools used to perform training. Section IV is a detailed presentation of the NN architecture and our approach. Section V describes the test that we performed and the obtained experimental results. We end with some final remarks and future directions for our research in Section VI.

## II. Related Work

Zezhou et al. [7] developed a method for colorizing images from a grayscale input image. The problem was formulated as a regression problem, and a DNN was used to solve it. They used a huge training database, which contains

2375-0138/17 $31.00 © 2017 IEEE
DOI 10.1109/iV.2017.16

several types of images such as trees, animals, buildings, sea, and mountains. The NN can represent an approximation system of continuous complex function [8]. Given a function $A = (G, C)$, where $G$ represents the grayscale images and $C$ represents the corresponding colorized images, their method is based on the statement that there exists a gray-to-color complex function called $F$, which can map the extracted features from each pixel of $G$ to the respective chrominance value using $YUV$ color space. The NN used has a number of input neurons equal to the size of descriptive features extracted from each single pixel in the grayscale image and two neurons in the output layer, the output of which are the $U$ and $V$ channels representing the color value. The activation function used in this work is a rectified linear unit (ReLU) [4], which allows a fast convergence of the training process. The algorithm used in the training phase is a classical backpropagation error method that changes weighted values on the synapses that connect neurons.

J. Lee and S. Lee [9] developed a system that allows image illumination to be applied in order to simulate that the same picture has been taken repeatedly in different hours of the day. They used an approach based on the adversarial network [10] to define a CNN as a function that changes the input image color to the one it should have in the selected hour of the day. A function is trained separating the sample's day-time and night-time images. Because their approach is based on the adversarial network, they used two CNNs, called generator and discriminator: the generator NN creates a fake night-time image, and the discriminator NN classifies if its input is fake or real. The first NN is composed of five convolutional layers; the second one is composed of six convolutional layers. They used features such as pooling, ReLU, and dropout layers. The final layers placed at the end of the discriminator are fully connected; the activation function for each neuron is a sigmoid.

T. Shih et al. [11] developed a system that creates an image in a particular hour of the day using a single input image. Their approach is data-driven and allows the automatic creation of a plausible looking picture. This approach is based on the use of a database of time-lapse movies of several scenes. These movies provide information on the color variation of scenes during the day. This method transfers color from movies with similar scenes to the input picture. The movies used covered several outdoor scenes such as a cityscape, buildings, and street views. The transfer performs a matching of the input image with time-lapse movies that contain a similar image and uses a Markov random field-based algorithm to find a match. Subsequently, they used an example based on locally affine model that transfers the local variation color between two frames of the movie in two different times of the input image.

The main difference among these methods and our work is that we consider only two classes of images: day-time and night-time. We define the problem as a regression problem in which the model has been trained using as the input the image to which has been applied artificial illumination and using as the output the same image but with the desired real-world illumination. The key aspect of this work is the perfect matching between image content representing input and output in such a way that the NN was trained only to change the colors scale of the images and not to translate or deform them.

## III. Background

The concept of AI is analogous to the human brain. A biological neuron is mainly composed of dendrites, synapses, and axons. Dendrites are the links that initiate the nerve signal to other parts of the human body, synapses connect each neuron to other neurons, and axons convey signals out of the neurons. Artificial neurons (AN) have a similar structure. In fact, they have a weighted link that takes data input to the inside of the neuron and several links that allow processed data to be conveyed from the same neuron to more internal neurons. AN are organized into layers in order to create a directed and weighted graph. Generally, a simple NN is composed of three layers, called the input, hidden, and output layers. Each neuron of each layer is connected with all those of the next layer. This approach creates a NN type called a feed-forward neural network because the data flows from left to right in the graph without forming cycles inside it. The algorithm commonly used to train this type of NN is called backpropagation, which computes a loss function given from the distance between the expected values and those obtained from the NN in a single training step, called epoch. This function is minimized through an algorithm called gradient descent, which updates the weights in the opposite direction to the gradient of the loss function.

In image processing, the NN often used is an already mentioned particular type of FcNN, the CNN. In this type of NN, the neurons are represented by the pixels of the image provided as input. In this case, a layer is composed of the data's volumes represented through the tensors. These volumes change shape as a result of the convolution operation performed during the training phase. Generally, the volume's shape reduces its height and width and increases its depth after a convolution operation. Each deep layer of the volume represents a set of components called a feature map, which has a different set of weights with respect to the other, but on a single feature map these values will be the same and will be shared along the $x$ and $y$ axes. These parameters are also called filters or kernels. The sharing of weights and local connectivity we mentioned previously explain why the CNN performs better on tasks related to the image processing and computer vision. The convolutional operation can be compared to a real scan of an image in order to extract relevant features. In order to achieve our goal, we built a NN from a structure composed of 16 convolutional layers, of which 13 were convolutional and 3

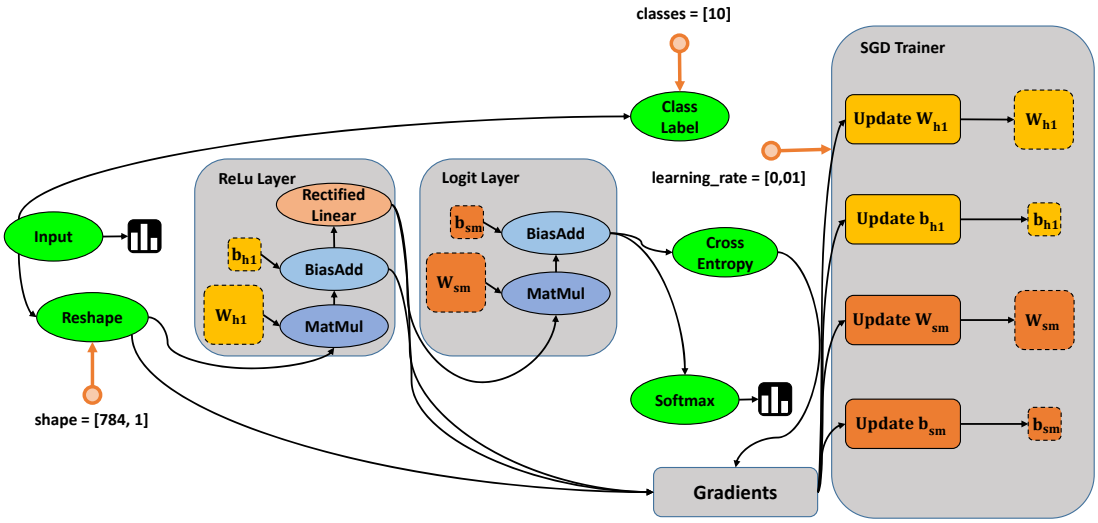


Figure 1. Diagram and graphic representation of a generic neural network through the TensorFlow framework. The rectangles represent the variables and arrows the tensors.

were fully connected, developed by the VGG-16team [12] as part of the ILSVRC-2014 (ImageNet Large-Scale Visual Recognition Challenge) competition. The first part performs a classification process, and the second part is formed by a structure that is responsible for the decoding of the image. It uses an approach based on deep residual learning (DRL) [13].

Today, GPU computing leverages parallel architecture to perform mathematically compute intensive operations that a CPU is not designed to handle well [14]. In particular, GPUs definitely can be used to speed up NN training, and there are several frameworks and libraries for training and using DNNs that exploit computational GPU acceleration, such as Caffe, Torch7, and Teano. In our work, we used TensorFlow. This library is written in C++ and CUDA and provided APIs for both Python and C++. In particular, we used the Python programming language. In order to use TensorFlow, there are two phases: $(i)$ a first phase, called the construction phase, regarding the creation of the graph which defines the training model and the task that the NN has to perform; $(ii)$ a second phase, called the runtime phase, in which are performed the operations defined in the previous phase. Figure 1 shows a simple graph of FcNN classification. The elliptical icons represent mathematical operations which the tensors are subjected to; the rectangular icons represent the variables in the model; and the arrows represent the tensors and their values that cross the graph in calculations. The bottom of Figure 1 shows the input tensor, which represents an image; subsequently, there is a reshape operation that transforms the image into a vector. The larger rectangles represent the NN's layers where is defined an activation function, a ReLU in this case. The final part of the graph represents the computing of the loss function through the mean square error, given from the prediction of the NN and the expected data. Subsequently, the gradient is computed through a chosen optimizer, in this case, stochastic gradient descent [15]. The last layer of the graph is responsible for updating the weights using information provided by the gradient. This operation is repeated until the error is not sufficiently small, or it reaches a number of iterations defined a priori.

## IV. Neural Network Architecture

As mentioned above, our goal is to simulate natural or artificial illumination using DL with a supervised learning paradigm. The input to the NN is an image with low brightness, and the output should be the same image but with a sufficiently high illumination to ensure a consistent view of the image. We developed a NN that consists of two main parts. The first part is composed of the NN called VGG-16 [12], which identifies a pattern from the image and then extracts the features in order to create a sufficiently complex function to approximate the lighting of the image itself.

As shown in Figure 2, the first operation that is performed is the import of VGG-16, composed of 16 layers of which we consider convolutional only layers, divided into 5 groups. The first two are composed of two convolutional layers and the three remaining layers from three fully connected layer. Among these groups is performed an operation called maxpool, which reduces the image size, represented by a tensor, through a non-linear operation of downsampling in order to adjust the tensors as an input of the next layer. As the image is represented using tensors, the output of each convolution operation is a tensor, called the activation tensor, to which is applied a normalization, called batchnorm, before

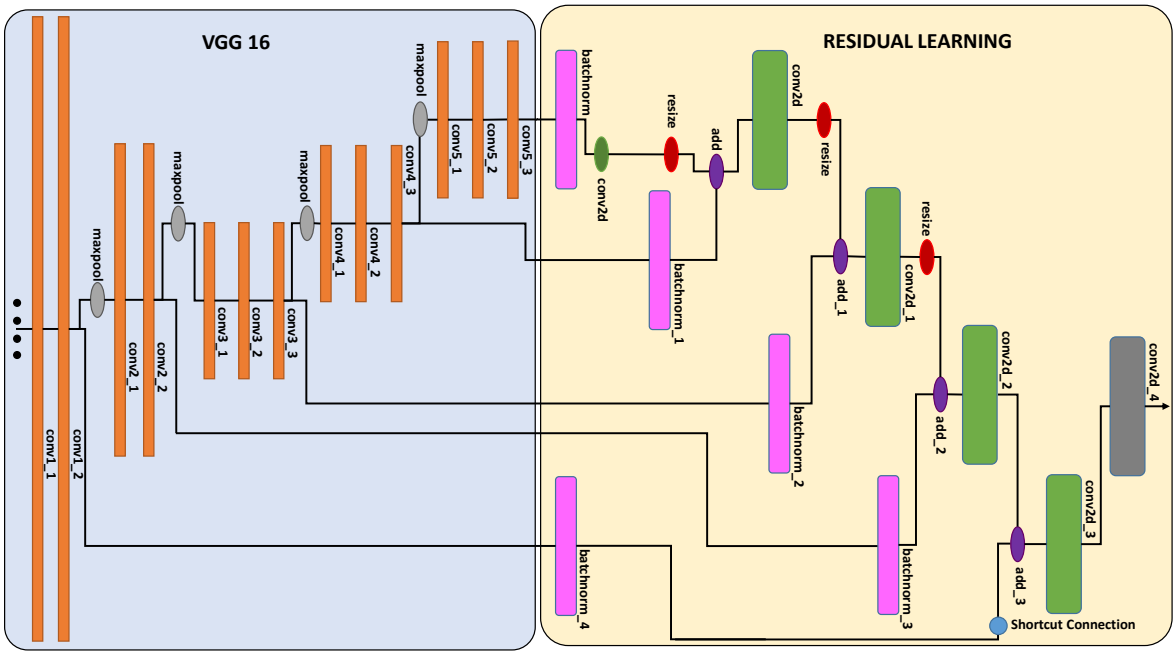


Figure 2. Neural Network Architecture. The left-hand side shows only 13 convolutional layers of the VGG-16. The final three fully connected layers of the original VGG-16 were not used in our approach. The right-hand side shows our contribution through the residual learning network.

passing as input to the next activation function of the next layer. The activation function used in this phase is the ReLU [16]. Changes in the parameters of the NN during training change the distribution of the outputs of each layer, so subsequent layers have to adapt to these changes during the training. In order to solve this problem, Ioffe et al. [17] proposed the use of the batch normalization, which assures that the input of the activation function has mean 0 and variance equal to 1:

$$BN(x_i) = \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta$$

where $\mu_B$ represents the mean and denominator is standard deviation. The smaller constant $\epsilon$ is added to the variance in order to avoid dividing by zero. Here, $\gamma$ and $\beta$ are learnable parameters. To the normalized tensor is applied convolution, upscale, and subsequently addition with the previous layer's output tensor of the VGG-16, element by element. This way the VGG-16 is crossed in reverse order using the tensor extracted, proceeding up to the input tensor through the connections, called shortcut, which avoids the problem of degradation that affects NNs composed of several contiguous layers [13]. After the last two convolutions in the residual learning, we get an RGB image with simulated lighting. The VGG-16's tensor is used twice during the forward phase in the NN: The first time is used to compute the activation values of convolutional layers of the first part of the NN; the second time is used to decode the image and then to arrive at the output tensor.

The second part of our NN is composed of a so-called residual learning network (RLN). It is known that the number of features extracted from a CNN increases in proportion to the number of layers of which it is composed. He et al. [13] in a Microsoft research study demonstrated that in this case a decrease in accuracy and an increase in error are noticeable both in the training and in the test phase. In more specific terms, considering the same dataset and iterations, a NN with fewer layers achieves better results than a NN with more layers, due to the degradation problem. The solution to this problem is to allow the NN to learn a residual function. Where *H(x)* is the function that a group of layers need to learn, and if we transform the problem in a way that the function to approximate is *F(x)* = *H(x)* + *x*, then we will have a residual function and the problem will be in the form *H(x)* = *F(x)* + *x*, where *F(x)* represents the residual function and $x$ represents the input of this function. The contiguous layers learn this residual function, and subsequently, the input of these layers is added to the output of themselves. In this way, the degradation problem is significantly attenuated. The approximation of the residual function *F(x)* is performed through the use of two or more layers. As the sum of the residual function and the input is performed element by element, these two dimensions need to be adapted. This operation is performed through bilinear interpolation [18], which is performed on the height and width dimensions. In order to obtain the same dimensions, even depth, we performed a convolutional operation with a

$[1\times1]$ kernel and subsequently $[3\times3]$ kernel. The task of the bilinear interpolation is to move the pixel values, based on a parameter of scale, which defines how much the feature map must be enlarged or reduced. In this way it is possible to obtain empty pixels. To these pixels must be assigned a value through an interpolation based on four known values around the considered pixel [18]. The extracted tensor from the last VGG-16's convolutional layer has a $512$ features map. Our RLN form this tensor perform a normalization through a $[1\times1]$ convolutional operation that reduces the features map number to $256$ to be able to sum this tensor and the previous layer's output tensor. This operation is performed after a bilinear upsampling operation that fits the width and height dimensions of the tensor. The VGG-16's output tensor is an input of the RLN through a convolution operation using the $[3\times3]$ kernel, $1$ as stride and SAME as padding, which adds to the input convolutional tensor a border of zeros so that the output feature map has the same dimensions. As it continues with the next convolutions, a bilinear resize operation is performed in order to double the feature map's width and height dimensions, and subsequently, this feature map is added with the output tensor of the VGG-16's convolutional layer after applying a batch normalization. Finally, a further convolution $[3\times3]$ and batch normalization are performed. The last two convolution operations return a tensor which has only three feature maps that represent the RGB channels in addition to the width and height that characterize the output image. At the end of the last convolution, a sigmoid activation function is used for normalizing the output values between $0$ and $1$. The training algorithm of our NN is the stochastic gradient descent [15]. The loss function is computed using the following formula:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - y_{o,i})^2$$

where $y$ is the known output, and $y_{o,i}$ is the predicted result of the NN. In our case, the loss function is not computed on all elements of the training dataset but on each input–output pair that is forwarded to the NN. This involves updating the weights at each iteration instead of each epoch as in the classic gradient descent algorithm. We also used a learning-rate equal to 0.0001 because this gave us the best results. As our NN consists of two parts, the weight values were initialized in two different ways: for VGG-16 we used a model pre-trained on $1.5$ million images in such a way to exploit the NN features. Because the weights are those obtained from the pre-trained model, this operation is called fine-tuning the NN. The RLN weights were initialized using a continuous distribution of probability called the truncated normal distribution [19]. This approach can avoid the gradient being amplified or completely dissolved, which can lead to critical errors during learning. In addition, the batch normalization operation used, which is done after every convolution operation, reduces the probability that a bad initialization of weights leads to a non-optimal convergence.

## V. Experimental Results

For the purpose of testing the performance and to predict the results of our NN, we performed several experiments. In order to obtain accurate predictions, we defined three domains of the application: two indoor environments and an outdoor environment. For each domain, we trained the NN defining a regression model able to provide a correct prediction of images belonging to the same domain. The first domain is represented by a real indoor environment whose dataset is composed of $523$ pairs of input–output images, captured using a camera placed on a tripod in order to ensure a perfect matching between the images. The images of the dataset were taken at a resolution of $6000\times4000$ and then resized to $224\times224$, which is the size accepted by our NN as the input. The low-lit images were obtained through the occlusion of all possible sources of light, and the high-lit image was obtained at the same angle but using the camera's flash. The flash sufficiently illuminated the small room we used. The use of low resolution for the input and output images of the NN was in order to reduce NN complexity and training time. We divided our datasets into two subsets: the first one, called the training set, was used to update the parameters of the NN; the second one, called the validation set, was used to test the level of generalization of the NN. Subsequently, we created an additional dataset called the test set that contained only low-lit images. This dataset was used to test the accuracy of the prediction of the NN. Its aim was to apply the light to an image with a low brightness whose corresponding correct image brightness, taken from the same domain of the other two datasets, was unknown. Our first dataset consisted of $473$ images for the training set and $50$ for the validation set. In this dataset, we added further images, using a technique called data augmentation, which creates new images by applying artificial noise to the saturation and contrast of the dataset's images, bringing the number of images to about $34000$. In the training phase, we used a system of dynamic change of the input of the NN. Specifically, each iteration is characterized by a training operation, and each two iterations are computed a validation operation. In the first case, loss-training error is computed using the training set of images, and the later loss-validation error is computed using an image belonging to the validation set. For this first dataset, the training time lasted around six days running about $6.6$ million iterations and showing a good grade of convergence.

Figure 3 shows the result of an inference of an image unknown to the NN after just $1000$ iterations. Figure 4 shows the result of two inferences on the same test image, which is shown to the far left of the figure. The first inference is represented by the central image in the figure and is the result obtained after about $1.5$ million iterations.

Figure 3. Example of real environment: the left-hand image is the input; the middle image is the prediction of the neural network; the right-hand image is the ground truth image.

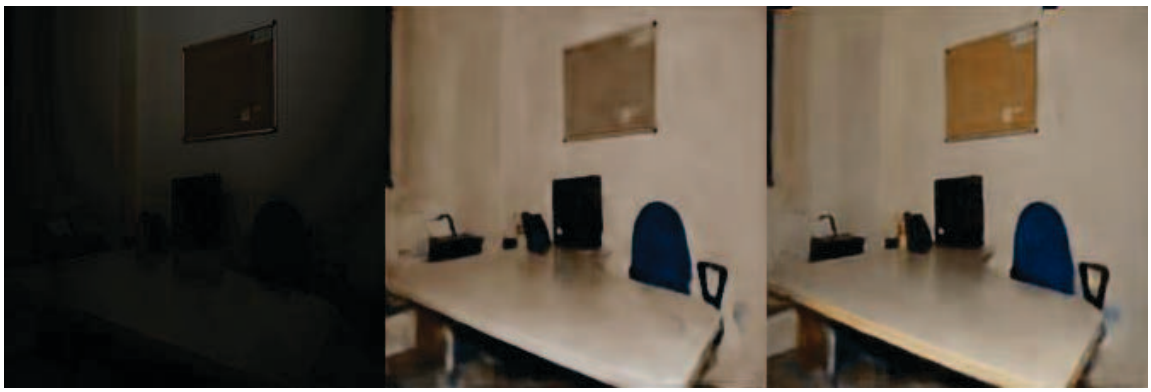

Figure 4. The left-hand image is the input to the neural network; the middle image is the prediction after 1000 iterations of the training process; the right-hand image is the prediction after 6.6 million iterations.

The second inference is represented as the far right image in the figure after about 6.6 million iterations; in which a much better improvement was demonstrated through an error value equal to 0.0005. The second domain was a virtual environment created with Unreal Engine 4 (UE4). The virtual scene represents a landscape composed of trees and plants. We obtained a dataset consisting of 3894 input–output pairs, of which 3834 were used for the training set and 60 were used for the validation set. Data augmentation was applied, leading to an increase in the number of images to about 276000. The images were obtained through a video in which a camera was moved along a default path within the virtual scene both in night-time and in day-time. In this experiment, we needed about 0.6 million iterations to obtain good results. We extended the training until we reached 1.2 million iterations, but in this case, we found that the NN was overfitting because the results of these inferences were worse.

In the middle of Figure 5, we show the results of inference after 0.6 million iterations with an error of 0.0004. The final domain was a virtual environment, also created with UE4. The scene, in this case, is an indoor environment, and the images were taken in the same outdoor mode. From the video, 2028 images were extracted, of which 1998 were used for the training set and 28 for the validation set. With data augmentation, these were increased to 143000 images. In this case, we obtained a good convergence to 0.12 million iterations. Figure 6 shows the result of inference after about 0.1 million operations. In order to improve the performance of the training operation, we exploited parallel computing allowed from the GPU during both reading and pre-processing of images from the disk during training, through a function in TensorFlow. The tests were performed on a machine equipped with a NVidia Tesla K40 GPU, which allows optimal performance in terms of computation time.

## VI. Conclusions

Our work shows that through a DNN a simulation can be created of artificial and ambient light on images. The main problem of our approach is the creation of a good dataset with a high amount of images that allows the NN to achieve optimal results in generic environments. A supervised DL algorithm generally achieves acceptable performance with

Figure 5. Results from the Unreal Engine 4 outdoor scene dataset.

Figure 6. Results from the Unreal Engine 4 indoor scene dataset.

around 5000 labeled examples per category and matches human performance when trained with a dataset containing at least 10 million labeled examples [20]. In our approach, we used datasets limited to isolated application contexts because, at the current state of art, the use of heterogeneous environment does not lead to good results. Possible applications of this approach are in the field of photo-editing or digital post-processing of images taken in a low-light environment. Our goal was to implement a suitable filter that can be applied to these images. This is a work in progress, and we intend to further investigate this NN with a sufficiently large dataset, allowing us to perform predictions regarding much larger and generic domains than those presented in this paper.

## ACKNOWLEDGEMENTS

The authors thank NVIDIA's Academic Research Team for providing the Tesla K40c card under the Hardware Donation Program.

## REFERENCES

[1] S. L. Poczter and L. M. Jankovic, "The google car: driving toward a better future?" *Journal of Business Case Studies (Online)*, vol. 10, no. 1, p. 7, 2014.

[2] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, 2012.

[3] B. Massey, M. Thomure, R. Budrevich, and S. Long, "Learning spam: Simple techniques for freely-available software." in *USENIX Annual Technical Conference, FREENIX Track*, 2003, pp. 63–76.

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

[5] N. Capece, U. Erra, and A. V. Ciliberto, "Implementation of a coin recognition system for mobile devices with deep learning," in *2016 12th International Conference on Signal-Image Technology Internet-Based Systems (SITIS)*, Nov 2016, pp. 186–192.

[6] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[7] Z. Cheng, Q. Yang, and B. Sheng, "Deep colorization," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 415–423.

[8] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural networks*, vol. 2, no. 5, pp. 359–366, 1989.

[9] J. Lee and S. Lee, "Hallucination from noon to night images using cnn," in *SIGGRAPH ASIA 2016 Posters*. ACM, 2016, p. 15.

[10] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, "Generative adversarial nets," in *Advances in neural information processing systems*, 2014, pp. 2672–2680.

[11] Y. Shih, S. Paris, F. Durand, and W. T. Freeman, "Data-driven hallucination of different times of day from a single outdoor photo," *ACM Transactions on Graphics (TOG)*, vol. 32, no. 6, p. 200, 2013.

[12] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *ICLR*, 2015.

[13] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2016, pp. 770–778.

[14] U. Erra, S. Senatore, F. Minnella, and G. Caggianese, "Approximate tf-idf based on topic extraction from massive message stream using the gpu," *Inf. Sci.*, vol. 292, no. C, pp. 143–161, Jan. 2015. [Online]. Available: http://dx.doi.org/10.1016/j.ins.2014.08.062

[15] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

[16] X. Glorot, A. Bordes, and Y. Bengio, "Deep sparse rectifier neural networks." in *Aistats*, vol. 15, no. 106, 2011, p. 275.

[17] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 448–456.

[18] W. H. Press, S. A. Teukolsky, W. T. Vetterling, and B. P. Flannery, *Numerical recipes in C*. Cambridge Univ Press, 1982, vol. 2.

[19] J. Burkardt, "The truncated normal distribution," *Department of Scientific Computing Website*, 2014.

[20] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.