Lebanese University

Faculty of Engineering III

Electrical and Electronics Engineering Department

Fire Detection Using Neural Networks

Fourth Year – Semester VIII

Spring 2020-2021

Prepared by:

-Mariam Hachem (5509)

-Rihab Atwi (5486)

Presented to: Dr Mohamad Aoude

Abstract

Despite the rapid growth of technologies and smart systems, certain problems remain unsolved or are solved with methods that deliver poor performance. One of these problems is the unexpected outbreak of a fire, an abnormal situation that can rapidly cause significant damage to lives and properties. The technologies underlying fire and smoke detection systems play a crucial role in ensuring and delivering optimal performance in modern surveillance environments. In fact, fire can cause significant damage to lives and properties. Considering that the majority of cities have already installed camera-monitoring systems, we will develop a model to detect the occurance of fires from images, using neural networks. Due to the recent trend of intelligent systems and their ability to adapt with varying conditions, deep learning becomes very attractive for many researchers. In general, neural network is used to implement different stages of processing systems based on learning algorithms by controlling their weights and biases. This report introduces the basic neural network concepts, with a description of major elements consisting of the network, and then, a brief description of the python code of the model.

Table of Contents

Table of Figures:

General Introduction

Building fire is one of the major threats to our daily lives. Although advanced technologies have been developed and implemented in the fire protection and extinguishing systems, fire accidents are still reported in every day. Since it is not possible to design a building with zero fire risk, engineers would modify the building designs and investigate the corresponding fire safety levels in order to reduce the probability of fire occurrence and the fire severity. In the course of the building design, it is a decision making process to determine the acceptability of the design such that, in case of fire, the fire scenario can still be maintained in a tenable condition to the occupants of the building during their evacuation. To make this decision, numerical computation model simulation is usually adopted to determine the tenability. This traditional approach usually requires extensive computer storage and lengthy computational time. Alternatively, ANN has been proposed and proven to be an efficient and effective decision making model in fire applications.

Artificial neural networks (ANN) have been widely adopted as decision support systems in different engineering applications. Recently, ANN has been employed to determine the occurrence of catastrophic fire and to predict the fire and smoke developments. Intelligent approach becomes an alternative way to evaluate the fire safety of a building instead of the traditional numerical approaches which require extensive computer storage and lengthy computation.

Neural networks rely on training data to learn and improve their accuracy over time. However, once these learning algorithms are fine-tuned for accuracy, they are powerful tools in computer science and artificial intelligence, allowing us to classify and cluster data at a high velocity.

In this project, we will develop a model to detect the occurance of fires from images. We will use a dataset of images taken from various security cameras sorted into three categories: default (no fire), smoke or fire. The resulting model could be deployed to alert authorities to the location of fires captured by security cameras installed in various locations (building, shops, streets, wilderness...).

This report is composed of two chapters:

Chapter 1 introduces a general overview of neural networks concepts, with a description of major elements consisting of the network, as well as activation functions that can be used.

Chapter 2 describes briefly the model developped in Google Colaboratory for the fire detection system, using Python programming language.

Chapter 1: Introduction to Neural Networks

1) Introduction

The artificial neural network is a computing technique designed to simulate the human brain's method in problem-solving. In 1943, McCulloch, a neurobiologist, and Pitts, a statistician, published a seminal paper titled "A logical calculus of ideas immanent in nervous activity" in Bulletin of Mathematical Biophysics, where they explained the way how brain works and how simple processing units—neurons—work together in parallel to make a decision based on the input signals. The similarity between artificial neural networks and the human brain is that both acquire the skills in processing data and finding solutions through training.

2) The Biological Neuron

To illustrate the structure of the artificial neural network, an anatomical and functional look must be taken on the human brain first. The human brain consists of about 1011 computing units "neurons" working in parallel and exchanging information through their connectors "synapses"; these neurons sum up all information coming into them, and if the result is higher than the given potential called action potential, they send a pulse via axon to the next stage. Human neuron anatomy is shown in Figure 1.

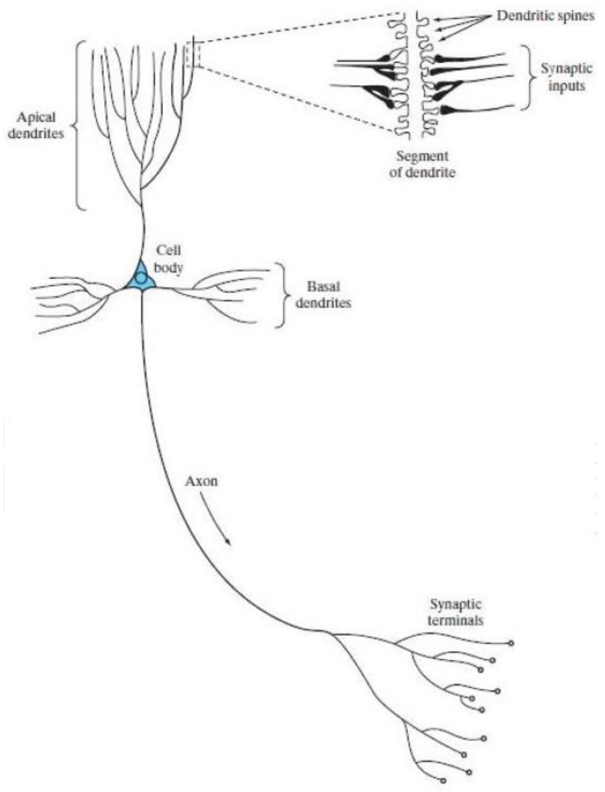

Figure 1: Human Neuron Anatomy

3) Neural Network's Architecture

A neural network is a series of algorithms that endeavors to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates. In this sense, neural networks refer to systems of neurons, either organic or artificial in nature. Neural networks can adapt to changing input, so the network generates the best possible result without needing to redesign the output criteria.

In the same way as the human neuron, artificial neural network consists of simple computing units "artificial neurons," and each unit is connected to the other units via weight connectors; then, these units calculate the weighted sum of the coming inputs and find out the output using squashing function or activation function. Figure 2 shows the block diagram of artificial neuron.

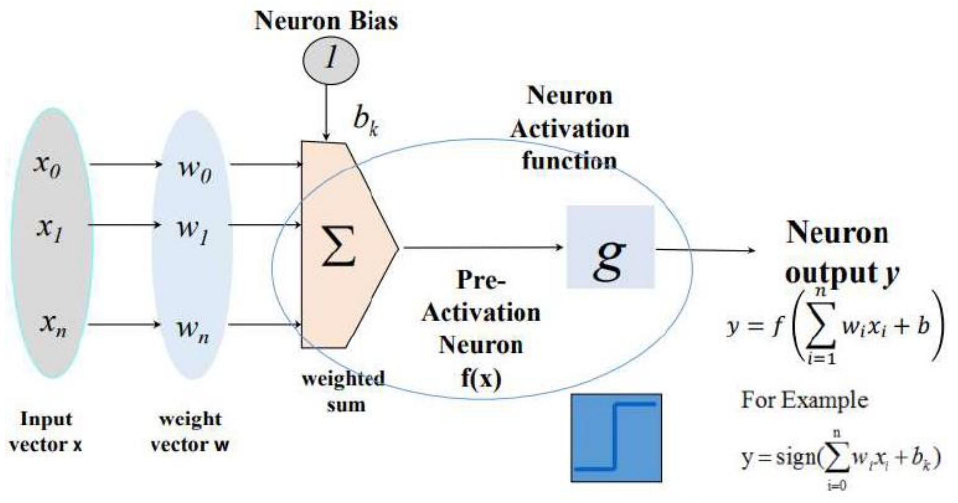
Neuron Bias
1
b_k
Neuron Activation function
x_0
x_1
x_n
w_0
w_1
w_n
Σ
g
Neuron output y
Pre-Activation Neuron f(x)
weighted sum
Input vector x
weight vector w
$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right)$
For Example
$y = \text{sign}(\sum_{i=0}^{n} w_i x_i + b_k)$

Figure 2: Block Diagram of Artificial Neuron

Based on the block diagram and function of the neural network, three basic elements of neural model can be identified:

1. Synapses, or connecting links, have a weight or strength where the input signal x_i connected to neuron k is multiplied by synaptic weight w_{ki}.

2. An adder for summing the weighted inputs.

3. An activation function g to produce the output of a neuron. It is also referred to as a squashing function, in that it squashes (limits) the amplitude range of the output signal to a finite value.

The bias b_k has the effect of increasing or decreasing the net input of the activation function, depending on whether it is positive or negative, respectively.

Mathematically, the output on the neuron k can be described as

$$y_k = g(\sum_{i=0}^{n} w_{ki} x_i + b_k)$$

Where

$x_0, x_1, \dots \dots ., x_n$ are the input's signals.

$w_{k0}, w_{k1}, \dots \dots ., w_{kn}$ are the respective weights of neuron.

b_k is the bias.

g is the activation function.

The problem of learning a neural network is reduced to learning the weights w_{ki} and biases b_k to best fit the input output data (x,y).

4) Activation Functions

An activation function in a neural network defines how the weighted sum of the input is transformed into an output from a node or nodes in a layer of the network.

The choice of activation function has a large impact on the capability and performance of the neural network, and different activation functions may be used in different parts of the model.

All hidden layers typically use the same activation function. The output layer will typically use a different activation function from the hidden layers and is dependent upon the type of prediction required by the model.

4.1) Activation for Hidden Layers

A neural network may have zero or more hidden layers. Typically, a differentiable nonlinear activation function is used in the hidden layers of a neural network. This allows the model to learn more complex functions than a network trained using a linear activation function.

a. Rectified Linear Activation (ReLU)

The ReLU, shown in figure 3, is the most used activation function in the world right now. Since, it is used in almost all the convolutional neural networks or deep learning. It is common because it is both simple to implement and effective at overcoming the limitations of other activation functions.

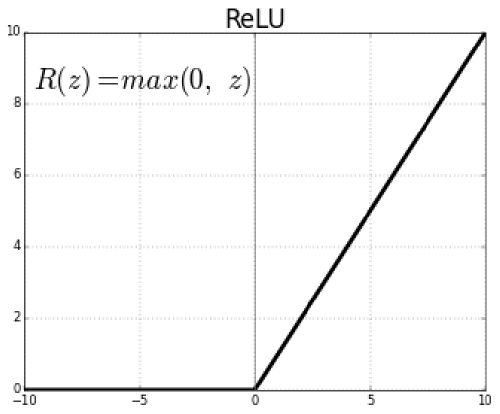

Figure 3: ReLU Activation Function

R(z) is zero when z is less than zero and R(z) is equal to z when z is above or equal to zero.

b. Sigmoid (Logistic)

The sigmoid function, shown in figure 4, takes any real value as input and outputs values in the range 0 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to 0.0.

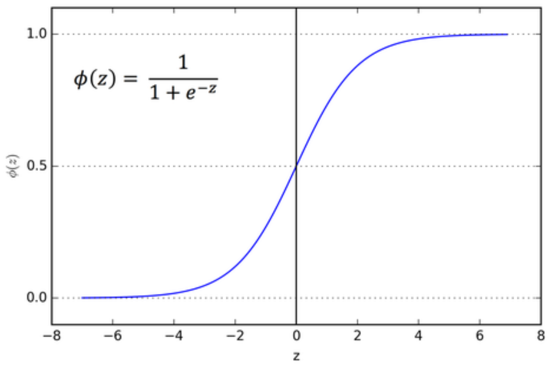

Figure 4: Sigmoid Activation Function

It is described by:

$$y = \frac{1}{1 + e^{-x}}$$

c. Hyperbolic Tangent (Tanh)

Tanh, as shown in figure 5, is very similar to the sigmoid activation function and even has the same S-shape.

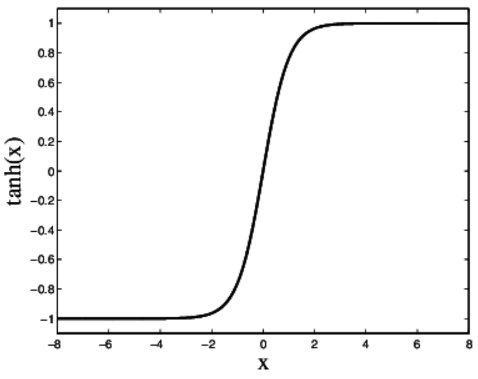

Figure 5: Hyperbolic Tangent Activation Function

The function takes any real value as input and outputs values in the range -1 to 1. The larger the input (more positive), the closer the output value will be to 1.0, whereas the smaller the input (more negative), the closer the output will be to -1.0.

It is described by:

$$y = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

4.2) Activation for the Output Layer

The output layer is the layer in a neural network model that directly outputs a prediction. All feed-forward neural network models have an output layer.

a. Linear Function

The linear activation function, shown in figure 6, does not change the weighted sum of the input in any way and instead returns the value directly. The output is proportional to the input.

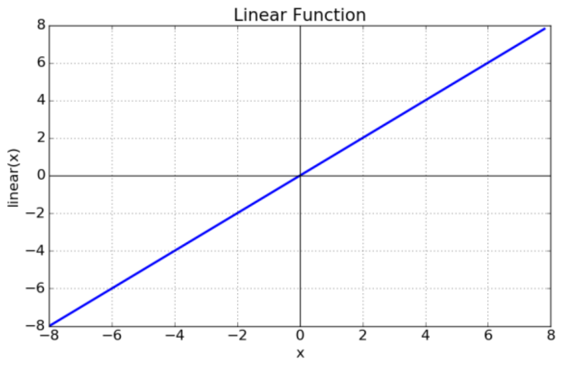

Figure 6: Linear Activation Function

b. Sigmoid (Logistic)

The sigmoid of logistic activation function was described previously. Target labels used to train a model with a sigmoid activation function in the output layer will have the values 0 or 1. Therefore, it is especially used for models where we have to predict the probability as an output. Since probability of anything exists only between the range of 0 and 1, sigmoid is the right choice.

c. Softmax Activation Function

The softmax function outputs a vector of values that sum to 1.0 that can be interpreted as probabilities of class membership.

The softmax function is calculated as follows: $y = \frac{e^x}{\sum e^x}$

Target labels used to train a model with the softmax activation function in the output layer will be vectors with 1 for the target class and 0 for all other classes.

5) Cost Functions

A cost function is a measure of "how good" a neural network did with respect to its given training sample and the expected output. It also may depend on variables such as weights and biases. A cost function is a single value, not a vector, because it rates how good the neural network did as a whole.

A cost function is a form of the error resultant from the difference between a “desired output” and the “actual output”. Training a network is equivalent to finding the “optimal” weights that result in a “minimized cost function”.

Specifically, a cost function is of the form:

$$C(W, B, x,)$$

Where W is our neural network's weights, B is our neural network's biases, x is the input of a single training sample, and y_d is the desired output of that training sample.

Some examples of cost functions used in neural networks are:

a. Quadratic Cost (Least Square Error)

$$J = \sum_i E(i)^2 = \sum_i (y_d(i) - \ y_a(i))^2$$

b. Cross Entropy

$$J = \sum_i E(i)^2 = \sum_i y_a(i) ln y_d(i) + \big(1 - y_a(i)\big) \ln(1 - y_d(i))$$

6) Conclusion

In this chapter, we presented briefly the architecture of a neural network, types of activation functions, and some used cost functions. Additional details will be explained in the second chapter, along with the code.

Chapter 2: Building the Fire Detection Model with Python

1) Introduction

Using a dataset of images taken from various security cameras, the fire detection model was developed in Google Colaboratory, which is a product from Google Research that allows anybody to write and execute arbitrary python code through the browser. It is especially well suited to machine learning, data analysis and education. We chose this environment to be able to run the code and track each execution step by step.

2) Dataset Preparation

A) To be able to download the dataset, we upload the "kaggle.json" file into the Colab virtual storage. The dataset will be downloaded to Colab's storage in a folder called "data".Images are sorted into three categories: default (no fire), smoke or fire. Images are placed in folders named after those images' label.

B) Two functions were implemented:

- A function "list_dataset" that returns: a list containing the file paths for all images in a given folder (train or test), and a list containing the corresponding label for each image (taken from the folder's name.

- A function "load_dataset" that takes the lists provided by "list_dataset" to: load all the images into memory, preprocess the images (resizing and normalizing), one-hot encode the labels, and shuffle the samples. It returns an array containing all the images along with an array containing their corresponding labels.

After one-hot encoding, each label that was a categorical value, becomes a binary vector where each column is assigned to a label:

[1, 0, 0] inidcates the 'default' label.

[0, 1, 0] indicates the 'smoke' label.

[0, 0, 1] indicates the 'fire' label.

Images are read, resized to 224x224 and normalized so that pixel values are grey scaled between 0 and 1, and stored in a Numpy array.

C) We used the two implemented functions to load the dataset and split it for training and testing, with the function *train_test_split* from the library *sklearn.model_selection*.

We have in total 864 images. Each image is a numpy array of dimensions 224x224. 70% of the data are for the training and the other 30% for the testing.

D) We plotted a histogram for the distribution of the classes, using the library *seaborn*. Unfortunately, the label distribution is not balanced, as shown in figure 7.

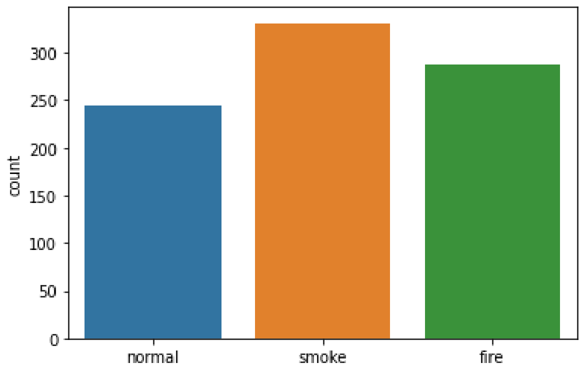

Figure 7: Classes Distribution of the Dataset

E) Using the library *matplotlib.pyplot*, we plotted three images: one belonging to each class. A 'default' (normal) image, then a 'smoke' one, and then a 'fire'.

3) Fully Connected Neural Network

A) Before entering the images as inputs to the neural networks, we flattened them: each image was reshaped from a 224x224 numpy array to a single 50176-dimensional vector.

B) Using the *tensorflow.keras* library:
We built a sequential model, so it is a forward propagation neural network, composed of dense layers.

We added 4 layers. The first layer has 512 neurons with the 50176 inputs representing the dimension of each image. The second and the third layers have 128 neurons each one. The output layer has 3 neurons because the output will be a 3-dimensional vector representing the label of the image.

The first three layers have the 'relu' as the activation function.

The output layer has the 'softmax' as the activation function, because it will return a probability for each class (each column of the output vector).

Finally, we showed a summary of the model with its layers and the number of parameters to train in each layer.

C) To compile the model, we used the stochastic gradient descent 'sgd' optimization algorithm, the 'categorical_crossentropy' loss function and the 'accuracy' metrics. Accuracy is one metric for evaluating classification models. Informally, accuracy is the fraction of predictions the model got right.

D) To fit the model, we used 15 epochs, a batch size of 100, and 20% of the data for validation.

The validation split is the fraction of the training data to be used as validation data. The model will set apart this fraction of the training data, will not train on it, and will evaluate the loss and any model metrics on this data at the end of each epoch.

The batch size is a number of samples processed before the model is updated.

The number of epochs is the number of complete passes through the training dataset.

We wanted to evaluate the performance of our model so we plotted the progress of the accuracy and the loss with the number of epochs, for the training and for the validation as well.

At the end, we obtained an accuracy of 0.908, which is high (When the code is run another time a different accuracy is obtained but it will always be high).

E) Finally, we wanted to test out model on the testing data. We chose a random image of index 250 from the 260 testing images; we printed the real label from its corresponding test label, and then the predicted output. We obtained a vector of 3 probabilities. The highest probability corresponded to the label of the image, so the model did a correct prediction.

F) To compute the precision, recall and F1-score for each class, we showed the classification report.

4) Conclusion

In the end, we built a sequential model composed of 4 dense layers, we trained it and we obtained satisfying results. We plotted the variation of the accuracy and the loss with the number of epochs during the training. We obtained a high accuracy and high metrics. The model did a correct prediction for the chosen image.

General Conclusion

The technologies underlying fire and smoke detection systems play a crucial role in ensuring and delivering optimal performance in modern surveillance environments. Considering that the majority of cities have already installed camera-monitoring systems, this encouraged us to develop a model to detect the occurrence of fires from images, using a fully connected neural network. The developed model showed high accuracy and precision, and therefore can be used to prevent from a fire outbreak in different places, thus saving people lives.

References

[1] Zayegh, A. and Bassam, N., 2018. Neural network principles and applications (Vol. 10). London: Pearson.

[2] Valikhujaev, Y., Abdusalomov, A. and Cho, Y.I., 2020. Automatic fire and smoke detection method for surveillance systems based on dilated cnns. Atmosphere, 11(11), p.1241.

[3] Lee, E.W.M., 2010. Application of artificial neural network to fire safety engineering. In Handbook on Decision Making (pp. 369-395). Springer, Berlin, Heidelberg.

[4] SAGAR SHARMA. (2017, sept 6). Activation Functions in Neural Networks. Towards data science. https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6.