Robotic Inference

Miguel A. Colmenares Barboza

***Abstract* — The projects presented in this document focus on the recognition of objects using a video camera that transmits live. For the recognition of objects, two neural networks with different samples were trained. Then with the video camera transmitting live, the neural network uses the images captured as input and infers as a label a tag for the object captured in the image as well as generating an electronic action type such as turning on a led.**

Index Terms —Robot, IEEEtran, Udacity, \LaTeX, deep learning.

I. INTRODUCTION

More than a decade ago, artificial neural network algorithms are used in robotic systems to solve complex problems and this document presents a project that seeks to address this particular issue. The project is divided into two parts.

The first part is developed as an exercise training a neural network to recognize handwritten digits, using the MNIST database. This database has 60 thousand training images and 10 thousand test images. It was created by Yann LeCun, Corinna Cortes, Christopher JC Burges to recognize postal codes and is a common exercise for those who are initiated in the world of deep learning.

In the second part, the training of a neural network is also developed, but in this occasion the object to be recognized and identified was chosen. This is for the purpose of developing your own database and not using an existing database. The details of the chosen object are disclosed in a later section.

II. DATABASE / ~~Data Acquisition~~

The Deep Learning GPU Training System (DIGITS) was used to train the two neural networks of the project.

DIGITS is not a framework. DIGITS is a wrapper for NVCaffe ™, Torch ™ and TensorFlow ™; It provides a graphical web interface for those frames instead of treating them directly on the command line.

One of the advantages of using digits is that this wrapper already comes with a storage of neural network models to use such as LeNet AlexNet and GoogLeNet. With DIGITS at hand, these are the actions taken for the training of a neural network.

When loading DIGITS, a graphic web interface opens.

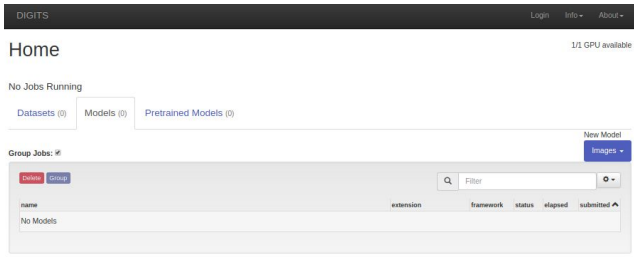

Fig 1. Graphic web interface of DIGITS.

The first step for training neural networks is loading a database. To load a database, open the Dataset section and click on Image> Classification, then choose the following options:

MNIST Dataset

In the Dataset section of DIGITS, the following was provided:

- The path of the main folder of our database
- Gray as a type of image because the images in the MNIST database are grayscale.
- 28 x 28 as the image size because it is the image size of the MNIST database.
- MNIST Dataset as the name of the database for DIGITS.
- Click on create

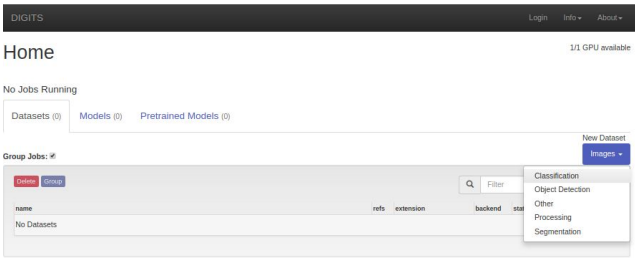

Fig 2. Dataset section. DIGITS.

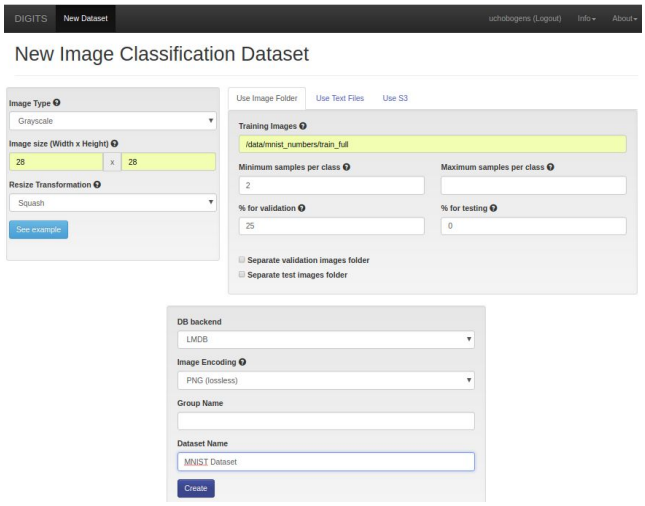

Fig 3. Dataset section. New Image Classification Dataset.

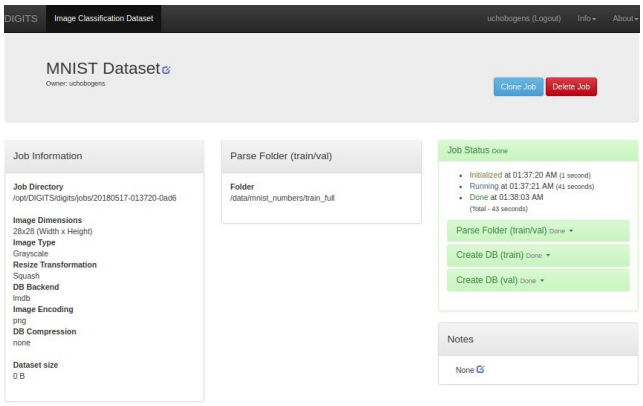

Fig 4. MNIST Dataset.

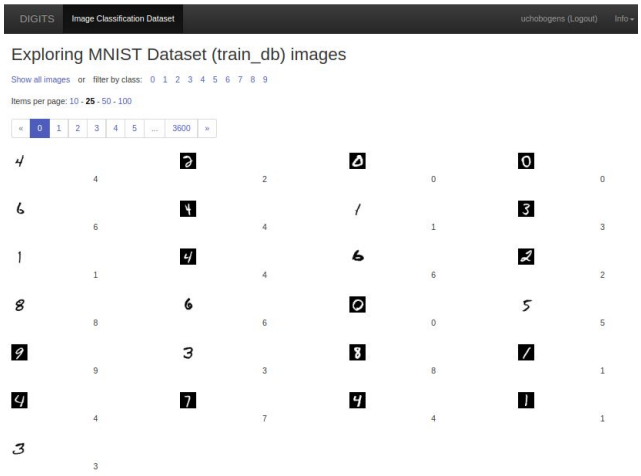

Fig 5. Exploration of MNIST Dataset.

DIGITS is now creating a data set from the train_full folder. Inside the folder there are 10 subfolders, one for each class (0, 1, 2, 3, ..., 9). All training images written by hand from '0 are in the folder' 0 ',' 1 are in the folder '1', etc.

The MNIST database of handwritten digits, available at this address http://yann.lecun.com/exdb/mnist/, has a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been normalized by size and have been centered on a fixed size image.

Now it is going to load another different database, to test another different neural network. The database to be loaded contains color images of candy boxes, juices in bottles, and empty images. All on a black conveyor belt.

Products Dataset

In the Dataset section of DIGITS, the following was provided:

- The path of the main folder of our database (/ data / P1_data)
- Color as a type of image because the images in the database are in color.
- 256 x 256 as image size.
- Products Dataset as the name of the database for DIGITS.
- Click on create

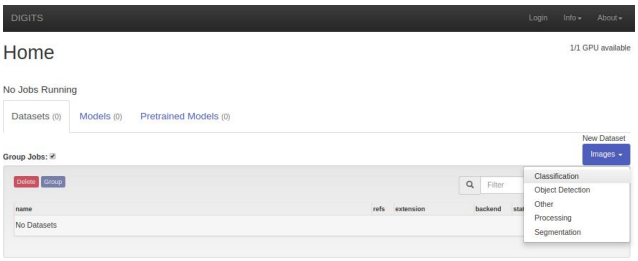

Fig 6. Dataset section. DIGITS.

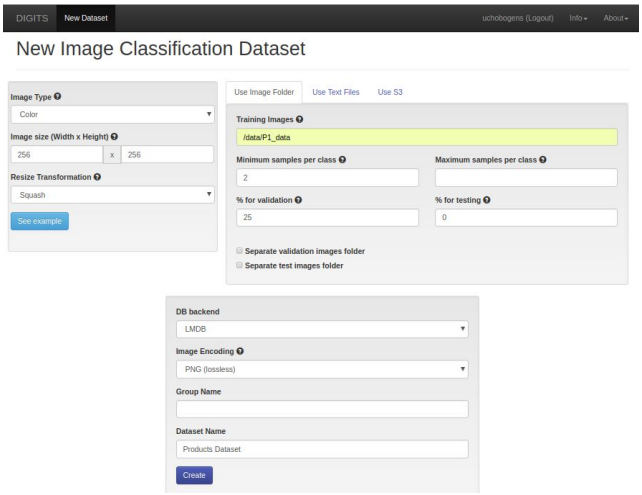

Fig 7. Dataset section. New Image Classification Dataset.

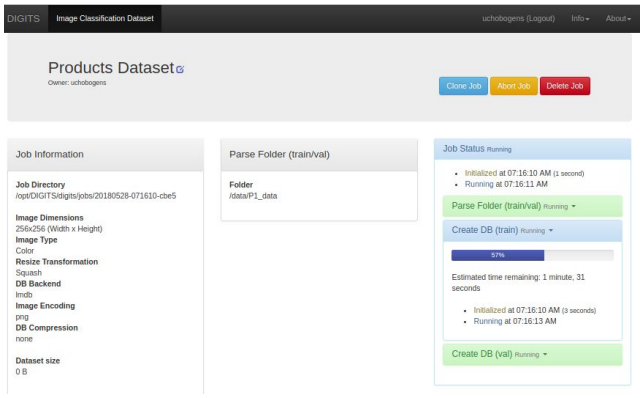

Fig 8. Products Dataset.

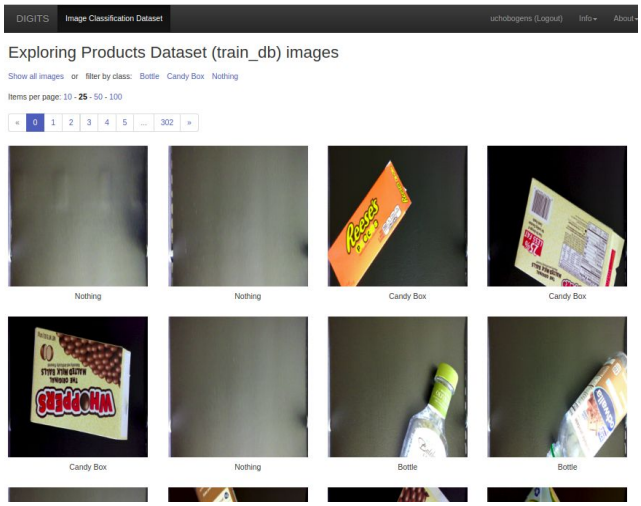

Fig 9. Exploration of Products Dataset.

DIGITS is now creating a data set from the P1_data folder. Inside the folder there are 3 subfolders, one for each class (Bottle, Candy_box, Nothing). All the training images of 'juices in bottles are in the folder Bottle', 'of boxes of sweets are in the folder' Candy_box 'and' of the black conveyor belt are in the folder Nothing '.

This database is provided by the Udacity Group (www.udacity.com) and is used to train a neural network that will be implemented in a hardware (Jetson TX2 for example) that controls a conveyor belt and an arm that is on the tape. When the tape is activated, the hardware camera is activated. If a bottle juice passes in front of the camera, the arm is not activated, if a box of sweets is passed in front of the camera, the arm is activated and the box is removed from the conveyor belt.

And to finish this section we load another different database to test another different neural network. The following database contains images of eleven toy animals and images of an empty white stage.

Animalitos Dataset

In the Dataset section of DIGITS, the following was provided:

- The path of the main folder of our database
- Color as type of image because the images of the Animalitos database are formed for the RGB color range.
- 256 x 256 as image size because it is the image size of the Animalitos database.
- Animalitos Dataset as the name of the database for DIGITS.
- Click on create

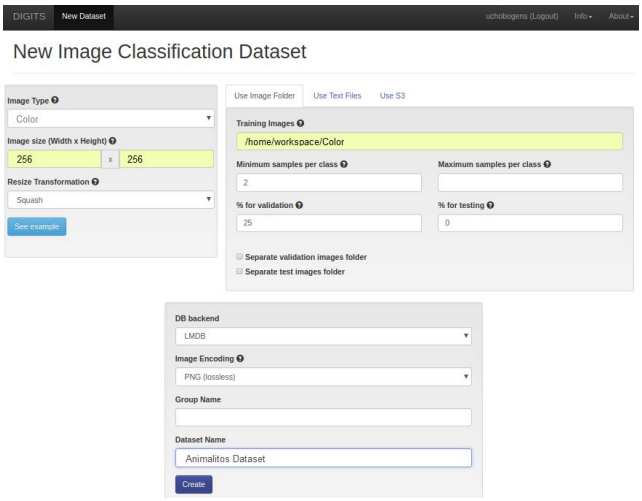

Fig 10. Dataset section. New Image Classification Dataset.

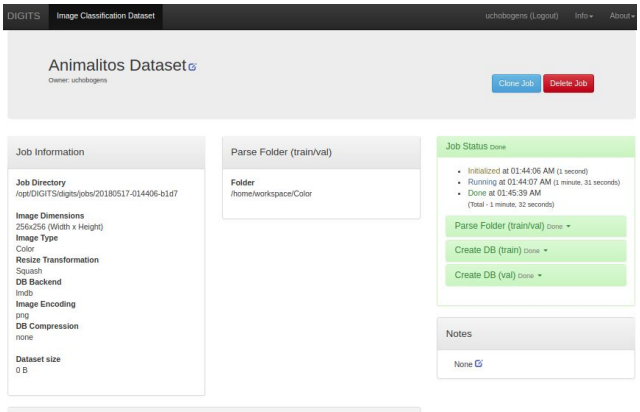

Fig 11. Animalitos Dataset.

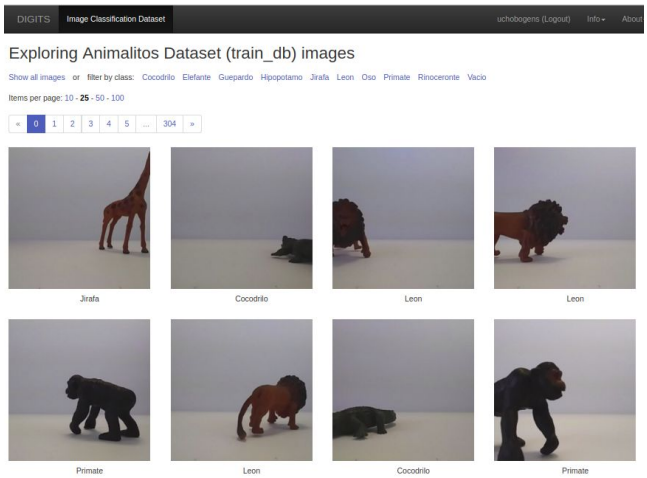

Fig 12. Exploration of Animalitos Dataset.

To create this database, a white stage was set up where the elements of the classes were placed and one by one they were taken a series of photos. The stage was armed with white anime. The elements that define the classes are 11 toy animals, that is to say that in the end I will have a folder called Animals with subfolders one for each animal. A 5 mega pixel camera of an LG K20 was used.

Fig 13. Scenario where the Animalitos model was implemented.

An application called Time Spirit was configured inside the phone to take a photo of 1280 x 720 pixel resolution every 0.5 seconds for 20 minutes. The toy animal was placed in front of the camera, the application was activated and the position of the toy was changed while the phone took the pictures for 20 minutes. This generated a video of 1 minute 20 seconds composed of 2400 photos. The same procedure was applied to the 11 toy animals.

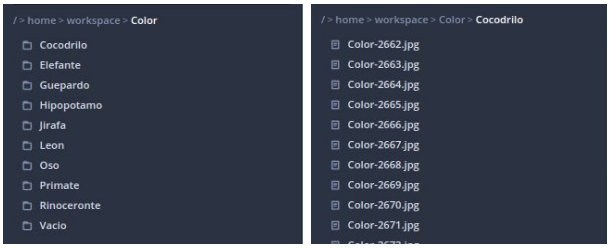

Fig 14.Preview of Animalitos Dataset.

Then the VLC Player program was used in the 11 videos to extract the 2400 frames of each video. Each group of images was examined to eliminate blurry photos and photos that did not have the object. Then Image Resizer for Windows was used to reduce the size of the images to 256 x 256 pixels which generated another group of pictures without object. These actions reduced the size of the groups to an average of 1000 photos. It is the database that was uploaded to DIGITS with the name of Animalitos Dataset.

Training / ~~(Background / Formulation)~~

Now that we have databases of handwritten numbers, the database of products, the database of toy animals and also the neural networks of the DIGITS model store; you can train the models for each database.

The next step for the training is to locate yourself in the Models section and create a new classification type model and then choose the following options:

MNIST Model

In the DIGITS model section, the following was provided:

- MNIST Dataset as the selected database.
- The number 8 as number of times for training.
- LeNet as a selected neural network.
- MNIST Model as the model name for DIGITS.
- Click on training

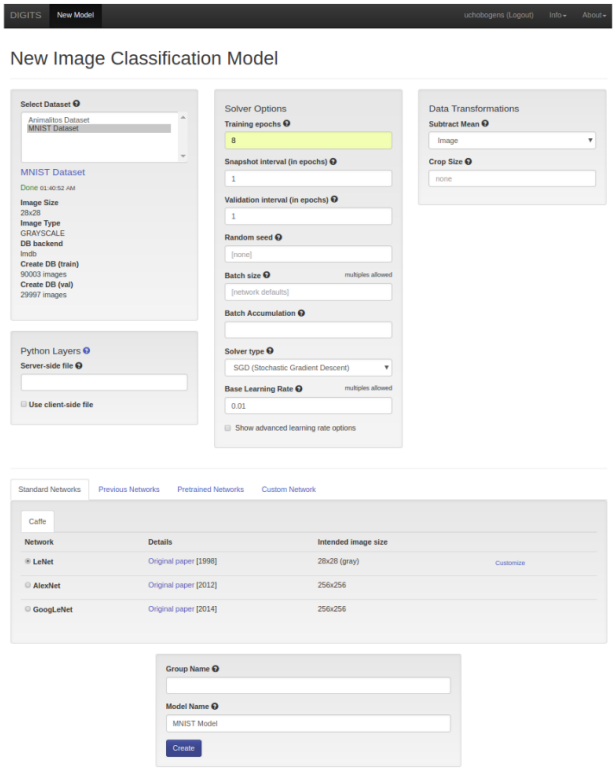

Fig 15. Model section. New Image Classification Model

A little about the LeNet model.

LeNet-5, is a pioneering 7-layer convolutional network developed by LeCun in 1998, which classifies the digits. It was applied by several banks to recognize handwritten numbers in checks (checks) digitized in gray scale input images of 32x32 pixels. The ability to process higher resolution images requires larger and convolutional layers, so this technique is limited by the availability of computing resources.

This model was chosen because it was precisely designed to recognize handwritten digits.

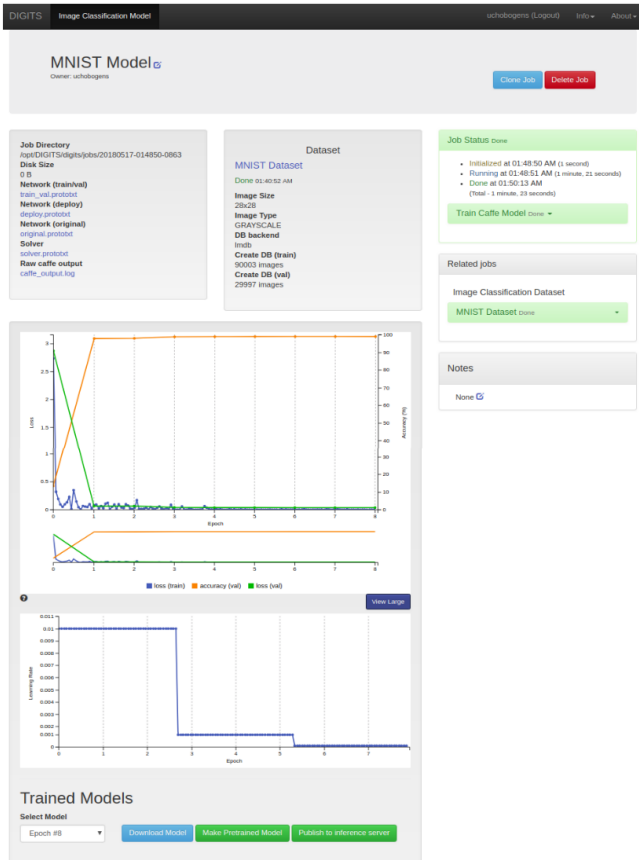

Fig 16. Training of the MNIST Model.

After training the MNIST model, a file list with the routes of different test images is tested.

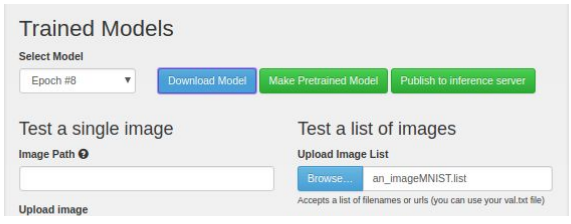

Fig 17. Options to download and test the model.

When we obtained percentages of probability close to 100% and correct labels of the test images, the model was downloaded. This action generated the named file 20180517-014850-0863_epoch_8.0.tar.gz

Now we proceed to train the neural network of the products. Located in the Models section and a new classification type model is created and then the following options are chosen:

Products Model
In the DIGITS model section, the following was provided:

- Products Dataset as selected database.
- The number 10 as number of times for training.
- AlexNet as a selected neural network.
- Products Model as the model name for DIGITS.
- Click on training

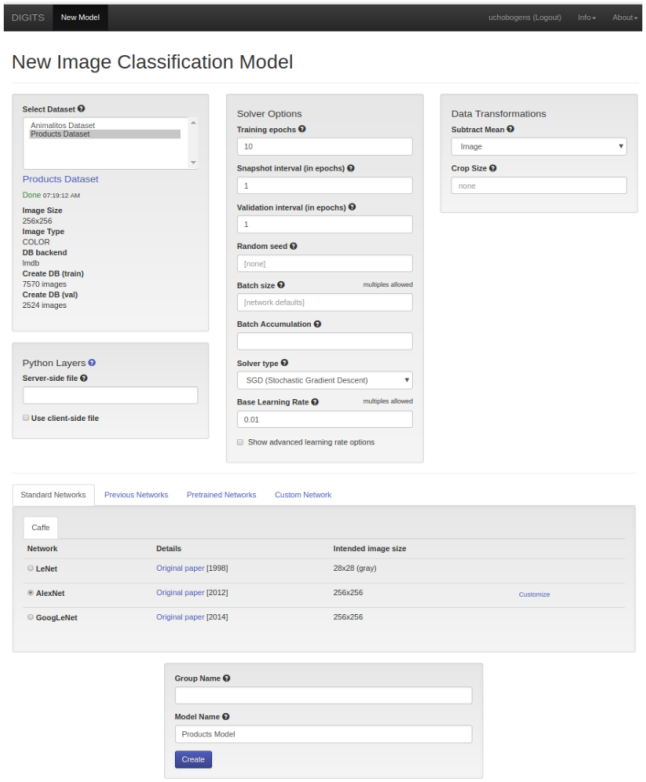

Fig 18. Model section. New Image Classification Model

A little about the AlexNet model.

This network has a very similar architecture to LeNet by Yann LeCun et al, but it is deeper, with more filters per layer and with stacked convolutional layers. It consists of convolutions 11x11, 5x5,3x3, maximum accumulation, abandonment, increased data, activations ReLU, SGD with impulse. Attaches ReLU activations after each convolutional layer and completely connected. AlexNet received training for 6 days simultaneously on two Nvidia Geforce GTX 580 GPUs, which is why its network is divided into two pipelines. AlexNet was designed by the SuperVision group, consisting of Alex Krizhevsky, Geoffrey Hinton and Ilya Sutskever.

This neural network was chosen because it is more complex and can work more complex data such as color and larger images.

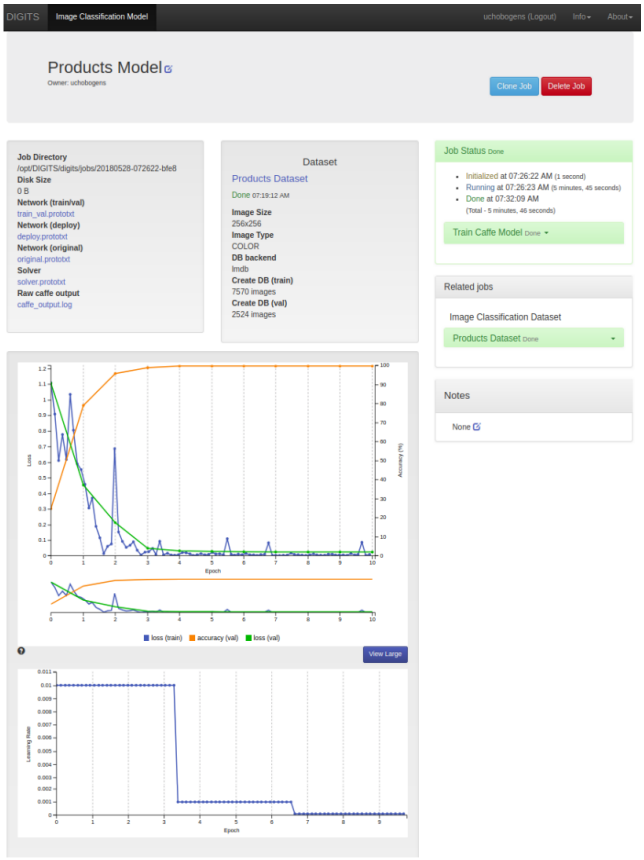

Fig 19. Model Training Products.

After the training of the model, a script was executed to determine the time of inference of the model and to obtain a value less than 10ms, the model was downloaded. This action generated the named file 20180528-072622-bfe8_epoch_10.0.tar.gz .

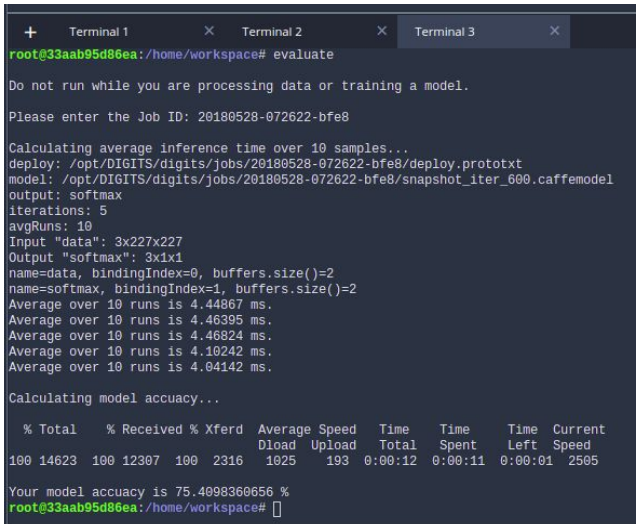

Fig 20. Evaluation of the Products Model.

For the training of the neural network of the toy animals you have to locate yourself in the Models section and create a new model of classification type and then choose the following options:

Animalitos Model

In the database section of DIGITS, the following was provided:

- MNIST Dataset as the selected database.
- Number 12 as number of times for training.
- GoogLeNet as selected neural network.
- Animalitos Med as the name of the model for DIGITS.
- Click on training

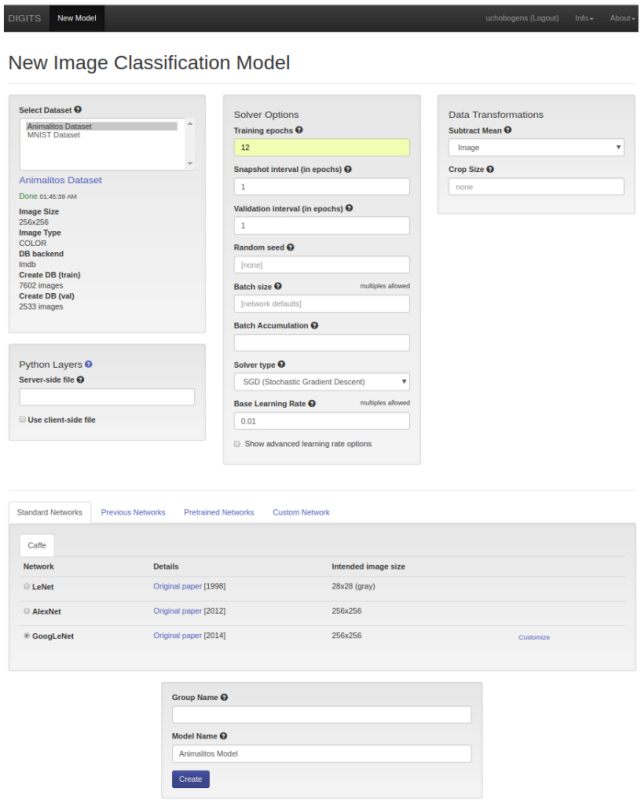

Fig 21. Model section. New Image Classification Model.

A little about the GoogLeNet model.

The winner of the 2014 ILSVRC competition was GoogleNet (also known as Inception V1) from Google. He achieved a top-5 error rate of 6.67%! This was very similar to the human-level performance that the organizers of the challenge were now forced to evaluate. It turned out that this was actually quite difficult to do and required human training to overcome the accuracy of GoogLeNets.

After a few days of training, the human expert (Andrej Karpathy) achieved a top-5 error rate of 5.1% (single model) and 3.6% (whole). The network used a CNN inspired by LeNet but implemented a novel element called the start module. Used batch normalization, image distortions and RMSprop. This module is based on several very small circumvolutions to drastically reduce the amount of parameters. The challenge of the 2014 ILSVRC competition is to evaluate algorithms for the detection of objects and classification of images on a large scale, with databases that exceed 200 classes and 50 thousand images. It is the reason why this neural network was chosen.

Fig 22. Training of the Animalitos Model.

After training the Animalitos model, a file ready with the routes of different test images is tested.

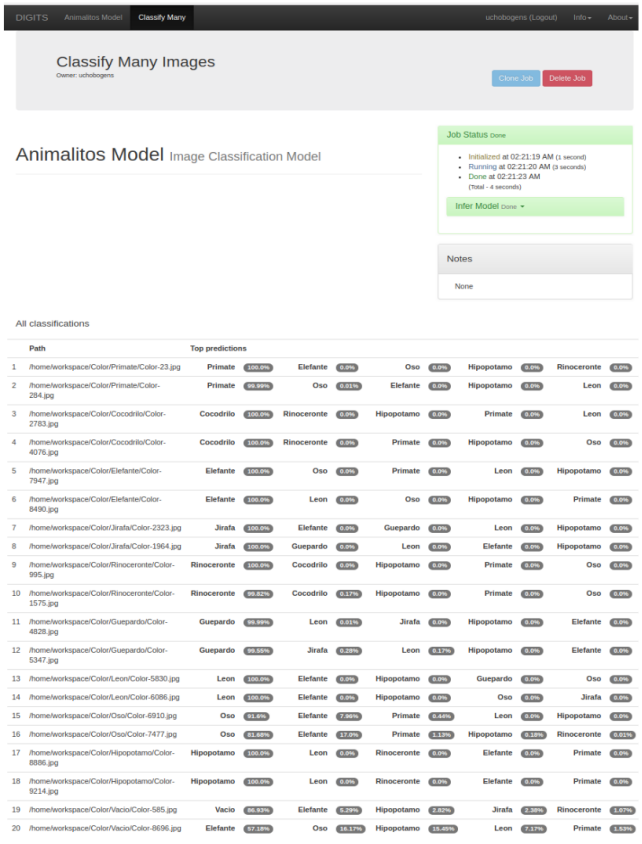

Fig 23. Results when testing the Animalitos model in different images.

When we obtained percentages of probability close to 100% and correct labels of the test images, the model was downloaded. This action generated the named file 20180528-074201-c46f_epoch_12.0.tar.gz

This model was also evaluated in its inference time, achieving a time of less than 10ms.

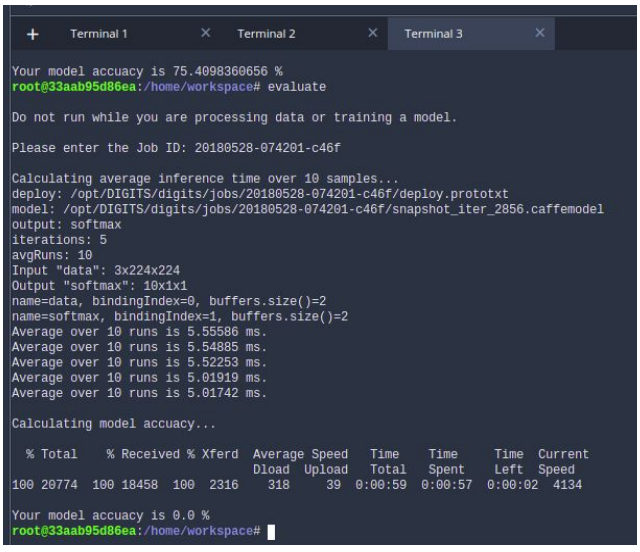

Fig 24. Evaluation of the Animalitos Model.

III. IMPLEMENTATION / ~~Results~~

An Nvidia Jetson TX2 Developer Kit was used for the implementation of the neural networks. The Jetson TX2 was installed the NVIDIA JetPack SDK which is ideal for creating artificial intelligence applications. The JetPack installer was used to update the Jetson Developer Kit with the latest operating system image, to install developer tools for both the host PC and the Developer Kit, and to install the libraries and APIs, samples and documentation needed to restart the development environment .

	Jetson TX2
GPU	NVIDIA Pascal™, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM® A57/2 MB L2
Video	4K x 2K 60 Hz Encode (HEVC) 4K x 2K 60 Hz Decode (12-Bit Support)
Memory	8 GB 128 bit LPDDR4 59.7 GB/s
Display	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4
Camera	5 MP
CSI	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.2
PCIE	Gen 2 | 1x4 + 1x1 OR 2x1 + 1x2
Data Storage	32 GB eMMC, SDIO, SATA
Other	CAN, UART, SPI, I2C, I2S, GPIOs
USB	USB 3.0 + USB 2.0
Connectivity	1 Gigabit Ethernet, 802.11ac WLAN, Bluetooth
Mechanical	50 mm x 87 mm (400-Pin Compatible Board-to-Board Connector)

Fig 25. Work table.

Fig 26. Complete set of toy animals.

After updating the Jetson TX2 system, the repository of the jetson-inference library located at https://github.com/dusty-nv/jetson-inference was cloned.

This is the configuration that was executed in a terminal of the Jetson TX2:

```
$ git clone http://github.com/dusty-nv/jetson-inferenc
$ cd jetson-inference
$ mkdir build
$ cd build
$ cmake ../
$ make
$ cd aarch64/bin
```

“bin” is a folder that contains applications and models to use the Jetson TX2 camera. In this folder there is the imagenet-camera executable that when you start it opens a window with the live video of the camera. To start it, you must reach the bin folder from a terminal and type: ./imagenet-camera.

While what is within a deep neural network can be complex, in essence, they are simply functions. They take something in and generate some results.

To successfully *implement* a trained model, there are two initial jobs.

1) The first job is to provide the model with the expected **input**.
2) The second job is to provide the end user with a **result** that is useful.

The input that the network expects is determined both by its architecture and by the way it was trained. The implementation requires writing code to convert the input to the input that the network expects. The result generated by the network is determined by its architecture and what it learned. The implementation requires writing code to convert the output that is generated to the output that the end user expects.

Both works are summarized in identifying four characteristics:

1) The entry of the *application*.
2) The entrance of the *network*.
3) The output of the *network*.
4) The result of the *application*.

If those 4 characteristics are identified, the implementation consists of writing code in any programming language from (1) to (2) and (3) to (4).

After identifying the 4 characteristics and understanding how the implementation works from the corresponding C ++ files, the next step was to extract and locate the model contained in file 20180528-074201-c46f_epoch_12.0.tar.gz. Downloaded from DIGITS at the time of finishing the training.

Note: In this section called implementation only the steps taken to implement the model of the toy animals are explained. The procedures for the model of the handwritten digits are not referenced or discussed, as they are a frequent topic in other texts and for executing the same steps that will be explained below.

Actions taken from the Jetson TX2:

(ctrl-alt-t) (To open the terminal in a window)

```
$ NET=~/jetson-inference/build/aarch64/bin/
networks/digitstojetson
$ mkdir $NET
$ tar -xvf
~/Downloads/20180528-074201-c46f_epoch_12.0.tar.
gz -C $NET
```

And finally. To execute the model:

```
$ ./imagenet-camera \
--prototxt=$NET/deploy.prototxt \
--model=$NET/snapshot_iter_2856.caffemodel \
--labels=$NET/labels.txt \
--input_blob=data \
--output_blob=softmax
```

Note: The name of the files must correspond to the name of the downloaded files.

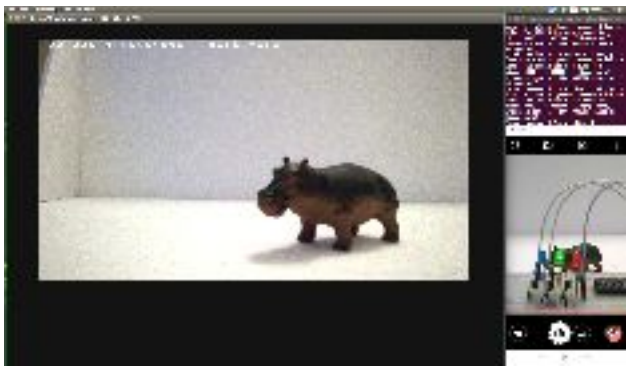

Fig 27.Screenshot of the Jetson TX2 when implementing the Animalitos Model.

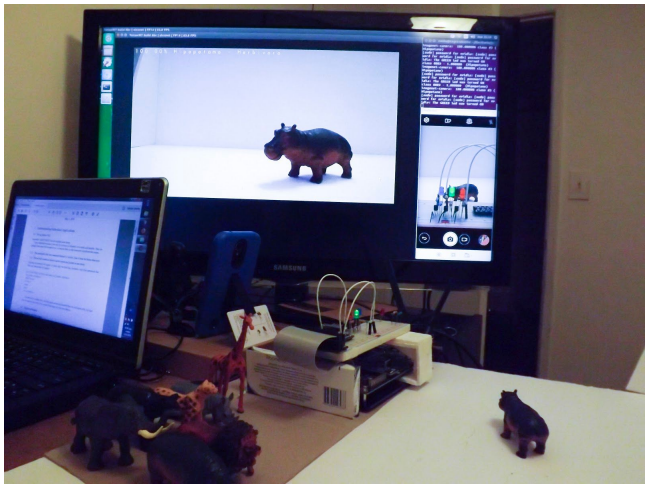

Fig 28. Photograph of the workspace at the time of implementing the Animalitos Model.

Electronic Implementation

After seeing that both models work correctly, it was decided to use the animal model to generate an electronic response in the GPIO port of the Jetson TX2 board.

The action that the JETSON TX2 will perform is to turn on a red LED if the toy animal in front of the camera is carnivorous, a green LED will light if the animal in front of the camera is of the herbivore type and a blue led if there is no animal in front of the camera.

To turn on a led from the GPIO port of the TX2 jetson, the following connection was made:

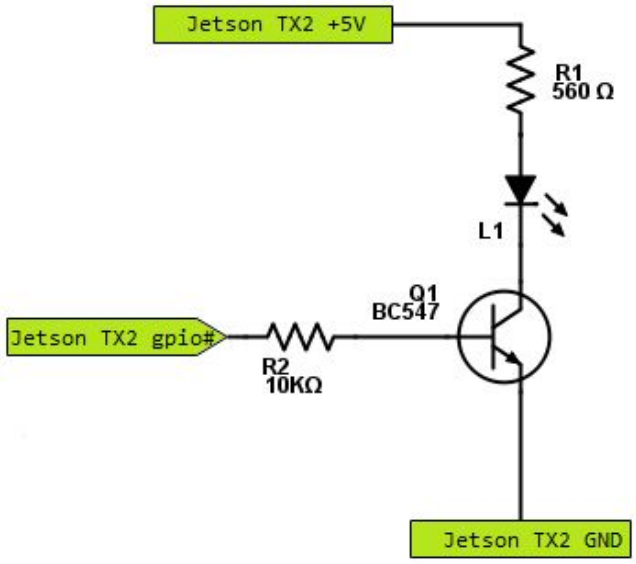

Fig 29. Circuit used to connect a Led to the Jetson TX2.

The same scheme is used for the 3 LEDs. The red led was connected to the gpio481 pin, the green led was connected to the gpio255 pin and the blue led was connected to the gpio397 pin.

Then a bash file was created for each led, where the following code was used:

```
$ sudo -s
$ cd /sys/class/gpio
$ echo 481 >export
$ echo out >gpio481/direction
$ echo 1 >gpio481/value
$ sleep 1
$ echo 0 >gpio481/value
$ echo 481 >unexport
```

This code exports or rather opens the gpio pin ###, gives an output status to the pin and then gives an output value of "1", which makes the gpio pin ### now have 5V and for turn on the led. Wait a second and then change the output value to "0", which makes the pin gpio ### have now 0V and with that the pin is turned off and to finish it closes the pin to avoid leaks or returns of currents . That is to say, for the security of the plate.

Then the `imagenet-camera.cpp` file located at *~/jetson-inference/imagenet-camera/imagenet-camera.cpp* was modified. In this file we added code to capture the label of the object that the model is identifying and then filter the label, know what kind of toy animal it is, and execute the bash file that corresponds to it.

In the world of electronics there is a saying. "If you can turn on a led you can move the world" and it is the same idea that is applied here, because if you can turn on the led you can also activate a device or control an engine, or show something by a display or what one wants to do.

And it worked. Depending on the animal / not toy animal that is in front of the camera, the jetson turns on a led.

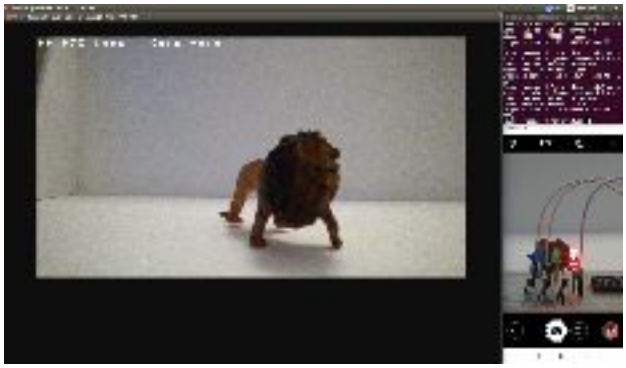

Fig 30. Label: 99.96% Leon - Carnivore. LED: Red

Fig 31. Label: 99.99% Elephant - Herbivorous. LED: Green

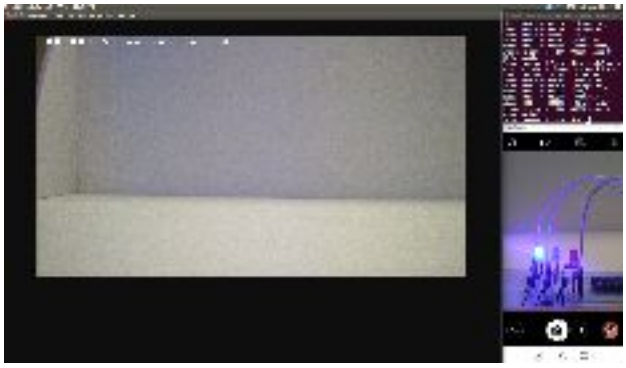

Fig 32. Label: 100.00% No-Animalito - @. LED: Blue

IV. RESULTS / ~~Discussion~~

The MNIST Productos y Animalitos models gave very good results. In the training, good accuracy of the different models was obtained.

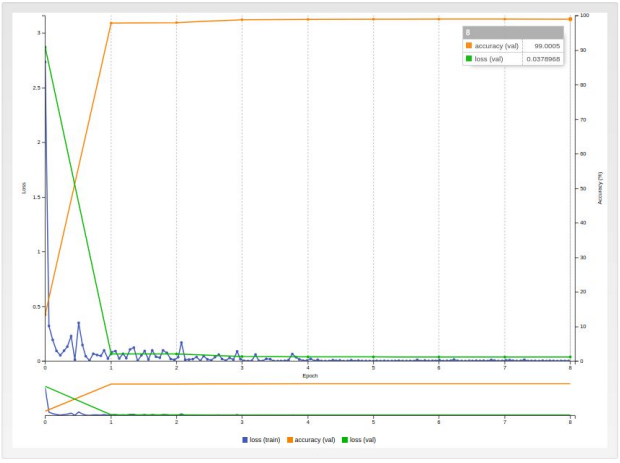

Fig 33. Training graph of the MNIST Model.
Accuracy = 99,0005. Lost = 0.0378

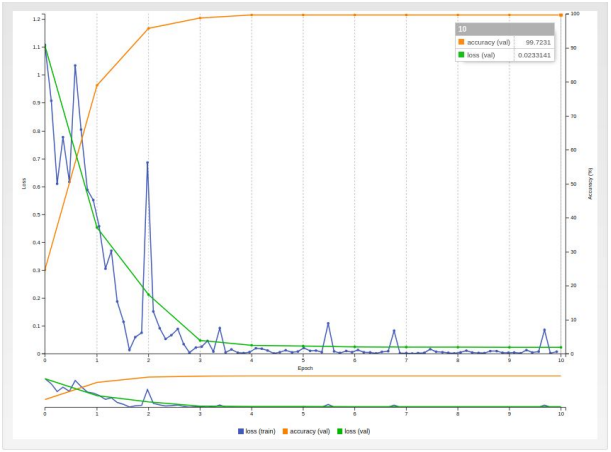

Fig 34. Training graph of the Products Model.
Accuracy = 99.7231. Lost = 0.0233

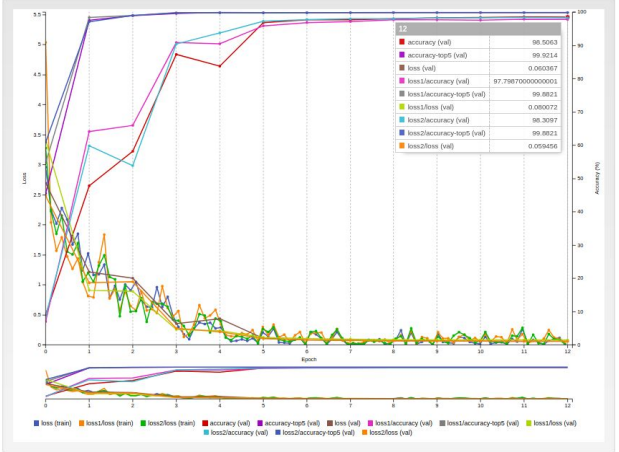

Fig 35. Training graph of the Animalitos Model.
Accuracy = 98.5063. Lost = 0.0603

The results of these graphs were then reflected in the tests of both models performed on a group of images, as illustrated in the training section.

The MNIST model was tested on 7 images giving an average accuracy of 92.75%. This average accuracy is very close to 100% and is considered an acceptable average.

The Products model was not tested but it was evaluated by a script which does not say that the model has an inference time of 4.46 ms and an accuracy of 75.41%. This average accuracy is very close to 100% and the inference time is below 10ms which are considered acceptable results.

The Animalitos model was tested on 20 images giving an average accuracy of 95.84%. This model was also evaluated in its inference time, resulting in an inference time of 5.52 ms. This average accuracy is very close to 100% and the inference time is below 10ms which are considered acceptable results.

The MNIST model only seeks to be an exercise in the recognition of handwritten digits, which is why its implementation is limited to only displaying the label and percentage of probability of the digit that is being observed.

The Products model was not implemented because it did not have a conveyor belt or a servo-motor with a bar at the end to use as an arm. For which only the Animalitos model was implemented.

The implementation of the Animalitos model consists of turning on a red led if the animal observed was a carnivore, lighting a green led if the observed animal was herbivore and lighting a blue led if there was no toy animal in the observed space. In the implementation of the model, good results were obtained. The model gave the object in front of the camera its corresponding label with accuracies above 80% and a quick response when turning on the corresponding led.

In the implementation of a model there are always two very important points to take into account. The time of inference which is the time it takes the model, determine a label. And the accuracy, which is a percentage of probability that the label is correct.

For the case of the implementation of the Animalitos model, the most important thing is the accuracy to know if the animal observed is carnivore or herbivore, or whether there is no toy animal in front of the camera. The time of inference in this project is not so important because there is a long time between the change of one toy animal to another. In other words, when the toy animal is exchanged for another toy animal, the model has identified the same object hundreds of times. Now if the object to be identified changes rapidly (like cars on a highway) then in this situation the time of inference is an important point to worry about and should be addressed.

V. CONCLUSION / ~~Future work~~

With both models the main objective was reached in the classification of objects, which is to give a correct label to the observed object, however the following considerations were taken.

The MNIST model recognizes the handwritten digits with good accuracy but being such a simple object the accuracy should be closer to 100%. To increase the accuracy the first thing that is thought is to increase the database but the database for the MNIST model is already quite large, on the other hand the 8 stages used to train the model could be few considering the number of images and classes contained in the database. To improve this model it is suggested to train it more.

The Products model was only used to test the neural network AlexNet which gave good results in its accuracy and time of inference, but this model was not implemented, so there is no comment to improve it in the future. It is only concluded that it is a more complex and useful Neural Network for large databases and with color images.

The Animalitos model also recognizes toy animals with good accuracy, and its implementation, although it is more complex, performs well but not correctly. However, the model can be improved in different aspects. The database (1000 images per class) is very small for the number of classes and the 12 periods used for training are few considering the database that should have.

In the implementation the code that turns the LEDs on and off contains a wait of 1 second to keep the led on but this generates a desynchronization between the execution of the model and the execution of the code that turns on the LED. This is an important aspect to correct within the code that executes the model and thus obtain a correct execution.

Correcting the details outlined above, it is considered that both models are commercial and functional.

As future work, we will seek to expand the scope of the Animalitos model, seeking to activate or control a more complex device than an LED in the implementation. And in terms of deep learning a project that remains pending is to train and implement a model of object detection, which remains for a subsequent publication.