Kitchen Utensils Classification Using a Deep Neural Network

Andres Gomez*, Andres Vargas*, Randy Cespedes*

*Escuela de Ingeniería Electrónica, Instituto Tecnológico de Costa Rica (ITCR), 30101 Cartago, Costa Rica,
{agomez10010, andresvargasguevara, rcespedes27dds}@gmail.com

***Abstract*—This paper summarizes the work of implementing a kitchen utensils classification algorithm using a deep neural network applying transfer learning. The algorithm is trained to classify ten different categories and achieved a testing accuracy of above 90%. The model was tested in an NVIDIA® Jetson Nano™ to perform live prediction from a camera at above 4 frames-per-second.**

***Keywords*—kitchen utensils, classification, DNN, CNN, inception, transfer learning.**

I. Introduction

The recognition of different types of kitchen utensils is becoming a trending issue with the introduction of new *kitchen-aid* and *cooking* 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 cooking utensils, it seems the market is moving to having robots using human tools, like it can be seen in figure 1.

Fig. 1. Medley cooking robot. Source [2]

In this paper, we propose a deep neural network for classification of images containing certain kitchen utensils. For the scope of this project, we will propose a network that can classify a single object in an image and without any occlusion. In section II, we present the key aspects of the dataset utilized and the applied pre-processing. In section III, we present the architecture of the proposed model. In section IV we present the strategy utilized for training and testing the initial results of this model with the proposed dataset. In section V, we present the results of evaluating our model while performing a live acquisition from a camera in a Jetson-Nano embedded system. Finally, we present some general conclusions in section VI.

II. Dataset and Pre-Processing

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. However, this dataset has different problems, making it immediately unsuitable for our learning task. Below we list some of the pre-processing steps applied to obtain the dataset used for our training process:

- **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.
- **Dropped Classes:** The following classes were dropped: *Bread Knife*, *Can Opener*, *Masher*, *Tongs* and *Pizza Cutter*. This due to low image count or poor quality of the images.
- **Removed low quality images:** a visual inspection of all images was done and several images were disposed mostly due to their low quality or resolution.
- **Added new images:** several new images were added in order to complete 100 pictures per class. The images were retrieved from standard Google searches of the different utensils.

The summary of the resulting dataset after the processing can be seen in table I. As it can be observed, our proposed model will have to be capable of classifying the 10 proposed categories or classes. We also present an example image for each of the 10 classes in figure 2.

TABLE I
Final Dataset Images Count

Class ID	Class Name	Number of elements
0	Bottle Opener	100
1	Dinner Fork	100
2	Dinner Knife	100
3	Fish Slice	100
4	Kitchen Knife	100
5	Ladle	100
6	Potato Peeler	100
7	Spatula	100
8	Spoon	100
9	Whisk	100
Total		1000

Fig. 2. Example of Images in the Dataset

III. Model Structure

Since the number of available images is still relatively low to train a network from scratch (100 images per class), we propose to rely on an existing network trained with significantly more images and then perform transfer learning. As it has be proven in recent years, transfer learning is one of the most powerful tools in machine learning to solve the basic problem of insufficient training data [4]. According to [5], very few people in these days train an entire Convolutional Network from scratch (with random initialization), because it is relatively rare to have a dataset of sufficient size for this type of tasks.

The overall architecture of the proposed model is presented in figure 3. As it can be observed, we will rely on the initial layers of an Inception-v3 model, which was proposed in [6], to perform the extraction of features of our images. After this, a custom classifier with different layers is proposed to perform the required prediction for the classes in our problem.

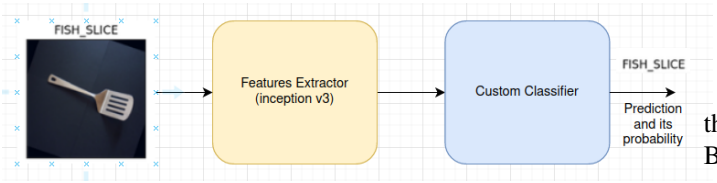

Fig. 3. Proposed Model High-Level Architecture

The features extractor of the Inception-v3 model is selected given its balance between accuracy and efficiency. Efficiency is an important aspect for our application since we intend to evaluate the model while operating with live data in an embedded board. The overall architecture of the inception-v3 model is presented in figure 4. We will rely on a pre-trained model from the ImageNet challenge [7], which contains close to 14 million of images of multiple categories. We will use this acquired knowledge to extract features from images for our specific scenario. It is worth mentioning that this pre-trained model expects images with resolution of 299×299 pixels and RGB, which means that images will need to be resized before being fed into the model.

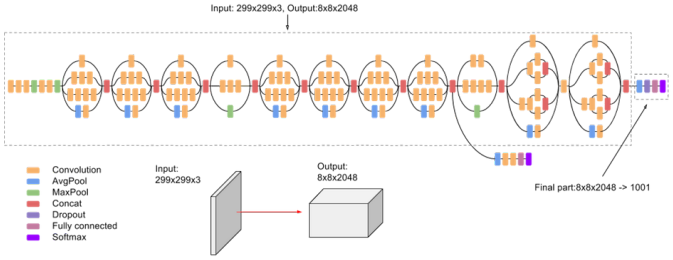

Fig. 4. Inception-v3 Model Architecture [8]

For the classifier, we drop the existing final layers in the inception-v3 model and use a custom classifier that is trained from scratch using our custom dataset. The proposed architecture for this classifier is shown on figure 5. As it can be observed, it has three fully-connected-layers that progressively down-sample the extracted features. The final layer produces 10 outputs, since this is the number of categories we intend the model to classify. A *ReLU* non-linearity is introduced after the first two layers as activation function. The final layer is a *Log Soft Max* which will provide the logarithm of the probabilities for each of the categories. The maximum number at this output can be used to determine the predicted category.

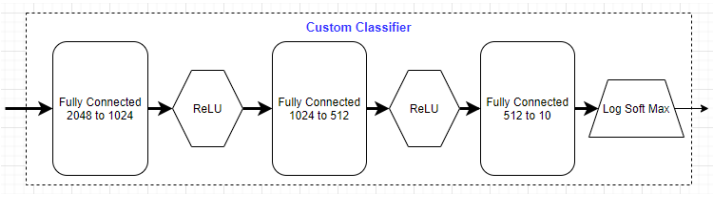

Fig. 5. Proposed Model Custom Classifier

IV. Training Process and Testing Results

We use the *pytorch* framework [9] for the definition of the model and to perform the training and testing process. Below we perform a summary of the key actions and hyper-parameters used as a part of the training process. It is worth mentioning that different hyper-parameters were used to experiment as a part of the training process. In here we describe the settings that yielded the best observed results.

- The complete dataset is randomly divided into two chunks: 80% is used for training and validation, and 20% is used to test the trained model.
- To augment the data seen by our model in each batch of the training process, we perform random rotation of the images (with a maximum of 30 degrees) and random horizontal flips. The images are initially resized to 320x320 and then cropped in their center to obtain the required 299x299 size.
- The training process is performed in batch sizes of 64 images.
- We utilize a dropout with a probability of 20% at the output of the first two *FCNs* of the classifier to reduce the likelihood of falling into over-fitting.
- We train our model for 75 epochs. The processs was observed to take between 8 and 9 minutes using a GPU in Google Collab.
- The learning rate was configured with an initial value of 0.015. However, it was also configured to decay to a 95% of its previous value in each epoch. This type of technique was key to enable an improvement of close to 6% in the testing accuracy.
- A negative log likelihood criterion is used to calculate the error as the training runs.
- A stochastic gradient descent (SGD) is utilized as an optimizer.
- The 80% of the dataset used for training, is actually divided into two separate chunks. We take 20% (from the 80%) and use it for validation as the training process runs. By doing this, as the training process goes through the epochs, we can confirm that the model is not over-fitting to the training data.
- For each epoch, we calculate the training and validation loss. The best model is obtained from the epoch that provided the lowest validation loss.

In figure 6, we can observe the obtained loss for the training and validation data against the epoch number. The loss decreases relatively fast until around the epoch 30, where it mostly oscillates around values close to 0.5. There is no clear sign of over-fitting since the training and validation loss do not strongly diverge. The best model is saved at epoch 53, where the validation loss had its lowest value of 0.497 and the training loss was of 0.469.

The obtained overall accuracy results for the saved model are displayed in table II. We can observe how the trained model was able to provide a 93.50% of accuracy in the predictions for the testing portion of the dataset.

TABLE II
OVERALL LOSS AND CLASSIFICATION ACCURACY RESULTS

Data Portion	Loss	Accuracy (%)
Training	0.469	84.53
Validation	0.497	86.16
Testing	0.290	**93.50**

In table III, we can observe the obtained accuracy results for each of the classes evaluated in the testing portion of the dataset. As it can be observed, only two classes display accuracy results lower than 90%: the dinner knife and the ladle. It was observed that the ladle instances incorrectly classified were mostly classified as spoons, due to their high level of similarity.

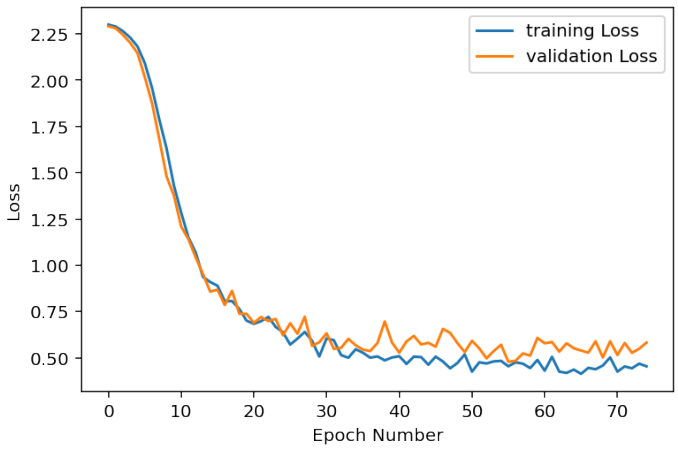

Fig. 6. Training and Validation Loss per Epoch

TABLE III
PER-CLASS TESTING CLASSIFICATION ACCURACY RESULTS

Class ID	Class Name	Accuracy (%)
0	Bottle Opener	100.00
1	Dinner Fork	95.00
2	Dinner Knife	85.00
3	Fish Slice	100.00
4	Kitchen Knife	90.00
5	Ladle	80.00
6	Potato Peeler	90.00
7	Spatula	100.00
8	Spoon	100.00
9	Whisk	95.00

V. LIVE CAMERA EVALUATION IN EMBEDDED BOARD

The embedded board used for testing is an *NVIDIA® Jetson Nano™ Developer Kit* [10] with Ubuntu 18.04 using low-memory LXDE desktop environment. The Jetson Nano™ in figure 7 has the following key features:

- GPU: 128-core Maxwell™ GPU.
- CPU: quad-core ARM® Cortex®-A57 CPU.
- Memory: 4GB 64-bit LPDDR4.

The camera used for testing was a *Raspberry PI Foundation Camera Module V2* [11] that can be seen in figure 8. This camera has a Sony IMX219 8-megapixel sensor and the resolution used for testing was 1280px by 720 px,

The model is tested from a Python script that uses *pytorch* and *OpenCV* with a *gStreamer* back-end for the acquisition. After this script is started, the camera will begin to capture and analyze the objects placed in front of it. If the object is not part of the 10 categories, "*Nothing*" should be displayed on screen. We determine that an object is non-present when the the probability of the prediction is below a 60% threshold.

Fig. 7. NVIDIA® Jetson Nano™ Developer Kit [10].

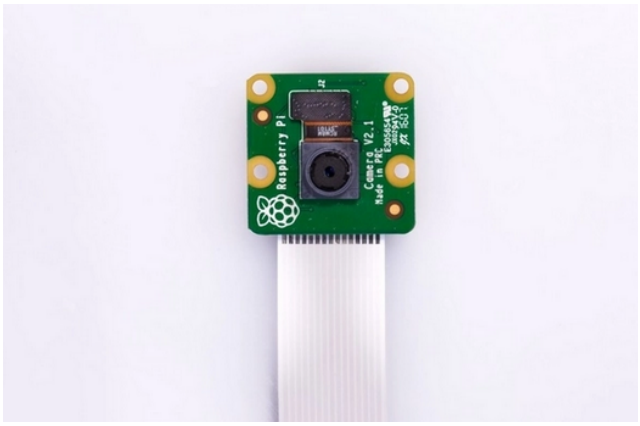

Fig. 8. The Raspberry PI Foundation Camera Module V2.

On the other hand, if the object is part of the 10 categories, the corresponding name should appear on-screen, along with the prediction probability. This can be seen in figures 9 and 10.

The model was observed to struggle with correctly classifying the objects if these were not still or if the contrast with the background was low. It was also seen to be affected by the lighting conditions. This type of behavior makes sense given the low count of images that were used for training and the limited scenarios they offered for the model to learn.

Running the model uses close to 90% of the 4GB of the NVIDIA® Jetson Nano™ and close to 50% in its four cores. The average frame rate observed was of around 4 FPS (frames per second).

VI. Conclusions

We have proposed a deep-learning model for classification of ten different kitchen utensils. The model was trained with only 100 images per category, and achieved a testing accuracy

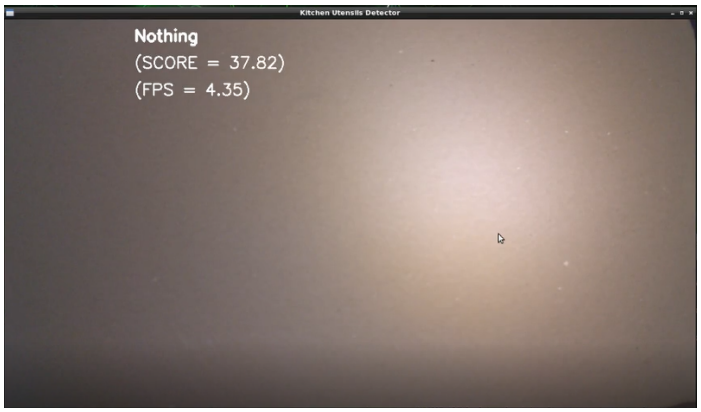

Fig. 9. Proposed Model running in NVIDIA® Jetson Nano™ with no object.

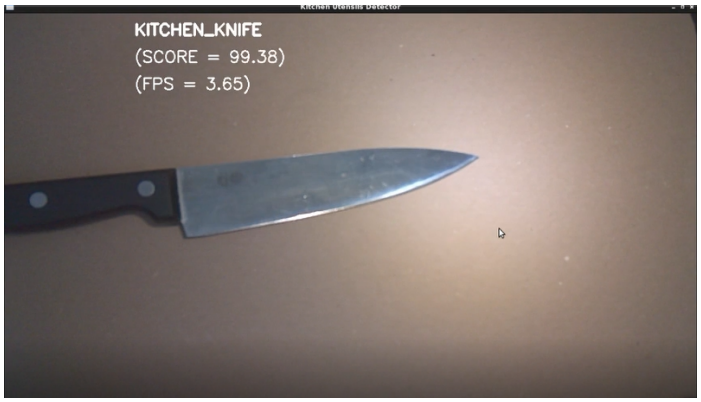

Fig. 10. Proposed Model running in NVIDIA® Jetson Nano™ with object.

of 93.5%. This was mostly thanks to the transfer learning applied from an inception-v3 model trained with the *ImageNet* dataset. Other techniques like using a validation split within the training data and a learning rate with exponential decay, proved to be useful to improve the generalization results of the model.

The model was also evaluated while performing live acquisition from a camera and using a *Jetson Nano*™ embedded board. The model was able to run at an average rate of 4 frames-per-second. It was capable of performing the correct predictions for exposed objects in the 10 categories, although it was not very robust to moving objects, low contrast scenarios and changes to lighting conditions. As a future work, the dataset used for the training process should be expanded to allow the model to generalize to more complex scenarios.

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] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, "A survey on deep transfer learning," *CoRR*, vol. abs/1808.01974, 2018. [Online]. Available: http://arxiv.org/abs/1808.01974

[5] S. University, "Cs231n convolutional neural networks for visual recognition - transfer learning." [Online]. Available: https://cs231n.github.io/transfer-learning/

[6] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the inception architecture for computer vision," *CoRR*, vol. abs/1512.00567, 2015. [Online]. Available: http://arxiv.org/abs/1512.00567

[7] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. S. Bernstein, A. C. Berg, and F. Li, "Imagenet large scale visual recognition challenge," *CoRR*, vol. abs/1409.0575, 2014. [Online]. Available: http://arxiv.org/abs/1409.0575

[8] G. Cloud, "Advanced guide to inception v3 on cloud tpu," online; accessed 20-March-2021. [Online]. Available: https://cloud.google.com/tpu/docs/inception-v3-advanced?hl=en

[9] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[10] N. Corporation, "Jetson nano developer kit," online; accessed 21-March-2021. [Online]. Available: https://developer.nvidia.com/embedded/jetson-nano-developer-ki

[11] T. R. P. Foundation, "Camera module v2," online; accessed 21-March-2021. [Online]. Available: https://www.raspberrypi.org/products/camera-module-v2/