

Bee2Fire: A Deep Learning Powered Forest Fire Detection System

Rui Valente de Almeida^{1,2}^a, Fernando Crivellaro¹^b, Maria Narciso¹^c, Ana Isabel Sousa¹^d
and Pedro Vieira¹^e

¹Physics Department, FCT NOVA, Campus de Caparica, 2829-516 Caparica, Portugal

²Compta, S.A., Alameda Fernão Lopes 12, 10th floor, 1495-190 Algés, Portugal

Keywords: Forest Fire Detection, Deep Learning, PyTorch, FastAI, IBM Watson.

Abstract: Bee2Fire is a commercial system for forest fire detection, inheriting from the Forest Fire Finder System. Designed in Portugal, it aims to address one of Southern Europe's main concern, forest fires. It is a well known fact that the sooner a wildfire is detected, the quicker it can be put out, which highlights the importance of early detection. By scanning the landscape using regular cameras and Deep Artificial Neural Networks, Bee2Fire searches for smoke columns above the horizon with a image classification approach. After these networks were trained, the system was deployed in the field, obtaining a sensitivity score between 74% and 93%, a specificity of more than 99% and a precision of around 82%.

1 INTRODUCTION

Bee2Fire is the new commercial name of the Forest Fire Finder (FFF) system, a forest fire detection system developed in 2007 by NGNS-IS, Lda (Valente de Almeida and Vieira, 2017). It was designed in a team effort between FCT-NOVA and Compta, S.A., as a stand-alone commercial AI-based forest fire detection device as a complement to its already deployed spectroscopic capabilities, that used a chemical analysis and traditional machine learning techniques (in particular Support Vector Machines) to infer on the presence of smoke in the atmosphere. This first paper represents an initial effort in developing this system.

In Portugal, southern of Europe and most other warm weather regions on the globe, wildfires are one of the most significant concerns, especially when considering global warming tendencies and what it entails in terms of extreme phenomena. The stage in which a forest fire is detected is a critical factor for its extinction, and the sooner the better (Valente de Almeida and Vieira, 2017). Bee2Fire aims to tackle this issue by employing deep learning models for the classification of images in which there may or may

not be a smoke plume in a remote area.

In this paper, we have used a portion of our image dataset to train deep learning PyTorch (Paszke et al., 2017) models using FastAI (Howard et al., 2018) as a "proxy platform", which were complemented by an IBM Watson Visual Recognition instance, trained with the same dataset. In the field, our combined classifier obtained a Sensitivity of between 74% and 93% and a Specificity of more than 99%, also scoring around 82% on its Precision value. All in all, and especially considering this was only a first approach, our system was very effective in detecting daytime smoke plumes. Future developments include increasing the dataset, changing to an object detection strategy and introducing reinforcement learning.

This paper is structured as follows: Section 2 briefly touches upon the theoretical concepts that were used for the development of this project; Section 3 discusses how these concepts were applied to the particular problem we were trying to solve; Section 4 presents the results that were obtained through the discussed methods; and finally Section 5 addresses what could be concluded through our efforts and most important future developments.

^a  <https://orcid.org/0000-0002-2269-7094>

^b  <https://orcid.org/0000-0002-7534-9149>

^c  <https://orcid.org/0000-0001-5079-9381>

^d  <https://orcid.org/0000-0003-2980-4742>

^e  <https://orcid.org/0000-0002-3823-1184>

2 THEORETICAL BACKGROUND

2.1 Artificial Neural Networks

Bee2Fire is a Deep Learning (DL) enabled application, which is comprised of an image classifier trained to recognise a smoke column that might indicate a fire event. DL is a branch of Machine Learning (ML), powered by deep Artificial Neural Networks (ANN), which are an ML concept developed during the twentieth century and are loosely based on the way biological neurons work (Rosenblatt, 1958). These mathematical structures are capable of *learning* to perform specific tasks by being exposed to a large number of examples. There are several types of ANN, and their usage depends on the type of task at hand. For instance, while Convolutional Neural Networks (CNN - see (Lecun et al., 1998)) are widely used for Computer Vision (CV) applications, Recurrent Neural Networks (RNN - see (Sherstinsky, 2018)) are more used for language translation or tabular data evaluation.

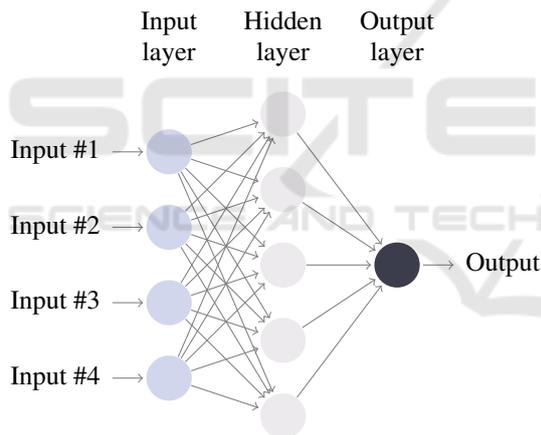


Figure 1: An Artificial Neural Network with one hidden layer.

ANN (see schematic representation in Figure 1) are comprised of a series of neurons, connected in layers through what are called edges (equivalent to biological synapses). Edges have weights attached to them, which determine how much they influence the neurons activation and consequently how they influence the whole network. Neurons themselves hold a real value (an activation), which is calculated by applying a non-linear function (the activation function) to a linear operation calculated over the values of the previous layer's neurons, the edges and the bias for each neuron. The bias is a numerical value that determines how easy (or difficult) for a neuron to be activated (Dreyfus, 1990; Goodfellow et al., 2016).

The process by which a Neural Network *learns* is called training. The goal of this process is to minimise a certain function, the cost function, which compares the output of the network with the desired outcome (i.e., the correct classification). Training an ANN is an iterative process that encompasses two major stages. The first stage is the feed forward stage. In it, all the activation values are calculated using the weights and biases, through the linear and non-linear operations embedded into the network (in the very first stage, weights are randomly initialised). The second stage is called BackPropagation (BP) and updates the weights in order to minimise the cost function by minimising its gradient (its partial derivatives with respect to the weights and biases in the network) (Nielsen, 2015).

Artificial Neural Networks play an important role in modern day society. From the keyboard application in our smartphone (Swiftkey, 2016) to the self driving capabilities of a rising number of our cars (Tesla, 2019), their application is almost ubiquitous. This small introduction to ANN and how one can work with them was not ever meant to be exhaustive, but rather a starting point. More curious readers are redirected to Michael Nielsen's online interactive book (Nielsen, 2015), for a much more in depth description.

2.2 Relevant Particularities of Deep Learning

DL is nothing more than the field of study, within machine learning, that addresses development and application of deep ANN, i.e., ANN with more than one hidden layers. It would therefore be reasonable to include the following discussion in the ANN discussion. However, even if there is no formal distinction between the two, the rate of development to which DL has been subject to in the last decade is vertiginous. By itself, this development would justify the division, but there is more. There are a series of particular techniques (coming from this prodigious growth of Deep Learning) that were born (and only make sense) in this context. In this subsection, we will approach the several that were found useful in the implementation of this project.

Although DL systems had been around for decades, the term Deep Learning was popularised from 2006 forward, when a group of researchers from Canada and the USA figured that one could efficiently train a deep model, something that was considered impossible. Deep learning architectures and training algorithms were at this time old creations, but they have two important requirements, which were diffi-

cult to get in times prior to the 21st century (Goodfellow et al., 2016):

Computational Power One of the reasons why deep neural networks have become so powerful in the last years is the increase in their size, but this introduces an amount of calculations that was simply impossible to manage before;

Data Deep Learning is able to "learn" complex patterns through the hierarchy of learned features (line to curve to circle to wheel). For this, models need enormous amounts of data, and this was something for which there was no capable infrastructure.

In CV, the most important year for DL has been 2012. In this year, the AlexNet image classifier (Krizhevsky et al., 2012) shattered the previous record in the ImageNet competition (Deng et al., 2009) and achieving errors of around 16%, when in 2011 the winning classifier had achieved 26%. This was the first year a deep learning model had won this competition, but it set the trend. In 2015, a mere 3 years after DL models had started leaving their mark on the competition, ResNet (He et al., 2015) achieved an error of 3.6%, 1.4% below human error. For the first time a computer was scoring higher than a human on a large scale image classification task (Goodfellow et al., 2016).

Bee2Fire's local detection engine is built upon a ResNet18 (18 layer ResNet) model. Their introduction by He and his team marked the appearance of one of the most important DL innovations, the concept of residual learning. In their paper, the authors have addressed the known problem of training error degradation appearing for deeper models by using a series of identity mapping shortcut connections skipping over one or more layers, and adding this data to the values that have undergone the linear and non-linear transformations of the skipped layers (He et al., 2015).

As stated above, the training process of this kind of architecture demands access to enormous amounts of data. Unfortunately, it is many times impossible to compile such a dataset. One could expect this to be an insurmountable obstacle for Deep Learning practitioners, but indeed this is not the case. The hierarchic nature of the learnt features in deep models allows an immensely powerful "trick" called Transfer Learning (TL) (Tan et al., 2018), that leverages the fact that whether one is trying to classify fish according to their species or houses according to their style, the basic image components are the same (lines, curves, circles, etc.). Therefore, if we train a network (e.g., a ResNet18) on one of these huge datasets, such as ImageNet's, we can then adapt the last few layers of the

model and train them with our own, much smaller, data. The process of working with pre-trained models which are then adapted to the task at hand is usually called fine-tuning.

Another particular method for working with deep neural models is one called *data augmentation*. The process is once again based on the learning capabilities of this kind of ANN, and is extremely useful for cases in which one can only access very small datasets. The training process aims to make the neural model learn abstract features about the dataset, and some of these abstract features are not related to the way in which the data is presented. Therefore, the same image can be fed multiple times into the dataset, and one can introduce small variations of every image (rotations, flips, noise, etc.). This can make a small dataset workable from a DL perspective (Mikolajczyk and Grochowski, 2018; Goodfellow et al., 2016).

Finally, we would like to discuss implementation methods. Deep Learning and ANN in general can, on a simple level, be easily implemented by hand, since they can be reduced to a series of successive algebraic operations. The problems come when one wants to ensure a high level of performance, which is an absolute requirement when one is dealing with deep models. The number of operations involved in training a reasonably deep neural model goes easily into the tens of millions (ResNet50 has 25.6 million parameters), so every microsecond counts in terms of operational computation time. Fortunately, the last few years have seen the appearance of several software platforms, mainly written in Python and leveraging GPU capabilities, designed to perform deep model training (and inference) with exceedingly high performance levels. Some of the most famous are Google's Tensorflow (Abadi et al., 2016), Caffe (Jia et al., 2014) or Facebook's PyTorch (Paszke et al., 2017). Bee2Fire is the result of the application of many of the theoretical concepts that we have presented in this section through PyTorch via FastAI (Howard et al., 2018), a popular library used for the automation of many of PyTorch's features. The next section details how this was achieved.

3 METHODS

As stated in Section 1, this paper details the implementation for a first approach to forest fire detection using deep learning methods. As a result, the number of images used is relatively small for this kind of application, including only 2378 images, labelled as 'clean' (1526) for clean skies, 'clouds' (571) for cloudy skies and 'smoke' (281) for a smoke col-

umn, acquired by several FFF systems deployed in the North of Portugal. Since this device aimed to detect smoke plumes above the horizon, it is only natural that this is reflected in our dataset, which consists exclusively of images with a very clear sky line (except in some cases of thick clouds). The problem we are trying to solve with our Deep Learning approach is one of single label classification (one image, one category). Before training, images are manually separated into three different folders, according to their label. Dataset division into training and validation sets is performed at training time.

For training, inference and neural network handling in general, we have used the FastAI library (Howard et al., 2018). FastAI is a python library developed with the purpose of automating many of PyTorch’s (Paszke et al., 2017) parametrisation requirements for neural network training. This automation results in a dramatic decrease in development times without compromising accuracy or performance in general, although resulting in a small and acceptable loss of flexibility in comparison with pure PyTorch.

For our application, we have trained a Resnet (He et al., 2015) model with 18 layers, pre-trained on the ImageNet (Deng et al., 2009) dataset. We used transfer learning to complete the training and fine-tune the network outputs to the three categories in which we are interested, as illustrated by Figure 2. Pre-processing consists in a simple resizing and the application of a Gaussian filter, which reduces high-frequency noise. Training starts by loading the data into a specific PyTorch structure called a databunch. This step not only gathers all the data but also handles dataset division and label separation. In addition, it can also apply some data augmentation, which in this case consisted only in small random transforms like rotation, zoom and jitter, besides resizing each image to a 224 pixels wide square. The final dataset that goes into training has 1903 training images and 475 validation images. The second stage is the actual training, which goes on for 20 epochs (training results in Figure 3).

The second stage of the model consists in reloading the data again, but this time resizing the image to a 400 pixels wide square. This technique, of starting the training with a smaller-sized image and then progressing onto larger images is called Progressive Resizing, and it is recommended by FastAI’s authors for increasing the network’s performance (Howard et al., 2018). This stage of the training went on for 10 epochs and results are visible in Figure 4.

As already stated, Bee2Fire combines two types of classifiers for forest fire detection. The last few

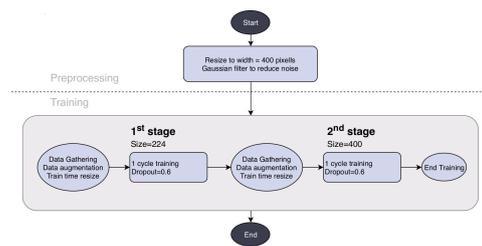


Figure 2: Dataset training workflow. Note that the images are re-dimensioned into two different pixel sizes immediately before training. This procedure was empirically found to retrieve better results, and is recommended by FastAI authors (Howard et al., 2018).

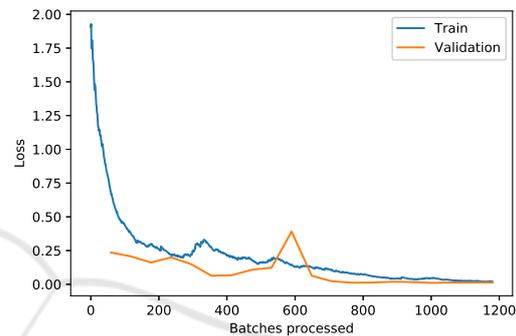


Figure 3: Loss plot for training the 224 pixel model after 20 training epochs.

paragraphs have been describing one of them, which was built by us using a FastAI/PyTorch approach. The other classifier was trained using IBM Watson’s Visual Recognition platform. For this classifier, and since we do not know what are the preprocessing steps applied by the platform itself, we chose not to do anything with the images other than resizing them to 640 pixels wide (for dataset size restrictions only), respecting the original aspect ratio. While the chosen dataset was precisely the same used to train our own classifiers, this and uploading the data onto IBM’s

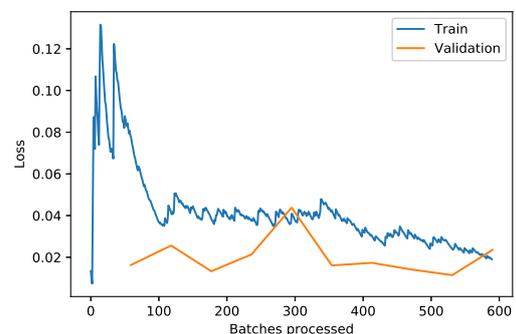


Figure 4: Loss plot for training the 400 pixel model after 10 epochs. Training loss has a clear downward trend, but there is no improvement in the validation loss, indicating the model is not learning with training.

servers was the only thing we had to do, since every other aspect of building the network is automatically run by the platform.

4 RESULTS

Figure 5 depicts the confusion matrix for the classifier trained using the methods described in Section 3. As one can see both from the matrix itself and also by the loss plots of Figure 3, the chosen model has clearly learned some pattern from the provided data.

What these results do not establish in any way is that the trained model can be applied to any other reality other than that in which the images were captured. Since the goal of this study was to develop a first approach to a commercially viable solution for fire detection using proprietary and publicly available classification engines, field testing was necessary. With this in mind, the team assembled a prototype device which was deployed in one of Compta’s commercial partner’s facilities, in Valongo, northern Portugal (see Figure 6 for a screenshot of the prototype’s interface).

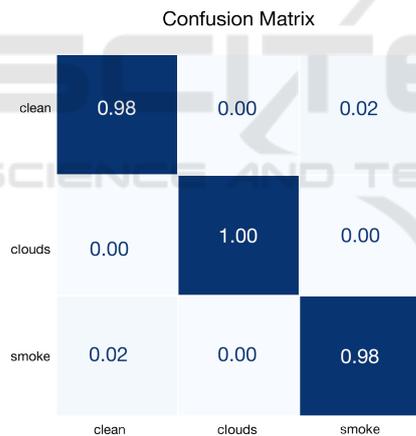


Figure 5: Normalised confusion matrix for Bee2Fire’s classifier, trained using data from previous installations. The matrix shows that the model performs very well with the selected dataset, achieving at least 98% of accuracy for all labels.

The prototype has been working since mid June with little to no intervention on behalf of the project’s team (with some minor maintenance work as an exception). The system’s mode of operation is similar to the one detailed in FFF’s paper (Valente de Almeida and Vieira, 2017), in which the camera is continuously rotating, and an image is captured every 10 seconds. Considering a period of 10 hours (this system only works during the day, for now), this gives us a

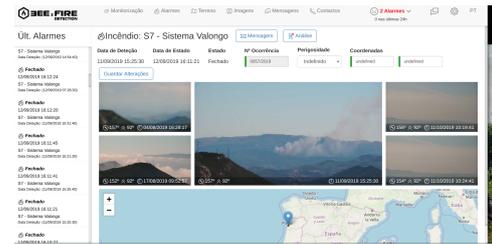


Figure 6: Bee2Fire’s control interface. Although temporary, this is the basis of the commercial interface that will be made available to clients in the near future.

total of 3600 classified images per day. These figures, and the fact that IBM’s visual recognition service is paid on a per-call basis, dictate we only consult the external service when our local classifier is “convinced” the image it is analysing is a smoke column. The classification process is better explained through a schematic representation, which the reader can find in Figure 7.

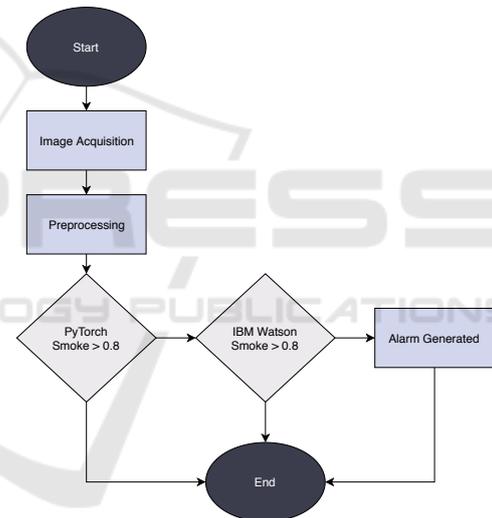


Figure 7: Schematic representation of the analysis process. The system uses a normal surveillance camera to acquire pictures of the landscape in which the device is deployed. Each image is analysed by the PyTorch routine and then, if the smoke probability is higher than a given threshold (typically 0.8), also by the IBM Watson classifier. If the threshold is surpassed in both classifiers, the system issues an alarm.

Given multiple software changes, which are expected in a prototype product, the deployed system has only been able to save alarms (images which trigger a classification score of more than 0.8 for the smoke category) since the beginning of September. Let’s assume, for quantification purposes, the (particularly wildfire-ridden) week of September 9 - 15. During this period, there were 19 wildfires, of which 14 generated one or more detection alarms in the first

10 minutes. In addition, the system issued 3 false alarms. These numbers allow us to calculate sensitivity, specificity and precision values for this test. Results are displayed in Table 1.

The obtained results must be carefully analysed, in light of the intended specifications of this type of device. It is this analysis that generates two different sensitivity values (see Table 1). Direct accounting of detected and undetected events can lead one to conclude the prototype has had a sensitivity of around 74% (during the testing period), which is quite unfair: 4 out of the 5 false negative predictions (system said there was no smoke, but there was a smoke column) are completely outside of what the device is supposed to see (see Figure 8 as an example). If one takes this into account, the sensitivity figure is rather more comfortable, of about 93%.



Figure 8: Example of an image depicting a smoke column that the classifier was not meant to detect, falling completely outside Bee2Fire’s distance specifications.

Now consider that no compensation is made, and one evaluates the system’s performance solely on whether it was or not able to “see” a smoke column. The low specificity value that our classifier has obtained indicates that it cannot successfully identify every positive occurrence. At first glance this might seem very negative, but one must remember that it is very rare for a smoke column to appear in only one image in each scanning movement of the system. Moreover, the low sensitivity is also offset by the fact that a single fire can easily produce more than 15 positive images (although quantifying this is very difficult).

Table 1: Specificity and sensitivity values for Bee2Fire’s classifier. Field test results.

Raw Sens.	Adapted Sens.	Specificity	Precision
73.68%	93.33%	99.99%	82.35%

5 CONCLUSIONS

This study describes the development and implementation of a first approach to automatic fire detection through Deep Learning methods. We have developed our own classifier using open source methods (PyTorch / FastAI) and combined it with a publicly available classifier from IBM. Results were very positive, and the system has achieved a sensitivity of between 74% and 93% for a specificity of more than 99% and a precision of around 82%. Although the first value may be considered on the low side, it is important to have in mind that even if one does not take into account the system’s specification requirements, the problems that may come from this low value are offset by the nature of wildfires and the operation mode of the system.

Regarding future developments, the most important of these is the inclusion of a greater number of images from our dataset into the training process. Surely this will represent an increase in the classifier’s performance and in its ability to generalise. Another development that should be pursued is the conversion of a classification system onto an object detection model. Most DL classification tasks are applied to images in which foreground/background distinction is obvious. In our case, the smoke column that we aim to detect can be in any place in the image, and since it depicts a landscape, one cannot clearly speak of foreground or background. An object detection model, which can be based on the same ANN, could possibly improve the ability for the system to understand if an image contains a smoke plume. Finally, the last future development that should be pursued is the inclusion of a reinforcement learning platform, in which human verification is entered automatically onto the classifier, which would theoretically result in a constantly improving (learning) model.

ACKNOWLEDGEMENTS

We would like to acknowledge the rest of Compta’s team, without whom it would have been impossible to deploy the system or to have an interface with which to control it.

REFERENCES

- Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mane, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viegas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems. arXiv:1603.04467.
- Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). ImageNet: A Large-Scale Hierarchical Image Database. In *Proceedings of the 2009 IEEE Conference on Computer Vision and Pattern Recognition*.
- Dreyfus, S. E. (1990). Artificial neural networks, back propagation, and the kelley-bryson gradient procedure. *Journal of Guidance, Control, and Dynamics*, 13(5):926–928.
- Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press.
- He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep Residual Learning for Image Recognition. arXiv:1512.03385.
- Howard, J. et al. (2018). fastai. <https://github.com/fastai/fastai>.
- Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. (2014). Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv:1408.5093.
- Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS'12 Proceedings of the 25th International Conference on Neural Information*, pages 1097–1105.
- Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-Based Learning Applied to Document Recognition. *proc. OF THE IEEE*.
- Mikolajczyk, A. and Grochowski, M. (2018). Data augmentation for improving deep learning in image classification problem. In *2018 International Interdisciplinary PhD Workshop (IIPhDW)*, pages 117–122. IEEE.
- Nielsen, M. A. (2015). *Neural Networks and Deep Learning*. Determination Press.
- Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. (2017). Automatic Differentiation in {PyTorch}. In *NIPS Autodiff Workshop*.
- Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408.
- Sherstinsky, A. (2018). Fundamentals of Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) Network. arXiv:1808.03314.
- Swiftkey (2016). Swiftkey debuts world's first smartphone keyboard powered by Neural Networks. <https://blog.swiftkey.com/swiftkey-debuts-worlds-first-smartphone-keyboard-powered-by-neural-networks>.
- Tan, C., Sun, F., Kong, T., Zhang, W., Yang, C., and Liu, C. (2018). A Survey on Deep Transfer Learning. arXiv:1808.01974.
- Tesla (2019). Autopilot — Tesla. <https://www.tesla.com/pt/PT/autopilot?redirect=no>. Last accessed in October 14, 2019.
- Valente de Almeida, R. and Vieira, P. (2017). Forest Fire Finder – DOAS application to long-range forest fire detection. *Atmospheric Measurement Techniques*, 10(6):2299–2311.