Intelligent Sensor Fusion with Deep Learning

Asm Nurussafa
Electronics Engineering
Hochschule Hamm-Lippstadt
Lippstadt, Germany
asm.nurussafa@stud.hshl.de

Abstract—**Odor classification is very important in both industry and everyday life. In this paper, we will introduce sensor fusion and deep learning models to classify products using their odor. We will use data from multiple gas sensors from different products such as tea, coffee etc. We will then use these data and combine them to train a model using Convolutional Neural Network (CNN) and then use this model to predict and evaluate futher data.**

Index Terms—**Odor classification, Sensor Fusion, CNN, Keras, Edge Impulse.**

I. Introduction

Odor detection and classification is crucial in many fields, from public health to industrial applications. Our sense of smell can alert us to dangers such as gas leaks, fires, and spoiled food, as well as help us identify pleasant scents like flowers, perfumes, and food aromas. One of the most significant benefits of odor detection is its potential for identifying disease. Studies have shown that certain illnesses like Parkinson's and Alzheimer's disease can be detected through changes in body odor, even before clinical symptoms appear. In fact, researchers are currently working on developing "electronic noses" that can detect specific odor signatures of various diseases, allowing for earlier diagnosis and treatment. [1]
In industry, odor classification plays a crucial role in quality control. For example, in the production of food and beverages, the presence of unpleasant odors can indicate contamination or spoilage. In the perfume industry, the ability to accurately classify and describe scents is essential to creating new fragrances and maintaining consistency in existing products.
In this paper, we will look at a simple study of odor detection to identify different types of tea, coffee etc. and discuss how sensor fusion techniques with deep learning models can be combined to better predict such products.

II. Sensor Fusion

Sensor fusion is the process of combining data from multiple sensors to improve the accuracy and reliability of measurements and inferences. This technique is commonly used in many fields, including robotics, autonomous vehicles, aerospace, and healthcare.

The main idea behind sensor fusion is to compensate for the limitations of individual sensors by combining their outputs into a more accurate and reliable estimate of the measured quantity. For example, if we want to estimate the position and orientation of a robot, we can use multiple sensors such as cameras, gyroscopes, accelerometers, and GPS, each of which provides partial and sometimes conflicting information about the robot's state. By fusing the data from these sensors, we can obtain a more robust and accurate estimate of the robot's position and orientation. [3]. Sensor fusion can be done in various ways, depending on the specific application and the types of sensors involved. There are many sensor fusion techniques [4].

Sensor fusion has many advantages, such as improving the accuracy, robustness, and reliability of measurements, reducing the effects of sensor noise and errors, and providing redundant information in case of sensor failures. However, it also has some challenges, such as the need for accurate calibration and synchronization of the sensors, the selection of appropriate fusion algorithms, and the management of large amounts of data. There are various types of sensor fusion [5] [6] , including:

- Data-level fusion: combining raw sensor data at the signal level.
- Feature-level fusion: extracting features from the raw data before combining them.
- Decision-level fusion: combining the decisions made by multiple sensors or algorithms.
- Hybrid fusion: combining multiple levels of fusion or combining multiple types of sensors.

III. Convolutional Neural Network (CNN)

A convolutional neural network (CNN) is a type of deep neural network that is widely used for image and video processing tasks. It is inspired by the structure and function of the visual cortex in the human brain, which is specialized in detecting and recognizing visual patterns. The key feature of CNNs is their ability to learn spatial hierarchies of features from the input data. They achieve this by using convolutional layers that apply filters or kernels to the input data and produce feature maps that capture the presence of certain visual patterns at different scales and locations in the input. [7] [8]

CNNs typically consist of multiple layers, including convolutional, pooling, and fully connected layers, and are trained using backpropagation and stochastic gradient descent. They are capable of learning complex and abstract features from the input data, and have achieved state-of-the-art performance on many computer vision tasks, such as image classification, object detection, segmentation, and recognition. Some of the advantages of CNNs include their ability to handle high-dimensional and complex input data, their ability to learn and generalize from large amounts of training data, and their ability to reduce the number of parameters and computational complexity compared to fully connected networks. [9] [10]

IV. Keras

Keras is an open-source deep learning library that is widely used for building and training various types of neural networks, including convolutional neural networks (CNNs). It provides a simple and user-friendly interface for designing, configuring, and training deep learning models, and supports both CPU and GPU computation. In the context of CNNs, Keras provides a high-level interface for defining the architecture of the network, including the number and type of layers, the activation functions, and the optimization algorithm. It also provides built-in functions for loading and preprocessing image data, as well as for visualizing and evaluating the performance of the model.
To use Keras for building a CNN, the user first needs to define the model architecture, typically using the Sequential or Functional API. The model can then be trained using the fit() function, which takes the input data, the output labels, and various training parameters such as batch size, number of epochs, and validation split. During the training process, Keras uses backpropagation and gradient descent to update the parameters of the model and minimize the loss function. It also supports various optimization algorithms such as stochastic gradient descent (SGD), Adam, and RMSprop, and allows the user to customize the learning rate and other hyperparameters. Once the model is trained, it can be used for making predictions on new data using the predict() function, and the performance can be evaluated using various metrics such as accuracy, precision, recall, and F1 score. [11] [12]

V. Case Discussion

A. Obtaining raw dataset

For our case study, we will work with the dataset from Benjamin Cabe [13]. The data is takes from odors of tea, coffee and some spirits. Benjamin used the SeeedStudio's Multichannel Gas Sensor v2 which can identify carbon monoxide(CO), nitrogen di-oxide(NO2), ethanol and some amounts of different volatile compounds. To augment this system, a spg30 sensor was added which gives some equivalent carbon dioxide mixture and a different measure of volatile organic compounds and finally, to run some tesets, a BME 680 sensor was used that can measure temperature, humidity , pressure and some air quality. These are then connected to a easy to use Wio Terminal (an Arduino-compatible prototyping platform). With this data from these sensors, we can simply use machine learning algorithms that can automatically find correlation between the features and combine them to classify different types of odors. However, this only works if we are simply trying to make an overall prediction and classification

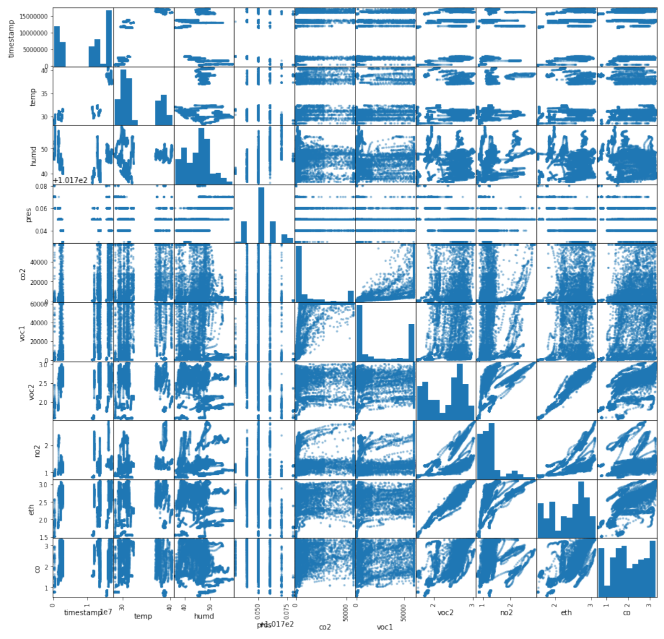


Fig. 1. Features of all raw data.

about somethings's odor. If we need the actual accurate gas measured, this technique likely would not work at least for the limited data we are planning to gather.

The data is then collected from three different environments: kitchen, office and outdoors.

B. Uploading dataset and Pre-processing the data

We will then upload the dataset from the different environments to a python script in Google Colab environment. The following plot, Fig. 1 shows the feature from all the different products in different environment.

So, the data are taken from the following products: background, coffee, spirit-gin, spirit-rum, spirit-vodka, spirit-whiskey, tea-black, tea-chamomile.
The features, as seen from the figure, are: temperature (temp), humidity (hum), carbon dioxide (co2), volatile compound 1 (voc1), volatile compound 2 (voc2), nitrogen dioxide (no2), ethanol (eth) and lastly, carbon monoxide (co). A correlation between the features can be seen in Fig. 2. As seen, the yellow boxes show that correlation is 1, as expected. The darker the color gets, the less correlation it has to that particular feature.

Now, the values of each of the features ranges widely and this would be difficult for our network to draw conclusions and correlations. For this, it is necessary to standardize and normalize our raw data. We can actually plot a histogram of our raw data features and we will see that none of the data has a Gaussian distribution. Anyways, to standardize and normalize, we will used the following formula as shown in Fig. 3. After that, our initial pre-processing is complete. The output is very similar to the Fig. 1 , only that the value ranges for -1 to 1 or 0 to 1. It is important to note that we dropped the timestamps feature since that is not required, and also the

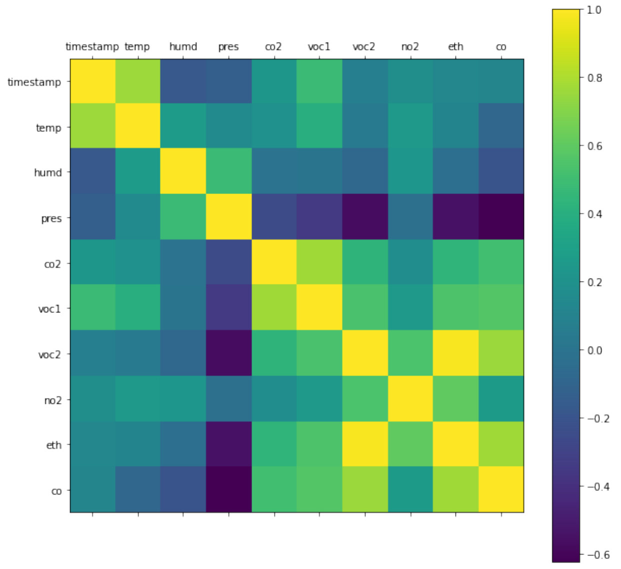
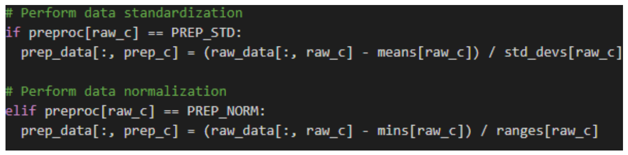


Fig. 2. Correlation between the features.



```
# Perform data standardization
if preproc[raw_c] == PREP_STD:
  prep_data[:, prep_c] = (raw_data[:, raw_c] - means[raw_c]) / std_devs[raw_c]

# Perform data normalization
elif preproc[raw_c] == PREP_NORM:
  prep_data[:, prep_c] = (raw_data[:, raw_c] - mins[raw_c]) / ranges[raw_c]
```

Fig. 3. Standard deviation and normalization.

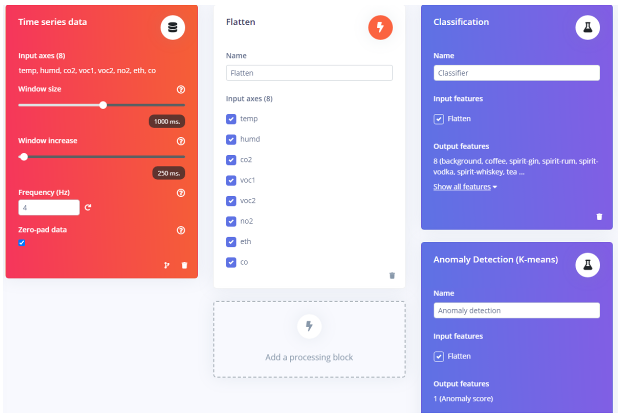
pressure parameter, since as seen from the plot, the values stay relatively static.

We, then, save this data as csv and move on to the next step.

C. Creating learner model in Edge Impulse

Edge Impulse is a free to use development platform for machine learning. It is very easy to use and gives more freedom to experiment with different kinds of models, allowing deployment to edge devices easily. We, then upload the saved pre-processed data to Edge Impulse with a default 80:20 default train:test split. The labelling of the pictures is inferred from the filenames.

Then, we start creating the impulse as shown in Fig.4. First, we choose the amount of data to be processed per minute. As a default, the we chose about 1000 ms. window size and as a precaution choose window increase size of 250 ms in case the sample data is greater than the window size. Next, we choose the pre-processing blocks. For our case, we chose the *Flatten* block and select all our eight parameters as input. Next, and the most crucial part, we choose our learning block. For our method, we chose the *NN Classifier* learning block. This block is based on the the Keras framework and we will see how we can tweak the options according to our need. It is notable that this classifier uses the ***Keras Sequential Model***. We also chose an *Anomaly Detection* to look out for any anomalies.



Time series data

Input axes (8)
temp, humd, co2, voc1, voc2, no2, eth, co

Window size: 1000 ms.

Window increase: 250 ms.

Frequency (Hz): 4

Zero-pad data ✓

Flatten

Name: Flatten

Input axes (8)
✓ temp
✓ humd
✓ co2
✓ voc1
✓ voc2
✓ no2
✓ eth
✓ co

Classification

Name: Classifier

Input features
✓ Flatten

Output features
8 (background, coffee, spirit-gin, spirit-rum, spirit-vodka, spirit-whiskey, tea ...

Anomaly Detection (K-means)

Name: Anomaly detection

Input features
✓ Flatten

Output features
1 (Anomaly score)

Fig. 4. Creating impulse

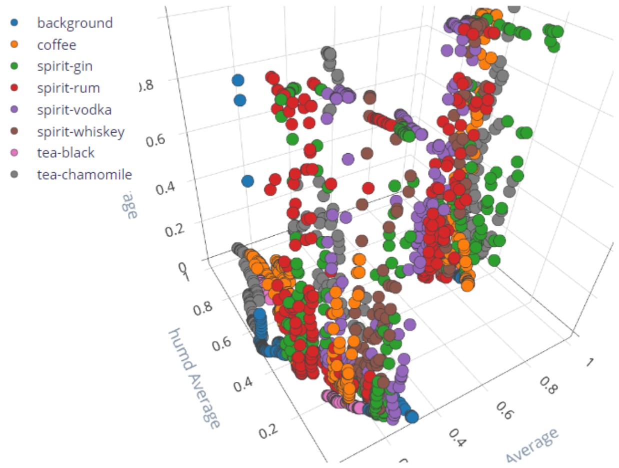


Fig. 5. Feature explorer.

Next, we choose our parameters for *Flatten* pre-processing block. Using this, we can calculate the average, minimum, maximum, root-mean square, standard deviation etc. After generating the features, it would look something show in Fig.

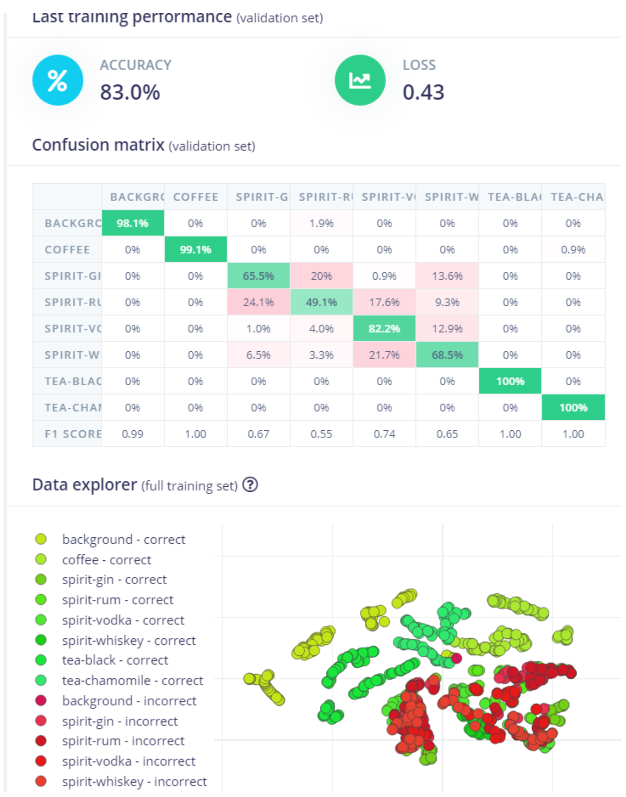
Up next, comes the most crucial part of our discussion, the *NN Classifier* learning block. For our training model, we used the following settings:

- Number of training cycles(epochs): 150
- Learning rate: 0.0005
- Validation set size: 20
- Batch size: 32

For the architecture of our model, we used the *EON Tuner* option in Edge Impulse, that basically calculates the accuracy, precision levels by comparing them to different combinations of layers. This takes some time but for our learner, we used four layers:

- Dense layer (40 neurons)
- Dropout (rate 0.25)
- Dense layer (20 neurons)
- Dropout (rate 0.25)

The dense layers are fully connected layers and is the most



Last training performance (validation set)

ACCURACY 83.0%

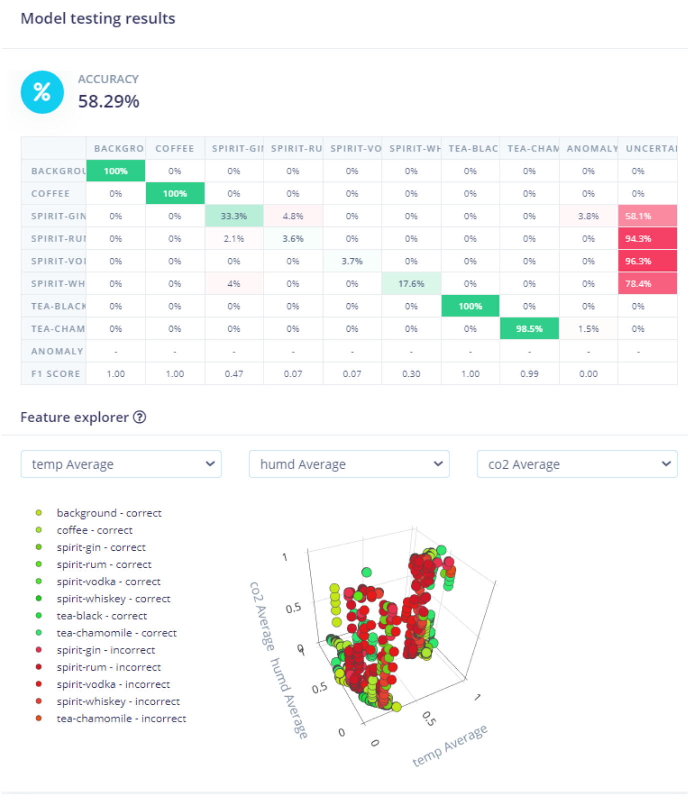
LOSS 0.43

Confusion matrix (validation set)

	BACKGR(	COFFEE	SPIRIT-G	SPIRIT-RI	SPIRIT-V(	SPIRIT-W	TEA-BLA(	TEA-CHA
BACKGRC	98.1%	0%	0%	1.9%	0%	0%	0%	0%
COFFEE	0%	99.1%	0%	0%	0%	0%	0%	0.9%
SPIRIT-GI	0%	0%	65.5%	20%	0.9%	13.6%	0%	0%
SPIRIT-RU	0%	0%	24.1%	49.1%	17.6%	9.3%	0%	0%
SPIRIT-VC	0%	0%	1.0%	4.0%	82.2%	12.9%	0%	0%
SPIRIT-WI	0%	0%	6.5%	3.3%	21.7%	68.5%	0%	0%
TEA-BLAC	0%	0%	0%	0%	0%	0%	100%	0%
TEA-CHAI	0%	0%	0%	0%	0%	0%	0%	100%
F1 SCORE	0.99	1.00	0.67	0.55	0.74	0.65	1.00	1.00

Data explorer (full training set) ?

Fig. 6. Training performance and confusion matrix.



Model testing results

ACCURACY 58.29%

	BACKGRO	COFFEE	SPIRIT-GI	SPIRIT-RU	SPIRIT-VO	SPIRIT-WH	TEA-BLAC	TEA-CHAM	ANOMALY	UNCERTAI
BACKGROU	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%
COFFEE	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%
SPIRIT-GIN	0%	0%	33.3%	4.8%	0%	0%	0%	0%	3.8%	58.1%
SPIRIT-RUI	0%	0%	2.1%	3.6%	0%	0%	0%	0%	0%	94.3%
SPIRIT-VOI	0%	0%	0%	0%	3.7%	0%	0%	0%	0%	96.3%
SPIRIT-WH	0%	0%	4%	0%	0%	17.6%	0%	0%	0%	78.4%
TEA-BLACK	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%
TEA-CHAM	0%	0%	0%	0%	0%	0%	0%	98.5%	1.5%	0%
ANOMALY	-	-	-	-	-	-	-	-	-	-
F1 SCORE	1.00	1.00	0.47	0.07	0.07	0.30	1.00	0.99	0.00	

Feature explorer ?

temp Average | humd Average | co2 Average

Fig. 7. Results for model testing

simplest form of a neural network layer. The dropout layers are used to help with overfitting. Since our data does not have any spatial parameter, the 1D / 2D convolutional or pooling is not required. In any case, the default Keras layers are already included in the source code. And finally our output has 8 layers according to the parameters. Additionally, Edge Impulse provided the *Keras Expert Mode* where we can tweak the source code and change, for instance, batch size , activation function etc. The output for this is seen as in Fig. 6. From this, we can see that the training has an accuracy percentage of 83 and a loss percentage of 0.43. The predicted and ground truth can be seen from the confusion matrix.

D. *Results after testing the model.*

Now that we have trained our model, we can test this against the set we kept aside for training. Once again, this is done randomly and the prediction is made based only on the learner model. As seen in Fig. 7. We can see that it yields an accuracy percentage of 58.29. It is notable that the model can fully predict, as seen from the confusion matrix, the following classes: background, coffee, tea-black and tea-chamomile.

From this study, we can infer that there is quite some room for improvement since the accuracy is low. To further improve the accuracy, we can take the following steps:

- Increase the number the epochs. This should increase the accuracy of the training model but the downside being it also increases the computation time.
- Increase the batch size. We used 32 for our batch size but we can increase this value to gain a better percentage of accuracy.
- Change the activation function. We used the deafault ’relU’ for our setting.
- Decrease dropout rate and experiment to find the best rate.
- Find the optimal learning rate where the model does not overshoot too quickly to increase loss percentage.
- Use low latency hardware.

Afterwards, this learned model can be deployed into edge devices, like the Wio Arduino terminal. In the end, we can test out products, for instance, if a particular product is coffee or not in *real-time*. We have, what we can call, an AI-nose. The implementation in Edge Impulse for this project can be found in [15]

VI. Further work and Summary

From our discussion in the paper, we have introduced important concepts such as sensor fusion, CNN, Keras. We observed how we can combine different features from different sensors and use them in a deep learning algorithm to predict specific products or compounds in *real-time*. We had low accuracy but can improve our performance significantly using the points discussed above. Although, this simple experiment has lots of limitations (would only work in the environments tested, etc.) , we can carry forward this same mechanism into more complex fields. For example, such an application would be of great help in medical sectors, in detecting smokes or to prevent fires etc. The applications, are simply, limitless.

All the relevant files for this experiment can be found in my github repository. [14]

VII. Declaration of Originality

I, ***Asm Nurussafa***, herewith declare that I have composed the present paper and work by myself and without the use of any other than the cited sources and aids. Sentences or parts of sentences quoted literally are marked as such; other references with regard to the statement and scope are indicated by full details of the publications concerned. The paper and work in the same or similar form have not been submitted to any examination body and have not been published. This paper was not yet, even in part, used in another examination or as a course performance. I agree that my work may be checked by a plagiarism checker.

10.01.2023, Hamm **Asm Nurussafa**

References

[1] Polymenidou, M., and Sklaviadis, T. (2018). Odor Detection and Disease Diagnosis. In Molecular Diagnostics (pp. 137-144). Humana Press.

[2] Wilson, A. D., and Baietto, M. (2010). Applications and advances in electronic-nose technologies. Sensors, 10(2), 470-498.

[3] Durrant-Whyte, H., and Bailey, T. (2006). Simultaneous localization and mapping: part I. IEEE Robotics and Automation Magazine, 13(2), 99-110.

[4] Borenstein, J., Everett, H., and Feng, L. (1996). Where am I? Sensors and methods for mobile robot positioning. University of Michiga

[5] Liu, H., Hu, X., Li, W., Xu, M., and Jiang, L. (2017). A survey of multisensor fusion methods. Information Fusion, 35, 68-75.

[6] Khan, S., and Ullah, S. (2017). A review of data fusion techniques. Information Fusion, 33, 100-111.

[7] LeCun, Y., Bengio, Y., and Hinton, G. (2015). Deep learning. Nature, 521(7553), 436-444.

[8] Goodfellow, I., Bengio, Y., and Courville, A. (2016). Deep learning. MIT Press.

[9] Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[10] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., and Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

[11] Chollet, F. (2015). Keras: Deep learning library for Theano and TensorFlow. https://keras.io/

[12] Brownlee, J. (2019). Introduction to Convolutional Neural Networks with Keras for Machine Learning. Machine Learning Mastery. https://machinelearningmastery.com/introduction-to-convolutional-neural-networks-with-keras-for-machine-learning/

[13] https://blog.benjamin-cabe.com/2021/08/03/how-i-built-a-connected-artificial-nose

[14] https://github.com/Asm-Nurussafa/Deep-Learning

[15] https://studio.edgeimpulse.com/public/189585/latest

IEEE conference templates contain guidance text for composing and formatting conference papers. Please ensure that all template text is removed from your conference paper prior to submission to the conference. Failure to remove the template text from your paper may result in your paper not being published.