Final Project Progress Report: Building a Pattern Recognition for Kitchen Utensils

Nombre: Ing. Andrés Gómez Jiménez	Carne: 200935203
Nombre: Ing. Andrés Vargas Guevara	Carne: 200527723
Nombre: Ing. Randy Céspedes Deliyore	Carne: 201054417

1 Introduction

In this document, we present the results of the progress for the Final Project of the Patter Recognition (Reconocimiento de Patrones in spanish) Course of the Master in Electronics with emphasis on Embedded Systems. The problem will be described with more detail in section 2.

2 Proposed Problem

The name of the project is "Building a Pattern Recognition for Kitchen Utensils". The recognition of different types of kitchen utensils is becoming a trending issue with the introduction of new *kitchen-aid* and *cocking* robots like the ones described in [1], [2]. The rationale behind why this problem is important lies on the fact that instead of creating custom cocking utensils, it seems the market is moving to having robots using human tools like it can be seen in figure 1.

Figure 1: Medley cocking robot. Source [2]

As a starting point, we propose to use the *Edinburgh Kitchen Utensils* [3] dataset. This dataset has a total of 897 image with 20 classes, the details of the quantity of images per dataset can be seen in table 1.

Table 1: Number of Images per class in Edinburgh kitchen utensils dataset.

Class ID	Class Name	Number of elements
1	Bottle Opener	30
2	Bread Knife	24
3	Can Opener	19
4	Dessert Spoon	33
5	Dinner Fork	59
6	Dinner Knife	51
7	Fish Slice	82
8	Kitchen Knife	39
9	Ladle	54
10	Masher	38
11	Peeler	18
12	Pizza Cutter	16
13	Potato Peeler	22
14	Serving Spoon	84
15	Soup Spoon	27
16	Spatula	53
17	Tea Spoon	105
18	Tongs	37
19	Whisk	44
20	Wooden Spoon	67
	Total	897

2.1 Dataset Preprocessing

In order to prepare the dataset, the following actions were taken:

- **Combined Classes:** several of the classes were fairly similar so it was decided to condense them into a single category. For example, the *Serving Spoon*, *Soup Spoon*, *Tea Spoon*, *Dessert Spoon*, and the *Wooden Spoon* were combined.
- **Some classes were dropped:** The following classes were dropped: *Bread Knife*, *Can Opener*, *Masher* and *Pizza Cutter*. This was mostly because their low image count or poor quality of the images themselves.
- **Remove low quality images:** a visual inspection of all images was done and several images were disposed.
- **Added new images:** several images were added in order to complete a 100 pictures per class. The images were retrieved from standard google searches of the different utensils.

The summary of resulted dataset after the processing can be seen in table 2.

2.2 Data Augmentation

As it is going to be mentioned in section 3 the selected tool for experimenting with the model creation is **PyTorch**. In order to increase the variability of the dataset it was decided to use the **torchvision.transforms** functions that are described here. The proposed configuration to use while training and testing can be seen in listing 1.

```
1
2 #For training, we apply special transformations like random rotation and horizontal flip to to data
      augmentation
3 train_transforms = transforms.Compose([transforms.RandomRotation(30),
4                                        transforms.Resize((224, 224)),
5                                        transforms.RandomHorizontalFlip(),
6                                        transforms.ToTensor(),
```

Table 2: Number of Images per class used in RPI Project

Class ID	Class Name	Number of elements
1	Bottle Opener	100
2	Dinner Fork	100
3	Dinner Knife	100
4	Fish Slice	100
5	Kitchen Knife	100
6	Ladle	100
7	Potato Peeler	100
8	Spatula	100
9	Spoon	100
10	Tongs	100
11	Whisk	100
	Total	1100

```
                                    transforms.Normalize([0.485, 0.456, 0.406],
                                                         [0.229, 0.224, 0.225])])

test_transforms = transforms.Compose([transforms.Resize((240, 240)),
                                      transforms.CenterCrop(224),
                                      transforms.ToTensor(),
                                      transforms.Normalize([0.485, 0.456, 0.406],
                                                           [0.229, 0.224, 0.225])])
```

Listing 1: TorchVision Transformation Functions

This transformation functions cause the following modifications:

- Randomly rotate images up to 30°.
- Randomly flip images horizontally.
- Convert all images to Tensors.
- Resize images to either 224 by 224 pixels or 240 by 240.
- Normalize tensor image data with specific *mean* and *standard deviation values*.

Some of the results of these modifications can be seen in figure 2.

Figure 2: Transformed Images using Torch-Vision.

3 Data Set Evaluation

In order to evaluate the dataset and as a getting started experiment, it was decided to use an existing model named **densenet121** [4] that is already created in Pytroch. This model has a total of 121 *dense layers* and is used for image classification purposes. The basic stricture of a *denseNet* model can be seen in figure 3.

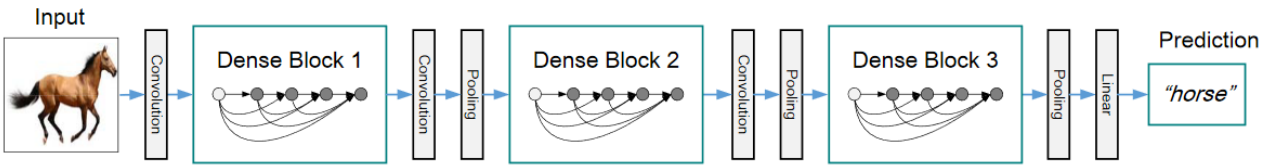

Figure 3: Model of a basic DenseNet model with three dense blocks. Source [4].

The model was trained in Google Colaboratory using a GPU during 30 epochs. The whole training process was completed in 144.5963 seconds and 80% of the images were used for training and 20% for validation. The results can be seen in figure 4.

The training loss values were obtained using the **PyTorch Negative Log Likelihood Loss (NLLLoss)** function which details can be found here. Finally the test results obtained can be seen in figure 5.

This initial experiment demonstrates that the dataset we have a t this point is a good starting point. Some extra evaluation will need to be performed on the DINNER_FORK category. As a next step, we will start developing our own model and compare it against the benchmark obtained in this initial experiment.

4 Hardware

The system chosen to capture the images and run the classification model is the NVIDIA® Jetson Nano™ Developer Kit [5] alongside the Raspberry Pi Camera Module v2 [6].

The camera was connected to the board's MIPI CSI-2 camera connector as shown in figure 6

Then the "Getting Started with Jetson Nano Developer Kit" [7] guide was followed. The image was written to a microSD card, which was later inserted in the board. Finally, after connecting the board to the power source, the screen shown in figure 7 will appear on the monitor.

On the Jetson Nano, GStreamer is used to interface with cameras [8]. And the most relevant display and capture settings can be controlled via the gst-launch-1.0 command, as shown in figure 8.

In order to fully understand the camera's capabilities, an application to control video4linux dirvers was installed: v4l2-ctl [9]. The application's output is shown in figure 9.

Finally, to better interact with the camera, Python 3.6.9 was installed (Python 2.7 is installed by default in the board's operating system). This decision is also based on the fact that all the previous testing (using Google Colab) was been done with Python 3, and we want to ensure compatibility.

```
>> Epoch: 14    Training Loss: 0.63727  Validation Loss: 0.63563
>>>>>>>> Validation loss decreased. Saving model!
DEBUG: Starting epoch 15
>> Epoch: 15    Training Loss: 0.50342  Validation Loss: 0.52900
>>>>>>>> Validation loss decreased. Saving model!
DEBUG: Starting epoch 16
>> Epoch: 16    Training Loss: 0.59572  Validation Loss: 0.80674
DEBUG: Starting epoch 17
>> Epoch: 17    Training Loss: 0.79384  Validation Loss: 0.68348
DEBUG: Starting epoch 18
>> Epoch: 18    Training Loss: 0.52743  Validation Loss: 0.56286
DEBUG: Starting epoch 19
>> Epoch: 19    Training Loss: 0.47260  Validation Loss: 0.55918
DEBUG: Starting epoch 20
>> Epoch: 20    Training Loss: 0.46124  Validation Loss: 0.97432
DEBUG: Starting epoch 21
>> Epoch: 21    Training Loss: 0.85080  Validation Loss: 0.84505
DEBUG: Starting epoch 22
>> Epoch: 22    Training Loss: 0.67474  Validation Loss: 0.57988
DEBUG: Starting epoch 23
>> Epoch: 23    Training Loss: 0.44748  Validation Loss: 0.55880
DEBUG: Starting epoch 24
>> Epoch: 24    Training Loss: 0.52501  Validation Loss: 0.67032
DEBUG: Starting epoch 25
>> Epoch: 25    Training Loss: 0.60452  Validation Loss: 0.64670
DEBUG: Starting epoch 26
>> Epoch: 26    Training Loss: 0.48032  Validation Loss: 0.53302
DEBUG: Starting epoch 27
>> Epoch: 27    Training Loss: 0.60012  Validation Loss: 0.52633
>>>>>>>> Validation loss decreased. Saving model!
DEBUG: Starting epoch 28
>> Epoch: 28    Training Loss: 0.74496  Validation Loss: 0.50126
>>>>>>>> Validation loss decreased. Saving model!
DEBUG: Starting epoch 29
>> Epoch: 29    Training Loss: 0.47401  Validation Loss: 0.52974
DEBUG: Starting epoch 30
>> Epoch: 30    Training Loss: 0.59890  Validation Loss: 0.51747
 =============== Finished Training, elapsed time: 144.59631609916687 seconds ==============
```

Figure 4: DenseNet121 Training results (30 epochs with 80% of the images).

```
Test Loss: 0.46863

Class 'BOTTLE_OPENER'. Accuracy: 100.00% (20.0/20.0)
Class 'DINNER_FORK'. Accuracy: 95.00% (19.0/20.0)
Class 'DINNER_KNIFE'. Accuracy: 55.00% (11.0/20.0)
Class 'FISH_SLICE'. Accuracy: 85.00% (17.0/20.0)
Class 'KITCHEN_KNIFE'. Accuracy: 85.00% (17.0/20.0)
Class 'LADLE'. Accuracy: 85.00% (17.0/20.0)
Class 'POTATO_PEELER'. Accuracy: 80.00% (16.0/20.0)
Class 'SPATULA'. Accuracy: 75.00% (15.0/20.0)
Class 'SPOON'. Accuracy: 100.00% (20.0/20.0)
Class 'WHISK'. Accuracy: 90.00% (18.0/20.0)

Test Accuracy (Global): 85.00 %
```

Figure 5: DenseNet121 Test Results.

Figure 6: System setup

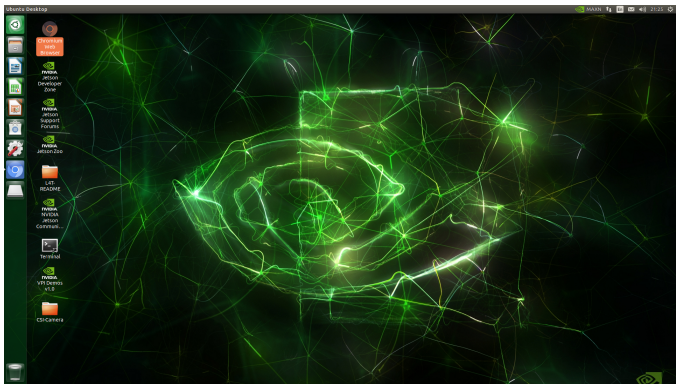

Figure 7: Linux4Tegra desktop

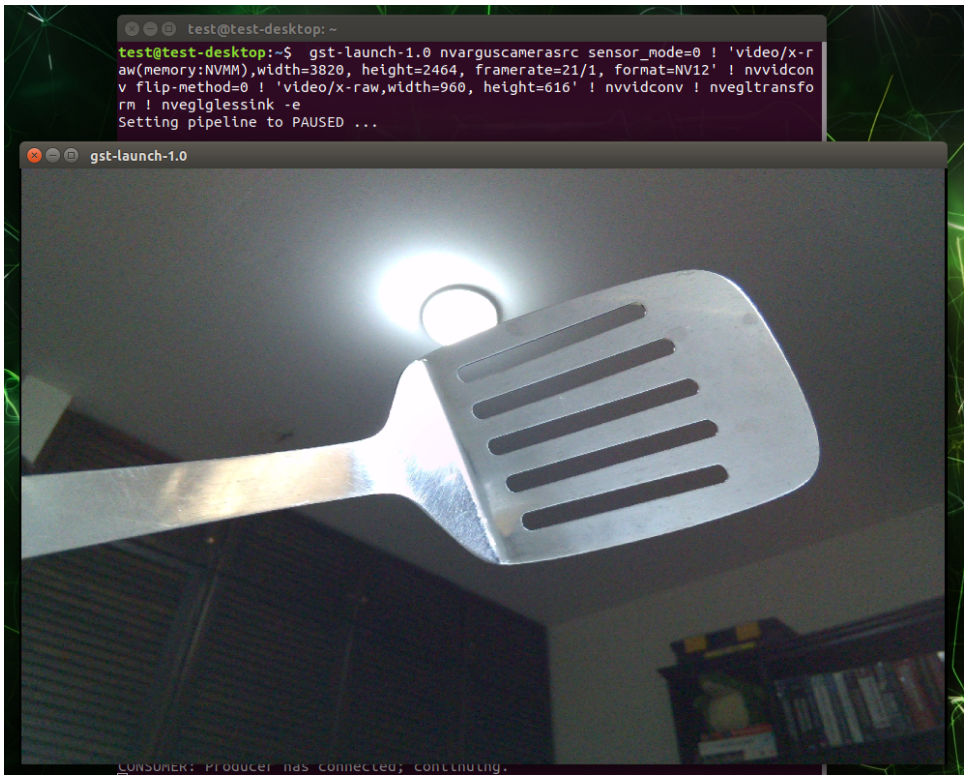

Figure 8: Launching GStreamer camera display

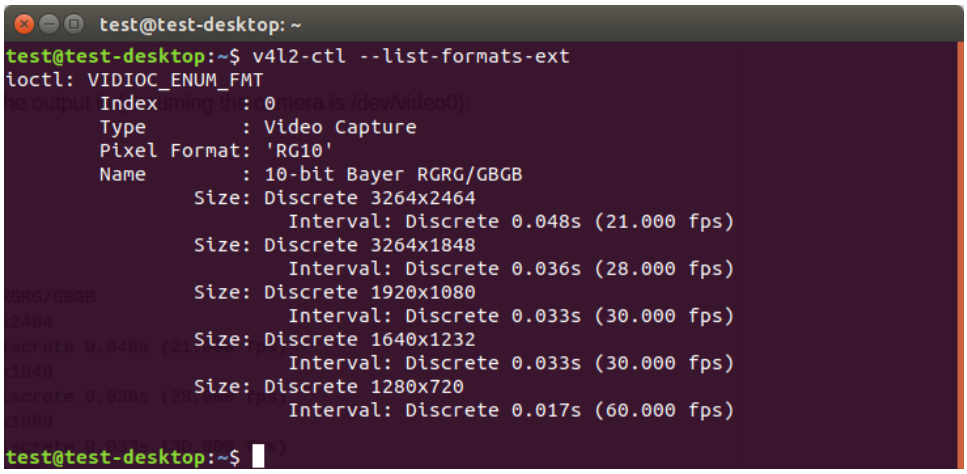

Figure 9: Displaying camera capabilities with v4l2-ctl

References

[1] R. Neate, *The robot kitchen that will make you dinner – and wash up too*, Online; accessed 19-March-2021. [Online]. Available: https://www.theguardian.com/technology/2020/dec/06/the-robot-kitchen-that-will-make-you-dinner-and-wash-up-too.

[2] M. Williams, *Robot chefs and automated kitchens*, Online; accessed 19-March-2021. [Online]. Available: https://www.herox.com/blog/352-robot-chefs-and-automated-kitchens.

[3] A. Goel and R. B. Fisher, "Classification of Kitchen Cutlery using a Visual Recognition Algorithm," no. 4, [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/UTENSILS/AGreport.pdf.

[4] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," *CoRR*, vol. abs/1608.06993, 2016. arXiv: 1608.06993. [Online]. Available: http://arxiv.org/abs/1608.06993.

[5] *Jetson nano developer kit*, Online; accessed 20-March-2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-kit.

[6] *Raspberry pi camera module v2*, Online; accessed 20-March-2021. [Online]. Available: https://www.raspberrypi.org/products/camera-module-v2/.

[7] *Getting started with jetson nano developer kit*, Online; accessed 20-March-2021. [Online]. Available: https://developer.nvidia.com/embedded/learn/get-started-jetson-nano-devkit.

[8] J. Hacks, *Jetson nano + raspberry pi camera*, Online; accessed 20-March-2021. [Online]. Available: https://www.jetsonhacks.com/2019/04/02/jetson-nano-raspberry-pi-camera.

[9] *V4l2-ctl - man page*, Online; accessed 20-March-2021. [Online]. Available: https://www.mankier.com/1/v4l2-ctl.