Binary classification architecture for Edge Computing based on cognitive services and deep neural networks

Cristian Chancusig
cachancusig1@espe.edu.ec
Universidad de las Fuerzas Armadas-ESPE
Sangolquí, Ecuador

Sergio Tumbaco
sdtumbaco@espe.edu.ec
Universidad de las Fuerzas Armadas-ESPE
Sangolquí, Ecuador

Darwin Alulema
doalulema@espe.edu.ec
Applied Computing Group (TIC-211)
Universidad de las Fuerzas Armadas-ESPE
Sangolquí, Ecuador

Luis Iribarne
luis.iribarne@ual.es
Applied Computing Group (TIC-211), University of Almería
Almería, Spain

Javier Criado
javi.criado@ual.es
Applied Computing Group (TIC-211), University of Almería
Almería, Spain

ABSTRACT

Systems based on computer vision and artificial intelligence are an alternative for repetitive inspection processes. However, it is possible to extend the learning capacity of these systems to classify multiple objects using edge computing. This allows combining local processing with cloud processing to expand the possibilities and reduce the response time. In this work, a classification architecture based on remote web services and local neural networks is proposed. To test this architecture, Microsoft Azure cognitive web services and its Computer Vision API have been used, combined with the use of transfer learning and ResNet 50. The cloud service allows the identification and labelling of image content, while the Edge service, based on the neural network, allows the generation of classification models for those objects not identified or incorrectly identified by the remote service. The architecture allows to extend the possibility of image recognition by integrating web services that combined with edge processing accelerate the identification process. The proposed architecture is composed of three layers; (a) a physical layer, for the mechanical and electronic structure; (b) a logical layer, which defines the interaction of the remote and local image recognition web services, and (c) an application layer, for the integration of the monitoring and control interfaces. Finally, the architecture was evaluated through functionality testing and performance metrics of classification models, as well as load and usability testing.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; • **Software and its engineering** → *Interoperability*.

KEYWORDS

Microservices, Cyber-Physical Systems, Edge Computing, Computer Vision, Neural Network

ACM Reference Format:
Cristian Chancusig, Sergio Tumbaco, Darwin Alulema, Luis Iribarne, and Javier Criado. 2022. Binary classification architecture for Edge Computing based on cognitive services and deep neural networks. In *MEDES '22: The International Conference on Management of Digital EcoSystems, October 19–21, 2022, Italy.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3508397.3564828

1 INTRODUCTION

Vision in humans is one of the main senses, through which more than 95% of information from the environment is received. This is used to carry out processes such as classification and identification [1]. By simulating the vision of the human eye, artificial vision is born, and with it comes the desire to provide a machine with learning and planning capabilities. Within the industrial environment, the quality control of a product mimics the human processes that help to identify that product, and this is a fundamental part of a sorting system.

Sorting and classification tasks that require sorting objects according to specific characteristics are very common within companies, and their fulfilment demands the use of a large workforce. In order to meet this need, the use of computer vision and artificial intelligence technology aims to assist in these iterative processes, improving the outcome in terms of accuracy and time. The performance of a sorting system can be diverse, such as in sorting fruit based on specific characteristics [10], sorting plastics from bulk trash [25], or sorting prismatic objects, rectangular objects and coloured cylinders [5]. Even computer vision is now being combined with robotic arms [18].

Some work on Artificial Intelligence in the field of image recognition has been developed using Convolutional Neural Networks

(CNN). The use of this technology is commonly employed in industry to improve productivity by means of product classification [9] Lie2020. This technology can be developed in different programming languages [15] [3]. One of the paradigms for its implementation are web services already trained with large data records, which allow a better performance [22] because they are offered [4] as SaaS (Software as a Service), allowing applications that consume these resources and are integrated in a specific business logic [20]. For this reason, many companies are currently using classification models created by external providers [23] to create their own applications without the need to develop the internal logic of these services.

Therefore, a classifier system is useful in several types of tasks. However, it is limited to the classification of a single group of objects, and these systems cannot be generalised to the classification of new objects. In order to extend the functionality of a classifier system, where multiple objects can be classified using a single system, a classification architecture for Edge Computing based on locally executed cognitive web services and deep neural networks is proposed to extend the reconcentration options of web services and reduce the response time. This proposal contributes to the literature with an architecture for image recognition applicable in industrial environments where low response times and adaptability to various scenarios are required.

This document is divided into six sections. The first part is an introduction to understand the state of research. The second part presents some of the concepts of the state of the art. The third section is followed by the design of the architecture in which each of its components is analysed globally. The fourth section presents the implementation of the architecture. The fifth section corresponds to the validation tests of the architecture. Finally, the sixth section is about conclusions and future work.

2 BACKGROUND AND FUNDAMENTALS

The most relevant technologies used in the design of the classification architecture are listed below.

Web service: A web service provides solutions that can be consumed by different clients or other services, and can be accessed via a web protocol [20] [21].

Edge Computing: A type of processing that directs computational data, applications and services away from cloud servers to the edge of a network. Developers can use edge systems to offer services closer to users [14].

Convolutional Neural Networks (CNN): A special type of multilayer neural network [8] named as convolution. CNNs have several layers: the convolutional layer, the nonlinearity layer, the clustering layer and the fully connected layer [2]. CNN has excellent performance in machine learning problems, because they occupy image data, computer vision and Natural Language Processing (NLP).

Transfer Learning: Within machine learning the training data and the test data are taken from the same domain, resulting in the feature space we have as input and the data distribution features being the same [24].

ImageNet: A database of images organised according to the WordNet hierarchy, where each node in the hierarchy is represented by hundreds and thousands of images. The project has been crucial in the computer vision and deep learning research advancement [11].

Confusion matrix: A tool used in the field of artificial intelligence to evaluate the performance of a supervised learning algorithm. The matrix contains in its columns the number of predictions of each class determined by the algorithm, while in its rows it represents the instances in the real class, i.e. it relates the predictions of the algorithm with the correct results that were shown or should have been shown [19].

3 PROPOSED ARCHITECTURE

For a general purpose object classification system, a three-layer architecture was established: (a) a physical layer, (b) a logical layer and (c) an application layer, as shown in Figure 1.

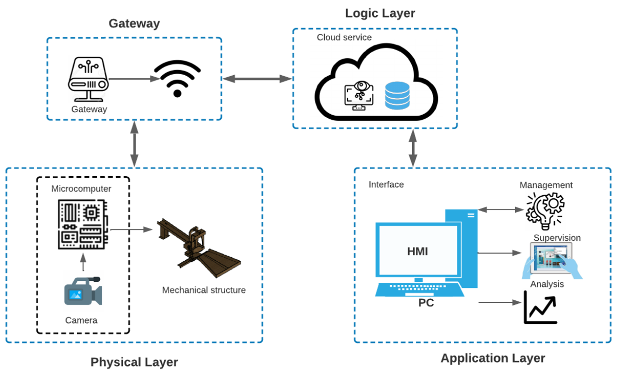


Figure 1: Classification System Architecture Design.

The physical layer addresses the selection of the type of mechanical structure, sensors, actuators and hardware elements of the system. The design of this layer sets the physical constraints for the input of objects to the sorting process, and determines the specific dimensions of the objects to be sorted.

The logic layer details the methodology used and the design of the main algorithm for object classification. Two features were determined for the design of the algorithm: (i) identification of objects through the consumption of a cognitive web service, and (ii) learning of new elements that are not possible to identify through the remote web service. For the implementation of the first requirement, the Microsoft REST API with Azure Cognitive Services was used. For the second feature, a learning model was defined using a pre-trained network, eliminating the last part of the network and replacing it with the new objects to be classified. In this way, the logical layer has a layout similar to the one shown in the Figure 2, based on a web service and the generation of an edge service. By running the edge service out of the cloud, it takes advantage of the benefits of an edge computing infrastructure [14] with the use of the service in the cloud and processing on the microcomputer locally.

The application layer is in charge of managing the system tools. This layer is subdivided into: (i) View and (ii) Controller. The user makes requests to the services from the View, and these are processed and resolved in the Controller.

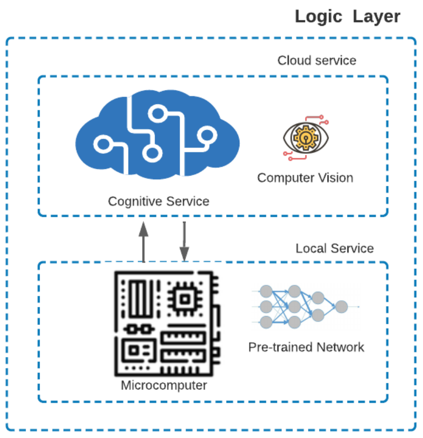


Figure 2: Logical layer of the rating system architecture.

The Controller enables the use of the cloud service and the edge service, as well as its connection to a cloud database. This first section also manages the communication technologies and protocols between the layers of the architecture. The View corresponds to the system interfaces (Monitoring and Control). As illustrated in Figure 3, the creation of two interfaces is proposed: a local interface for supervision and control, and a remote web/mobile interface for process supervision only.

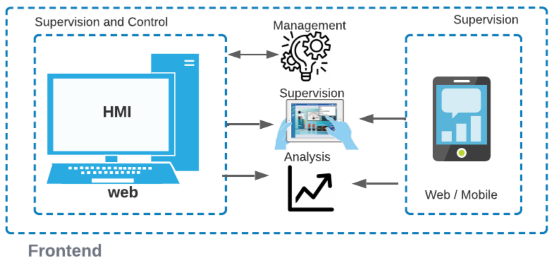


Figure 3: Application layer block content.

4 EXPERIMENTAL SCENARIO

A test scenario for vegetable and fruit sorting has been considered to validate the proposed architecture. In this sense, the architecture of the sorter has been implemented as illustrated in Figure 4. It shows the technologies, tools and protocols implemented in each layer.

4.1 Logical layer

The logical layer of the architecture is based on two services: a cloud service and an edge service. Both services work in order to obtain a classification algorithm for multiple objects in two classes defined by a user. The first service requires the consumption of the cognitive service offered by Microsoft Azure and its Computer Vision API. The cognitive service allows the identification and labelling of objects within images. For the second service of learning and generation of classification models, a pre-trained network running on the microcomputer is used.

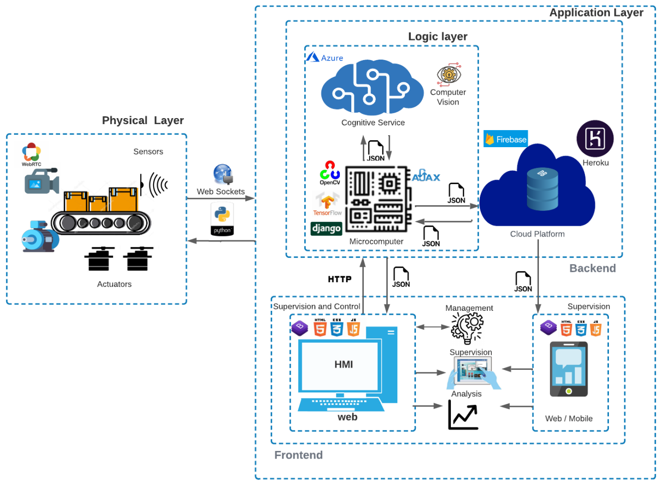


Figure 4: Classification system architecture.

a) *Cloud service:* Microsoft Azure is used as the cognitive service provider. It is necessary to create a subscription for access to a group of resources in order to consume the services, and to manage each resource individually. In this case, the cognitive service Computer Vision is used, which is offered as a FaaS (Function as a Service) of image tags to access the functionalities of the API. It provides a list of tags in JSON format that are identified as relevant to the content of the image that is captured and provided to the system. These tags are used for the identification and classification process.
The Listing 1 shows the implementation for calling this service. The tag_image operation as FaaS allows access to the Computer Vision API, which generates the list of relevant tags from the content of the image that was entered. Two input methods are supported: the first by uploading an image and the second by specifying an image URL. A successful JSON response will be returned. If the request fails, the response will contain an error code and a message [17].

Listing 1: FaaS service requesting.

```
tag_image_in_stream(image, language='en', model_version=
  'latest',custom_headers=None, raw=False, callback=None,
  **operation_config)
```

b) *Local service:* A CNN is generated following the workflow recommended by Keras for tasks using transfer learning [13]. in order to generate classification models for objects that the cloud service does not identify or identifies incorrectly.
An instance of a base model is created, and the pre-trained values are loaded. For the implementation, the ResNet50 architecture is used as a pre-trained network and the values obtained by being trained with Imagenet due to its accuracy of up to 92.1% and reduced disk size of 98 MB [12]. Listing 2 shows in the code snippet the implementation of the base model and the loading of values.

Listing 2: Training Model.

```
resnet_model = Sequential() pretrained_model =
tf.keras.applications.ResNet50(include_top=False,
input_shape=(img_height, img_width, 3), ooling=
'avg', weights='imagenet')
```

It freezes all layers in the base model configured in the previous step avoiding their retraining and modification of their values. Listing 3 shows the path of each layer of the network and its freezing in its last layer.

Listing 3: Training stage.

```
for layer in pretrained_model.layers:
 layer.trainable = False
```

A new 3-layer model is created and added to the pre-trained base model. The multidimensional input is flattened by the Flatten layer. A Relu activation function is added due to its good behaviour with images and great performance with CNN. Finally, a Softmax function is added for probability representation and generation of 2 classes as outputs. Listing 4 shows the generation of this new model.

Listing 4: Model generation.

```
resnet_model.add(pretrained_model)
resnet_model.add(Flatten())
resnet_model.add(Dense(512, activation='relu'))
resnet_model.add(Dense(2, activation='softmax'))
```

To make use of this new model, it is compiled and trained with a new dataset. Listing 5 shows the training of the model using the fit function.

Listing 5: Model training

```
resnet_model.compile(optimizer='adam',loss='categorical_
␣crossentropy',metrics=['accuracy'])
resnet_model.fit(train_ds,validation_data=val_ds,
 epochs=7)
```

4.2 Application layer

This layer addresses two blocks within the system architecture, the Controller, which is responsible for using the services generated in the logic layer and integrating them into the system. It addresses the monitoring and control interfaces of the system. Two interfaces are generated for the project: a web interface for control and supervision, which is executed locally, and a web/mobile interface only for supervision executed remotely.

a) *Controller:* The core of the system bases its programming structure on the Django framework illustrated in Figure 5. It starts with a request from a client via a web browser to the server. The View is in charge of handling the user requests that are sent between the templates and the model. The model manages the data, stores and sends the information requested by the user [6]. Within the programming structure, two applications are generated with Django, a "Cloud-Service" and a "LocalService", both with their respective models, views and templates. The first application highlights the call to the Azure API and its integration into the system by defining a function within the views. The Listing 6 below shows the API call and the storage of the prediction labels in a list.

Listing 6: Classification services.

```
with open(addressF, mode='rb') as image_stream:
tags_result_remote = cv_client.tag_image_in_stream(image_
stream, language="es")
 if (len(tags_result_remote.tags) == 0):
  aviso = "No␣objects␣detected."
 else:
  for tag in tags_result_remote.tags:
   tagsOb.append(tag.name.capitalize())
   tagConf.append(round(tag.confidence*100,2))
   global listaOb
   listaOb = tagsOb
 return tagsOb,tagConf
```

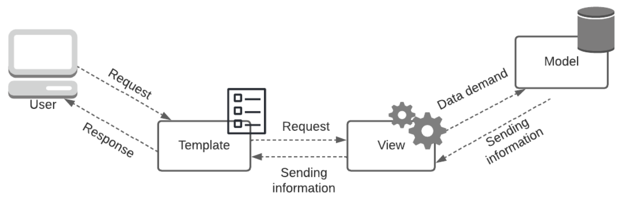


Figure 5: Project structure used by the Django framework.

For the application of the Remote service, the Listing 7 shows the capture of images as data for the generation of two repositories that in turn will serve as data sets for the creation of the classification models.

Listing 7: Remote service.

```
elif "btnCapEn" in request.POST:
 global bandFr,idCam
 bandFr=True
 camera = cv2.VideoCapture(idCam),
 img = camera.read()
 camera.release()
 if (SelecRes == "Objet␣1"):
  global contImOb1,contImOb2,dirFotoRes
  contImOb1 = contImOb1+1
  dirData = "PathSavePicObj1.jpg"
  contImOb2 = 0
  cv2.imwrite(dirData, img)
 else:
  contImOb2 = contImOb2+1
  dirData = "PathSavePicObj2.jpg""
␣␣cv2.imwrite(dirData,␣img)
␣␣contImOb1␣=␣0
␣␣listData␣=␣os.listdir(dirCarpData)
```

Once the data has been captured, the integration of the Edge service into the structure is presented. By means of Listing 8, a training function is generated, which feeds the prediction model already trained with the image captured through the tools provided by OpenCV.

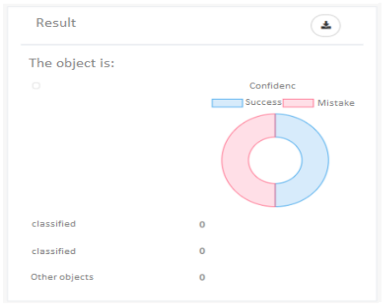
Listing 8: Edge service.

```
image = cv2.imread(direccionF)
image_resized = cv2.resize(image,(img_height,img_width))
image = np.expand_dims(image_resized, axis=0)
pred = modelo_final.predict(image)
valPredic=np.amax(np.multiply(pred,100))
valPredic=np.float64(valPredic).item()
valPredic=round (valPredic,2)
prediccionR = class_name[np.argmax(pred)]
```

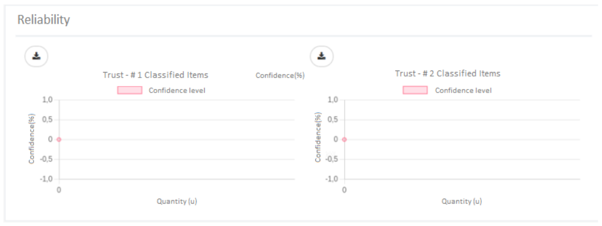
For communication between the data captured by the physical sensors of the system and the action commands sent from the interface, the Web Sockets communication protocol is used (Listing 9). The file "routing.py" is created, in which the URLs for WebSocket connections to the consumer are added [16].

Listing 9: Physical component services.

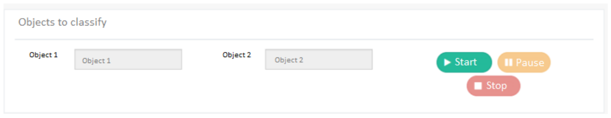
```
ws_urlpatterns = [
 path('ws/redAzure/', WSConsumerAzure.as_asgi()),
 path('ws/redResnet/', WSConsumerResnet.as_asgi()) ]
```



(a)



(b)



(c)

Figure 6: Interface sections for data entry, control and supervision of the system: (a) Monitoring and display of results, (b) Confidence graphs, and (c) Controls

The Redis channel is configured (Listing 10), in order to enable communication.

Listing 10: Physical component services.

```
CHANNEL_LAYERS={
  'default':{
   'BACKEND':'channels_redis.core.RedisChannelLayer',
   'CONFIG':{
     'host':[('127.0.0.1')]
   } } }
```

b) *View:* The local and remote interfaces were generated from templates for both services. For the local web interface, with the control and monitoring function, the following sections are generated:
- Capture and prediction of the objects to be classified in the cloud service.
- Folder creation and data capture for the training of the classification models in the local service.
- Command controls for start, pause and stop of the system.
- Video Stream of the process, data and system prediction result graphs.
- Graphs of the confidence level vs. the number of classified objects.

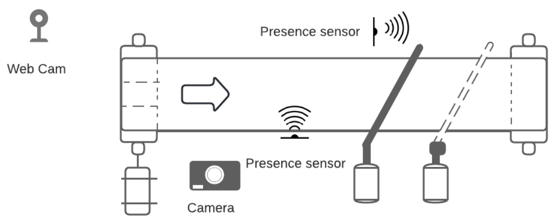


Figure 7: Layout of elements of the system model theme.

4.3 Physical layer

A conveyor belt type structure is used for Cyber-Physical System, which allows the objects to be fed in serially. Under the guidance and design and construction considerations mentioned in [7] a standard light conveyor belt was implemented. The belt runs on two end pulleys, a driving pulley and a idler pulley, as illustrated in Figure 7. The wheelbase is assumed to be 64 cm, the belt width is assumed to be 10.5 cm. The main pulley shall be connected to the engine and shall be located on the left and inlet side of the system.

The electronic sensors and actuators are distributed as shown in Figure 7. The object enters from the left side until it is detected by the first motion sensor FC51 to stop the belt, captures the image with the Rev 1.3 camera, predicts its result and continues until it is sorted by one of the two gates adapted to servomotors. It is detected at the output by the second motion sensor, concluding the sorting cycle. Figure 8 shows the layout of the prototype already implemented.

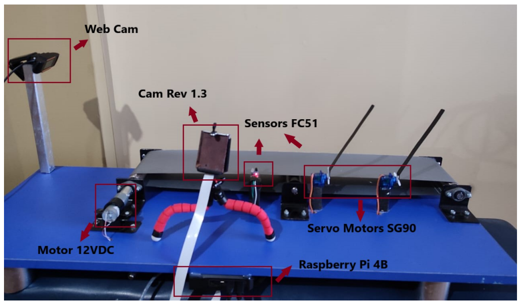
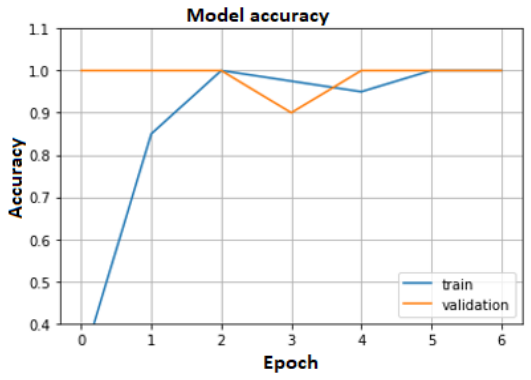
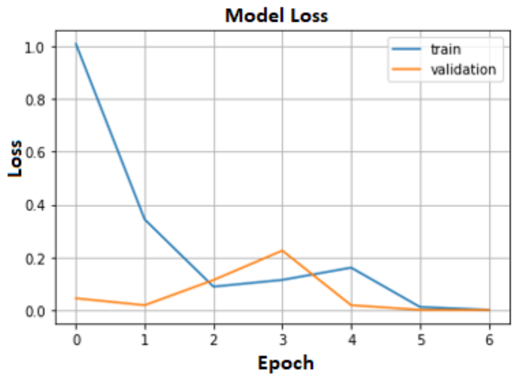


Figure 8: Real distribution of the elements implemented in the model of the clasification system.



(a)



(b)

Figure 9: Validation of the trained classification model for 50 training and validation images and 7 training epochs: (a) Performance, (b) Loss function.

5 VALIDATION

Three types of tests were carried out to validate the implemented architecture: functional tests, load tests and usability tests. Functional tests are intended to determine the performance of the classifier and its services, and are therefore evaluated by calculating the performance parameters of classification models and using confusion matrices. On the other hand, load tests are aimed at evaluating the performance of the server hosting the remote interface. It will be possible to define its access limitations by simulating different scenarios created in Gatling. Finally, the usability tests employ a usability scaling system to measure how user-friendly the classification system is through its use by various users.

5.1 Funcionality test

The generated model is evaluated with a data set of 50 images, 25 per class and a total of 7 phases, obtaining the results presented in Figure 9.a, in which for the validation and training data the accuracy tends to one.

On the other hand, the loss function tends to values of zero in the validation and training data for the generated model as illustrated in Figure 9.b. This is the value needed for training of 50 images and 7 phases.

The classifier is based on the Cloud service and Edge service, so performance tests are determined for both services. Five tests are performed for the Cloud service to validate its classification performance with objects according to their shape, texture and colour. For the local service, tests are performed with objects that have been misidentified by the Cloud service.

a) *Cloud Service:* Three test groups are set up to evaluate the performance of the cloud service. Tests with objects with similar shape and different shape. For Test 1 apples are used with pears, for test 2 fruits are used with vegetables. For the second group with objects of different texture, glass objects are used with plastic objects. For the third group with different coloured objects, red objects are used with blue and yellow objects with green.

A confusion matrix is generated with balanced evaluation data with 10 objects for each of the classes, with a total of 20 objects in each test. This is how the calculation of metrics for measuring the performance of classification models is performed. The values obtained for each case are detailed in Table 1. For the tests performed with the shape of the objects with the apple´pear and fruit-vegetable groups, the percentage of accuracy remained at 95% and the F-value also at 95%, establishing that the performance of the classification model works correctly for objects with similar characteristics to those presented in the tests.
For the test performed with the plastic-glass texture, an accuracy value of 90% and an F-value of 90% were obtained, concluding that the classification model presented by the cloud service has a performance that is considered adequate for the classification system.
Finally, for the colour tests performed for the red-blue and yellow-green groups, accuracy and F-value values of 100% were achieved, determining that the cloud service classification model works correctly to identify colours.

Table 1: Performance Parameters for the Cloud Service Rating Model (AC:Accuracy,ER: Error rate, SE: Sensitivity, SP: Specificity, PR: Precision).

		AC [%]	ER [%]	SE [%]	SP [%]	PR [%]	VPN [%]	F-Value [%]
Shape	**Test 1**	95	5	100	90	91	100	95
	Test 2	95	5	100	90	91	100	95
Texture	**Test 3**	90	10	90	90	90	90	90
	Test 4	100	0	100	100	100	100	100
Color	**Test 5**	100	0	100	100	100	100	100

b) *Service in Local:* Tests are performed on objects that the cloud system could not identify correctly. Green and red apples that are colour-independent are labelled as apples only, the group of pearl and red onions that are indistinctly labelled as onions only. Finally, passion fruit with “granadilla” are both identified as passion fruit. For these three groups their classification models obtained the values detailed in Table 2. The accuracy value and the F-value remained at 95%, only variations existed between the sensitivity and precision values. For the classification system, the classification priority of object one compared to object two is not relevant, therefore, only the F-value and accuracy are kept in consideration. It is concluded that the classification models generated for the three tests work correctly in the system.

Table 2: Performance parameters for the local service classification model (AC:Accuracy, ER: Error rate, SE: Sensitivity, SP: Specificity, PR: Precision).

	AC [%]	ER [%]	SE [%]	SP [%]	PR [%]	VPN [%]	F-Value [%]
Test 1	95	5	100	90	91	100	95
Test 2	95	5	90	100	100	91	95
Test 3	95	5	100	90	91	100	95

5.2 Load tests

This is done using a Gatling script based on the Scala language. The requests that can be sent are established, as this is a supervision page that does not have various functionalities, so in this case there are two requests that represent the number of buttons that the user can press to navigate among windows. In Figure 10, it can be seen that 200 requests have been made to the server and it has responded successfully to all of them. The response time of these requests was less than 800 ms, indicating that the server is functioning correctly.

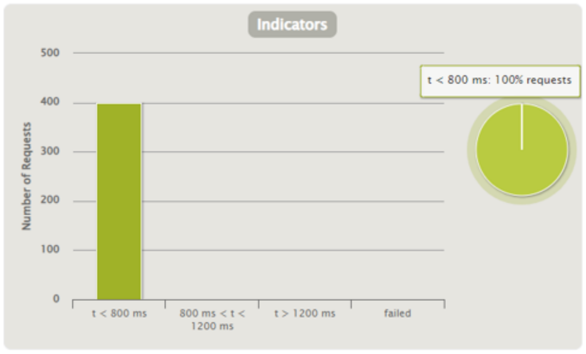


Figure 10: Simulation of 200 requests to the server.

A scenario has been simulated in which the number of users accessing the page increases for 60 seconds. The maximum number of users is 4000. After finishing the tests, the results of the tests are shown in the table 3. It can be seen that the server responds efficiently with users under 1000. However, as the number of users increases, the number of requests fails.

5.3 Usability testing

In order to verify the user-friendliness of the local and remote interface, the System Usability Scale (SUS) was used, which has

Table 3: Simulation results for load tests.

User	Total Requests	Maximum [Req/s]	Success [%]	Failure [%]	Response t<800 ms [%]	Response t>1200 ms [%]
100	200	15	100.0	0.0	92.0	8.0
200	400	8	100.0	0.0	100.0	0.0
500	1000	20	100.0	0.0	99.0	0.0
1000	2000	68	99.0	1.0	49.0	39.0
4000	8000	133	94.0	6.0	1.0	91.0

10 standardised questions. Ten users were tested and were able to use the application individually, because the local page displayed controls a single system. However, for access to the public page there was no problem with more than 2 users logging in at the same time. The answers to the questions asked can be seen in the table 4. Prior to use, a brief explanation of the operation of the system was given and it was noted that there were no problems in the use of the system. At the end of the tests, the SUS scores and the average value of 90.5/100 were obtained, which means that the system is acceptable in terms of usability, fulfilling what was expected.

Table 4: SUS results.

Qn. / U.	Qn1	Qn2	Qn3	Qn4	Qn5	Qn6	Qn7	Qn8	Qn9	Qn10	Score SUS
U1	4	1	5	1	4	1	5	1	5	1	95
U2	3	3	3	2	3	3	5	3	5	2	65
U3	4	1	5	1	5	1	5	1	5	1	97.5
U4	5	1	5	1	5	1	5	1	5	1	100
U5	4	1	4	1	5	1	4	2	5	1	90
U6	5	1	5	1	4	1	4	2	5	2	90
U7	4	1	5	1	4	1	5	2	5	1	92.5
U8	4	1	5	1	4	1	5	1	4	1	92.5
U9	5	1	4	1	5	1	4	2	5	2	90
U10	5	1	4	1	4	1	5	2	5	1	92.5
									Average		90.5

6 CONCLUSIONS AND FUTURE WORK

The classification model employed by the Cloud service presents accuracy and F-value values between 90% and 100%, determining that the performance of its classification process within the system is correct for groups in which the objects to be classified belong to a more general category. Likewise, the classification models generated by the local service show accuracy and F-value values of 95%, concluding that the performance is adequate when the objects to be classified belong to more specific categories or with more distinctive characteristics.

The Django framework used for the programming project enabled the integration of the system's tools and services. At the same time, the use of the Heroku platform as a service (PaaS) allowed the Django application to be deployed on a cloud server, generating a public link for access to the remote web/mobile interface for system monitoring. The use of the Google FireBase platform and its Cloud Storage services for image storage and RealTime as a database gave the system the capacity to store information and synchronise data in the control and supervision interfaces.

Future work includes: (i) Bringing the task of generating classification models to the cloud, thereby expanding the data set that can be input, as well as the phases used for training. (ii) Furthermore, we propose the use of techniques such as Web Scraping to access the Google Colab platform and the use of virtual machines to generate the model and download it to the microcomputer, ensuring that it is only necessary to load it and use it in the system; (iii) The local interface is executed within a local server generated by Django, and the idea is to take it to a server in the cloud and have access to control the machine from a remote location. (iv) Finally, it is proposed to extend the system by increasing the number of classes for each classification model, for which it will be necessary to modify both the services and the mechanical structure.

ACKNOWLEDGMENTS

This work has been funded by the EU ERDF and the Andalusian Government (Spain) under the project UrbanITA, PY20_00809.

REFERENCES

[1] Aguila Antonio Zárate, Aguila Reyes Alejandro, Servando Ariel, Aguila Zárate, and Mendoza Blanco Fernando. 2009. *Análisis de lA visión humana orientado al diseño arquitectónico analysis of human vision orientated toward architectur Al design.* Technical Report 1. http://erevistas.saber.ula.ve/index.php/ecodiseno/article/view/3888/0

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. 2018. Understanding of a convolutional neural network. *Proceedings of 2017 International Conference on Engineering and Technology, ICET 2017* 2018-January (mar 2018), 1–6. https://doi.org/10.1109/ICENGTECHNOL.2017.8308186

[3] Jarosław Bernacki. 2021. Robustness of digital camera identification with convolutional neural networks. *Multimedia Tools and Applications* 80, 19 (jul 2021), 29657–29673. https://doi.org/10.1007/S11042-021-11129-Y/TABLES/10

[4] Devshree Bharatia, Priyanka Ambawane, and Piyush Rane. 2019. Smart Electronic Stick for Visually Impaired using Android Application and Google's Cloud Vision. *2019 Global Conference for Advancement in Technology, GCAT 2019* (oct 2019). https://doi.org/10.1109/GCAT47503.2019.8978303

[5] Alex Eduardo De La Cruz and Juan Francisco Donoso. 2016. *Diseño y construcción de una máquina didáctica clasificadora de objetos mediante visión artificial para el Laboratorio de Automatización Industrial de Procesos Mecánicos de la Facultad de Ingeniería Mecánica.* Ph. D. Dissertation. EPN, Quito. https://bibdigital.epn.edu.ec/handle/15000/16486

[6] Django. 2022. The web framework for perfectionists with deadlines. https://www.djangoproject.com/

[7] Forbo. 2022. Recomendaciones para la construcción de instalaciones. (2022). https://forbo.blob.core.windows.net/forbodocuments/7378/305_fms_recomendaciones_para_la_construccion_de_instalaciones_es.pdf

[8] Anirudha Ghosh, Abu Sufian, Farhana Sultana, Amlan Chakrabarti, and Debashis De. 2020. Fundamental Concepts of Convolutional Neural Network. *Intelligent Systems Reference Library* 172 (2020), 519–567. https://doi.org/10.1007/978-3-030-32644-9_36

[9] A Gnana, Selva Kumar, S Aathisha, S Dharani, and N Revathi. 2019. Machine Vision Technique Based Smart Fruit Sorter. (mar 2019). https://doi.org/10.1109/JSEN.2016.2580221

[10] Christian Montoya Holguín, Jimmy Alexander Cortés Osorio, and José Andrés Chaves Osorio. 2014. Automatic recognition system of fruits based computer vision. *Ingeniare* 22, 4 (oct 2014), 504–516. https://doi.org/10.4067/S0718-33052014000400006

[11] ImageNet. 2021. ImageNet. https://www.image-net.org/

[12] Keras. 2022. Keras Applications. https://keras.io/api/applications/#usage-examples-for-image-classification-models

[13] Keras. 2022. Transfer learning fine-tuning. https://keras.io/guides/transfer_learning/

[14] Wazir Zada Khan, Ejaz Ahmed, Saqib Hakak, Ibrar Yaqoob, and Arif Ahmed. 2019. Edge computing: A survey. *Future Generation Computer Systems* 97 (aug 2019), 219–235. https://doi.org/10.1016/J.FUTURE.2019.02.050

[15] Hoshang Kolivand, Saba Joudaki, Mohd Shahrizal Sunar, and David Tully. 2021. A new framework for sign language alphabet hand posture recognition using geometrical features through artificial neural network (part 1). *Neural Computing and Applications* 33, 10 (aug 2021), 4945–4963. https://doi.org/10.1007/S00521-020-05279-7/FIGURES/14

[16] A Melnikov and I Fette. 2011. The WebSocket Protocol. https://datatracker.ietf.org/doc/html/rfc6455

[17] Microsoft. 2022. Computer Vision documentation - Quickstarts, Tutorials, API Reference - Azure Cognitive Services. https://docs.microsoft.com/en-us/azure/cognitive-services/computer-vision/

[18] Guillermo Moncada. 2018. *SISTEMA ROBÓTICO CLASIFICADOR DE OBJETOS CON VISIÓN ARTIFICIAL.* Ph. D. Dissertation. Universidad Técnica Federico Santa María. https://repositorio.usm.cl/bitstream/handle/11673/42363/3560901544160UTFSM.pdf?sequence=

[19] José Manuel Sánchez Muñoz. 2016. Análisis de Calidad Cartográfica mediante el estudio de la Matriz de Confusión. *Pensamiento matemático* 6, 2 (2016), 9–26.

[20] José Salmerón Rubio. 2020. *Desarrollo de aplicación web basada en FaaS con .NET Core Evolution desde aplicación monolítica.* Ph. D. Dissertation. UNIVERSIDAD POLITECNICA DE MADRID.

[21] S. Subbulakshmi, Anvy Elsa Saji, and Geethu Chandran. 2020. Methodologies for selection of Quality Web Services to Develop Efficient Web Service Composition. *Proceedings of the 4th International Conference on Computing Methodologies and Communication, ICCMC 2020* (mar 2020), 238–244. https://doi.org/10.1109/ICCMC48092.2020.ICCMC-00045

[22] Mohammad Moeen Valipoor and Angélica de Antonio. 2022. Recent trends in computer vision-driven scene understanding for VI/blind users: a systematic mapping. *Universal Access in the Information Society* 1 (feb 2022), 1–23. https://doi.org/10.1007/S10209-022-00868-W/FIGURES/7

[23] Tom Vermeire, Dieter Brughmans, Sofie Goethals, Raphael Mazzine Barbossa de Oliveira, and David Martens. 2022. Explainable image classification with evidence counterfactual. *Pattern Analysis and Applications* 25, 2 (may 2022), 315–335. https://doi.org/10.1007/S10044-021-01055-Y/TABLES/5 arXiv:2004.07511

[24] Karl Weiss, Taghi M. Khoshgoftaar, and Ding Ding Wang. 2016. A survey of transfer learning. *Journal of Big Data* 3, 1 (dec 2016), 1–40. https://doi.org/10.1186/S40537-016-0043-6/TABLES/6

[25] Chen Zhihong, Zou Hebin, Wang Yanbo, Liang Binyan, and Liao Yu. 2017. A vision-based robotic grasping system using deep learning for garbage sorting. *Chinese Control Conference, CCC* (sep 2017), 11223–11226. https://doi.org/10.23919/CHICC.2017.8029147