UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Neural networks for image classification using the CIFAR-10 dataset

Computational Intelligence (MAI): Final Project

Professors

NEBOT CASTELLS, Angela
ROMERO MERINO, Enrique

Team members

BEJARANO SEPULVEDA, Edison Jair
edison.bejarano@estudiantat.upc.edu
REYES FERNANDEZ, Grecia
grecia.carolina.reyes@estudiantat.upc.edu
LIHONG, Ji
lihong.ji@estudiantat.upc.edu
XIAO, Fei
xiao.fei@estudiantat.upc.edu

Polytechnic University of Catalonia
C/Jordi Girona Salgado,1-3 08034 BARCELONA Spain
Master's degree in Artificial Intelligence
Barcelona, 2021

Contents

Abstract

The main purpose of this project is two fold: first, it was intended that each and every one of the team members acquired the knowledge to be able to understand and implement different image classification models while seeking to enhance accuracy, and second, based on the different models that were created, their performances are assessed and compared. In order to meet these objectives, the CIFAR-10 dataset, consisting of 60,000 tiny images was used to perform both the training and the testing of the models. The Computational Intelligence (CI) methods that are explored in this work fall under the field of Neural Networks. Within this category, the present work leverages Convolutional Neural Networks (CNN) in particular. On the one hand, a CNN model was built from scratch and on the other, two pre-trained deep architectures of CNNs, VGG16 and ResNet50, were leveraged by means of transfer learning with the aim of comparing the results obtained with the different models. To carry out the comparison of the models, the best performing ones were selected and uploaded to Kaggle, where their scores were obtained. Our results showed that, although it was possible for us to obtain values of accuracy as high as 0.8717 and a loss value of 0.4298 by creating and training our own CNN model, there are great advantages to leveraging transfer learning techniques, since they provide an agile way of obtaining very good image classification models at a lower computational cost. In contrast to the previous value, the best of the models that were created using transfer learning (here referred to as *ResNet50 - Model 2*) had an accuracy of 0.9573 and a loss value of 0.1749.

Keywords: Computer Vision, Image Classification, Convolutional Neural Networks, Transfer Learning, CIFAR-10, VGG16, ResNet50.

1 Problem statement and goals

1.1 Introduction

It is well known Artificial Intelligence can be defined as the ability of machines to learn a variety of tasks that imitate the use of reasoning in the human brain. Also the machines are computer system developed that perform by perception, classification and continuous learning logic to improve and later interact with reality [1]. Another important concept that emerges when is questioning what is AI, is the concept of learning, this concept refers to the gain of knowledge by means of reasoning beginning with some data and based on previous experiences to predict information and make decisions about some particular task, all of this has as result developing of wisdom [1].

The next great variant that allowed the evolution and implementation of intelligent models was the new era of big data, and thanks to the rapidly increasing computational performance, and new machine learning algorithms, many new AI tools have emerged in recent years with the objective to better understand and interpret our environment. These new techniques that are constantly been created and perfected are part of everything that currently constitutes the branches of artificial intelligence [2]. These are: Neural Networks (NN), Machine Learning (ML), Deep Learning (DL), Computer vision, Natural Language Processing (NLP), Natural Language Generation (NLG), Recommender Systems, Predictive Analysis. Some of these branches or technique are used for object recognition in images and usually are:

- Computer vision: It Is considered as the ability to interpret our world in visual perception to the machines by techniques of image prepossessing, and with that develop the wisdom to understand the content of digital images such as videos or photographs. There are many applications and types of tasks were developed inspired by computer vision, some of them are in object Classification,

identification, detection, segmentation and recognition.

The computer vision problem must be solved as steps for interpretation, and in some cases are computationally very expensive when working with large volume of images.

- Neural networks: Neural Networks are complex structures made of artificial neurons, the neurons are units that have the ability to process information and when it is connected between multiple of them is obtained as result a network with the functionality of receiving multiple inputs to produce a single output. Neural networks can be used, for example to recognize images. In each layers of the network, the system analyzes the images, looking for patterns such as color, edeges or shapes. Depending on the type of learning that has been applied to them and the purpose for which they are to be used, there are several neural network structures: Feedforward, Recurrent and Convolutional neural networks. In this work we will use the Convolutional ones.
- Deep learning: The concept deep always is mentioned to refer to the number of hidden layers in the neural network. As is mentioned in [3], deep learning includes methods such as convolutional neural networks (CNNs), or deep belief networks and variations thereof. Convolutional Neural Networks have a big role in deep learning, and the way it works is basically because are multiple convolutional and pooling layers that help to deep learning model to extract the most relevant features from visual datasets. Thanks to the robust type of architecture that can be created with the multiple layers of convolutions, CNN can learn a strong hierarchy of feature extraction that is not affected by rotation, translation, or space and instead generates an optimal.

In this project has as its main purpose the object recognition in images for the CIFAR-10 dataset. For this, section 2 presents the CI methods where the concepts and architectures used are explained, in the section 3 experiments and evaluation: the preprocessing of the data, the evaluation and results of the different configurations for the CNN architectures (own implementation), VGG16 and ResNet50 are explained, these two last used them as backbone. To conclude, section 5 Discussion and section 6 Conclusions and future work.

1.2 Goals

The main purpose in this project is to use the CIFAR-10 dataset to develop some system that made object recognition in images. The secondary objectives are:

- Developed a CNN system for object recognition in images for the CIFAR-10 - Dataset
- Implemented ResNet-50 and adapt it for object recognition in images for the CIFAR-10 - Dataset
- Implemented VGG-16 and adapt it for object recognition in images for the CIFAR-10 - Dataset
- Compare all the previous models to observe the performance, advantages and disadvantages between all of them
- Obtain the differences in creating a CNN from scratch and using transfer learning

2 CI methods

For the object image recognition task, the neural computation was implemented. Because it was considered that neural networks with the convolutional concept may be the best way to solve this problem. These kinds of neural networks mimic the processing behavior of the human brain and the field of computer vision is inspired in the eye, with this last, it is possible to give the machine the ability to interpret our reality in a similar way as we do us when we use our sight.

2.1 Architectures

2.1.1 FCNN

CNN is chosen to realize the image classification. The architecture is defined relying on the traditional style: with the stage increasing, the number of the filters increases in the convolutional layer, also there is an activation function following the convolution processing. Then a maxpooling function is connected with the convolutional layer. The result, processed by the combination of the convolutional layer and maxpooling, would be considered as the input of a fully-connect neural network(FCNN) and an activation function 'ReLu'. At last, a dense network in class number and 'softmax' are applied on.

2.1.2 VGG16

The VGG16 Architecture was developed and introduced by Karen Simonyan and Andrew Zisserman from the University of Oxford, in the year 2014 [5]. The VGG16 model achieved 92.7% top-5 test accuracy in ImageNet. The ImageNet dataset contains more than 14 million annotated images according to the WordNet hierarchy [6], it has become increasingly common to use it to training CNN to learn new low-level features of the images (colours, edges, textures) while keeping the knowledge of the objects and shapes. VGG16 is used in many deep learning image classification techniques and is popular due to its ease of implementation, is why we choose this architecture as a base model. It's a 16 layer deep network, The input to the convnets is a fixed-size 224 x 224 RGB image, the image is passed through a stack of convolutional layers with a small filter of 3×3 and there is also 1x1 filters acting as linear transformation of the input. Then, the hidden layers uses ReLu. It have 3 fully connected layers, the first two have 4096 channels each, and the third have 1000 channels, one for each class. In total it uses around 14 million parameters.

2.1.3 ResNet50

Deep learning algorithms use different architectures, one of the most popular and efficient is ResNet50. Residual Neural Network (ResNet) architecture that introduces a solution for gradient fading and allows convolutional networks to be deeper, since intuition indicates that as an algorithm stacks a greater number of layers, it should be more capable of extracting characteristics and offer better results for a specific task[7]. For this project it was used transfer learning technique with ResNet50 architecture to use the pretrained weights with the ImageNet image dataset. This architecture is our base model, it has 50 layer deep architecture trained with 1000 categories of millions of images and 23,5 million parameters, it's well known for good generalization and fewer error rates on recognition tasks.

2.2 Transfer Learning

It is considered as a machine learning method where the starting point is with pre-trained models, because the learning or bias in these neural networks, as well as their architecture, can be recycled and adapted to different problems with the desired approach.
The way of this methodology works is just to reuse the lower layers of the previous model because, in this part of the model, we have the weights and features trained and learned, then, we freeze those layers and it is removed the top layers to maintain the weights. Next, it is added new layers on top of the configuration desirable to adjust the new model for the new task. With this only is necessary to train the top layers to adjust the weights for the problem and at the same time is fed for the experience or

the weights of the other model. It is possible to train this new model with a few epochs obtained in this method one powerful and economic method to resolve our different tasks in less time and resources.

2.3 Optimizer

Stochastic gradient descent, the most common algorithm used in training neural networks for finding the minimum of a function. It is commonly used in deep learning models to update the weights of a neural network through backpropagation. Optimisers try to improve the amount of information used to update the weights, mainly through using previous (and future) gradients, instead of only the present available gradient. In this work, SGD, Adam and RMSprop have been used, however there are another good optimizer.[8]

- **SGD** Updates the current weight w_t using the current gradient $\partial L/\partial w$ multiplied by some factor called the learning rate, α. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily
- **RMSprop** Root mean square prop or RMSprop is another adaptive learning rate that tries to improve AdaGrad. Instead of taking the cumulative sum of squared gradients, it take the exponential moving average of these gradients.
- **Adam** Adaptive Moment Estimation is another method that computes adaptive learning rates for each parameter. In addition to storing an exponentially decaying average of past squared gradients like RMSprop, Adam also keeps an exponentially decaying average of past gradients, similar to momentum. In other words, is simply a combination of momentum and RMSprop and it converges faster than the others optimizers and takes advantage of the AdaGrad and RMSProp algorithms, the algorithm calculates the exponential moving average of the gradient, and the beta1 and beta2 hyperparameters control the rate of decline of these moving averages.

2.4 Performance Metrics of Quality

At the moment that is created, trained, and tested our model, it is vital to obtain metrics to obtain information on how well it is working and at the same time measure the performance in our models (this during training and testing). Some of these metrics are:

- **Confusion matrix** It is a performance measurement for classification problem where output can be two or more classes. Estimates of the possibilities of classification models are derived from the expressions true positive (TP), true negative (TN), false positive (FP), false negative (FN), which exist in the confusion matrix. It is useful for measuring Recall, Precision, Specificity, Accuracy, and AUC-ROC curves [9].
- **Accuracy** From all the classes (positive and negative), how many of them we have predicted correctly, so is calculated as the sum of two accurate predictions (TP + TN) divided by the total number of data sets (P + N). The best accuracy is 1 and the worst is 0 [9].
- **Cross-Entropy** It is a lost function, it works well for multi-class classification, measures the performance of a classification model whose output is a probability value between 0 and 1. It increases as the predicted probability diverges from the actual label.[10]
- **Precision** When we have a class imbalance, accuracy can become an unreliable metric for measuring our performance. Therefore we need to look at class specific performance metrics too. This metric can be explained by saying, from all the classes we have predicted as positive, how many are actually positive. [9]

- **Recall** Is also one of the important metric, it is the proportion of actual positive cases which are correctly identified.
- **F1-score** Is a combination of two important error metrics: Precision and Recall, and it helps to measure Recall and Precision at the same time.[9]

3 Experiments and Evaluation

3.1 Dataset: CIFAR-10

CIFAR-10 was collected by A. Krizhevsky [11], V. Nair and G. Hinton. The dataset containing 60,000 images of 32×32 pixel color photographs, these are very small images. CIFAR-10 have 10 classes,the class labels and their standard associated integer values are below, (0: airplane,1: automobile,2: bird,3: cat, 4: deer,5: dog, 6: frog, 7: horse, 8: ship,9: truck). This dataset is widely used for benchmarking computer vision algorithms in the field of machine learning.

3.2 Data Pre-Processing

We started with splitting the data x_train, y_train, and x_test and y_test (Table 1). Its three color layers (R, G and B) are maintained, but they will be resized to ensure that they all have the same dimensions and, in this way, the proposed CNN can operate for each of the samples.

set	images	shape
x_train	50,000	(32x32x3)
x_test	10,000	(32x32x3)

As a second step, a normalization stage into range (0,1) in 'float32' type. And finally convert class vectors into the one-hot encoding type, which used to be column vectors and describe the type with positive integers.

Data Augmentation: Is a popular technique used by deep learning to increase the generalizability of a possibly overfitting model. By generating additional training data and exposing the model to different versions of data with transformations and size within the same class, thus the training process becomes more robust and is more likely to generalize the resulting model to the test set.[12] This technique has demonstrated good performance as compared to explicit data augmentation that directly transforms the input data. In this paper we use some important transformations like:

- **Rotation:** Rotation augmentations are done by rotating the image horizontally or vertically on an axis between 1 and 359 degrees, we use both.
- **ZCA Whitening:** It is an image preprocessing method that leads to a transformation of data such that the covariance matrix is the identity matrix, leading to uncorrelated features.
- **Flipping:** Horizontal axis flipping is much more common than flipping the vertical axis, but in this case we use both. This augmentation is one of the easiest to implement and has proven useful on datasets such as CIFAR-10 and ImageNet.
- **Rescale:** The image can be scaled outward or inward. While scaling outward, the final image size will be larger than the original image size. Most image frameworks cut out a section from the new image, with size equal to the original image.

3.3 CNN Model

In this architecture, the convolutional window size is scaled as $3 * 3$, and 'ReLu' is selected as the activation function after convolution. We use the default configuration of maxpooling. In FCNN part, the number is set as 128, and 'ReLu' is selected as the activation function for the FCNN. The end part is a combination of 10 dense network and 'softmax' function. Some other techniques are utilized in the architecture to optimize the model. We add a 'Batch_Normalize' function after each convolutional layer. And a 'Dropout' function is added after the pooling stage. As a result, this architecture is defined and the specific information shown as Figure 1:

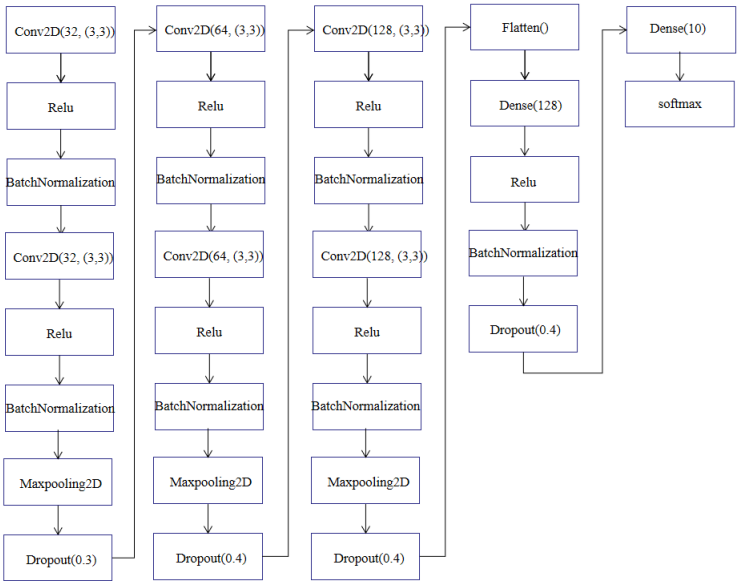

Figure 1 – CNN Architecture Design

As for the hyperparameters, these are set as: batch_size=32, epochs=100. The configuration of this Convolutional neural network has 551,722 parameters trainables and 1,152 Non-trainable. Also as is shown in the figure 19 in appendix A.

3.3.1 Evaluation

Considering the hyperparameters: batch_size=32, epochs=100. As for the training part, the type of optimizer and loss-function and learning rate are the three most import components for model training. For this architecture, we prepare some choices for each component: optimizer:[Adam, RMSprop, SGD], loss-function:[crossentropy], learning rate:[10e-3, 10e-4, 10e-5]. The loss and accuracy in training and testing are shown in from Figure 2 to 10.

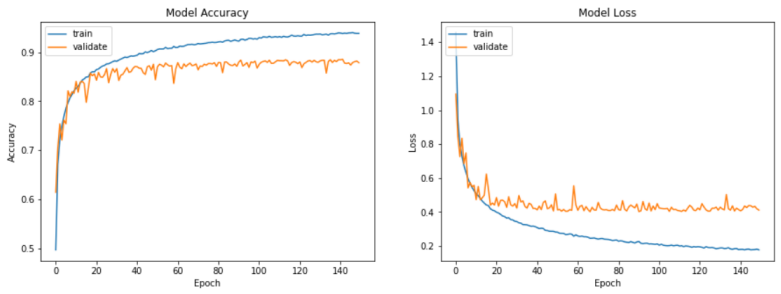

Figure 2 – Cate_Crossentropy,RMSprop,lr=10e-3

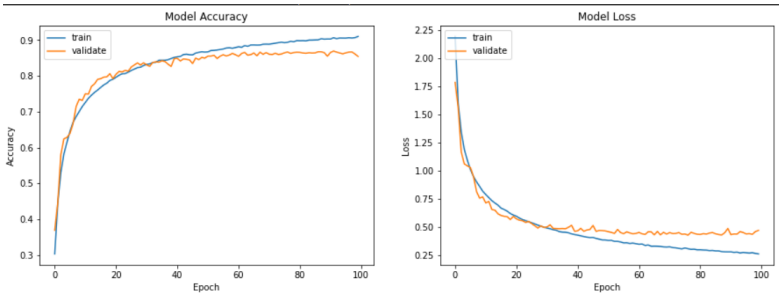

Figure 3 – Cate_Crossentropy,RMSprop,lr=10e-4

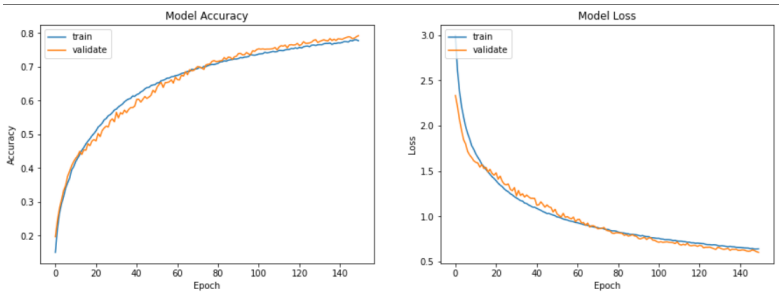

Figure 4 – Cate_Crossentropy,RMSprop,lr=10e-5

For the RMSprop optimizer, better performance is obtained when we make the learning rate smaller. For 10e-3 there is a difference between the results of the train (0.95) and test (0.87), likewise it doesn't seen improvement since epoch 20. For 10e-4 the difference between the results of the train and the test is shortened, and it stabilizes at epoch 50. When the learning rate is 10e-5, a better loss is obtained, however the accuracy of the test is lower than the others (0.79), in the this configuration a better performance is obtained.

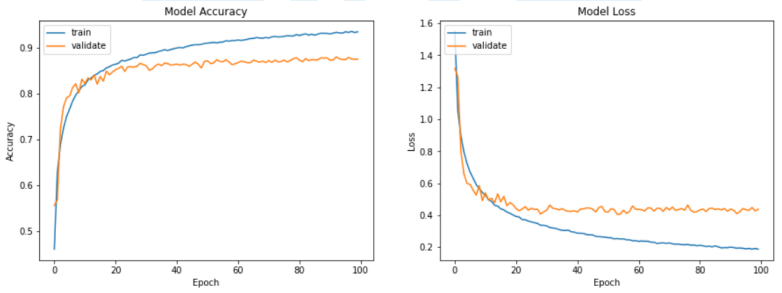

Figure 5 – Cate_Crossentropy,Adam,lr=10e-3

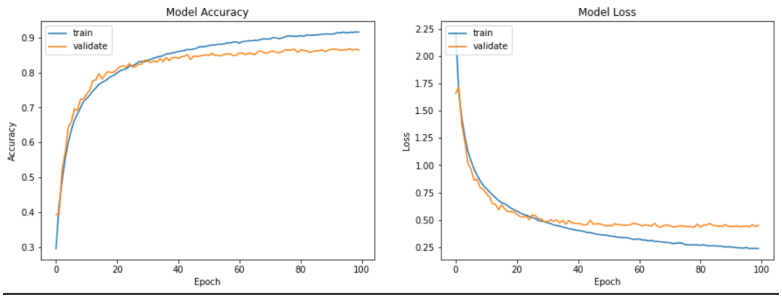

Figure 6 – Cate_Crossentropy,Adam,lr=10e-4

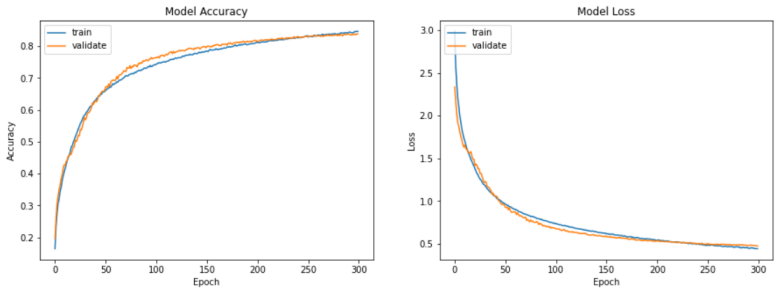

Figure 7 – Cate_Crossentropy,Adam,lr=10e-5

Using Adam optimizer and learning rate 10e-3, about accuracy, we obtain 0.92 in train and 0.85 in validate, so we have 0.07 of difference, the difference increases since epoch 20. About loss, we get 0.19 in train and 0.44 in validate, so we have 0.25 of difference, the difference start increasing in at epoch 19. Using learning rate 10e-4, about accuracy, we obtain 0.90 in train and 0.85 in validate, so we have 0.05 of difference, the difference increases since epoch 24. About loss, we get 0.25 in train and 0.50 in validate, so we have 0.25 of difference, the difference start increasing in at epoch 24. Finally, using learning rate 10e-5, about accuracy, we obtain 0.81 in train and 0.81 in validate, so we have very little difference and it's stable in every epoch. About loss, we get 0.48 in train and 0.48 in validate, so we have very little difference and it's stable in every epoch. The best results, was with learning rate 10e-4 where best results in validate and quite good in overfitting.

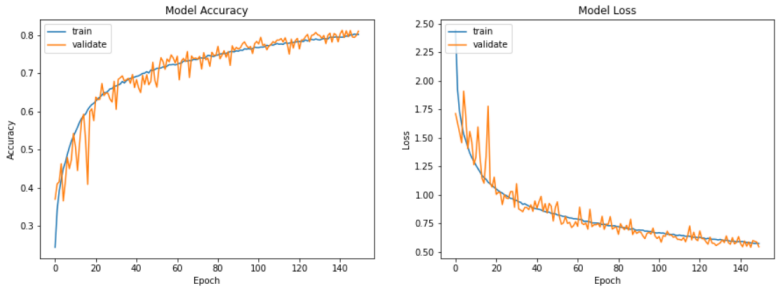

Figure 8 – Cate_Crossentropy,SGD,lr=10e-3

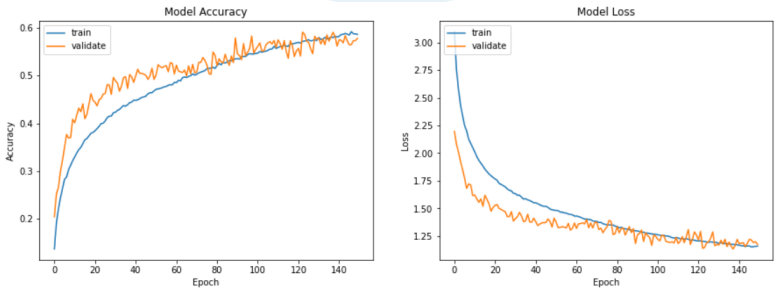

Figure 9 – Cate_Crossentropy,SGD,lr=10e-4

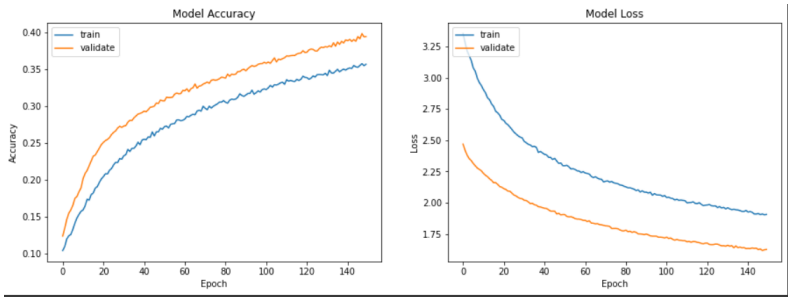

Figure 10 – Cate_Crossentropy, SGD,lr=10e-5

Using SGD optimizer and learning rate 10e-3, about accuracy, we obtain 0.8 in train and 0.8 in validate, so we have little difference, but many peaks in the validate curve. About loss, we get 0.54 in train and 0.54 in validate, we have little difference, but many peaks in the validate curve. Using learning rate 10e-4, about accuracy, we obtain 0.58 in train and 0.57 in validate, so we have 0.01 of difference, and many peaks in validate curve. About loss, we get 1.21 in train and 1.22 in validate, so we have 0.01 of difference. Finally, using learning rate 10e-5, about accuracy, we obtain 0.33 in train and 0.39 in validate, so we have 0.06 difference. About loss, we get 2.05 in train and 1.73 in validate, it's 0.38 of difference. The best results, was with learning rate 10e-3 where best results in train and validate and good in overfitting.

3.3.2 Results

By observing the curves of model accuracy and loss, also the test loss and test accuracy, the combination: optimizer:[Adam], loss-function:[crossentropy], learning rate:[10e-4] performs best. The following table shows the results obtained by each configuration:

Table 1 – Evaluation CNN results

Type	Test loss	Test accuracy
Adam,10e-3	0.437	0.874
Adam,10e-4	0.449	0.864
Adam,10e-5	0.475	0.838
RMSprop,10e-3	0.411	0.878
RMSprop,10e-4	0.469	0.854
RMSprop,10e-5	0.602	0.792
SGD,10e-3	0.544	0.810
SGD,10e-4	1.175	0.577
SGD,10e-5	1.626	0.394

The confusion matrix of this combination shows except for type "cat" and "dog", the rest types can be classified with a high accuracy. This is the reason why we apply other techniques

3.4 Transfer Learning Models

3.4.1 VGG 16

Taking as base model VGG16, it is necessary up sample the input to get the 224x224x3 as VGG16 needs, as is shown in the figure 20 apendix A, was implemented a global average pooling to get a good

representation and pass then to a flatten and use batch normalization to avoid overfitting, finishing with a dense layer to do the 10 classes classification. Our final model have 14,721,866 parameters in total.

Using VGG16 and learning rate 10e-4, about accuracy, we obtain 0.98 in train and 0.88 in validate, so we 0.1 difference between curves. About loss, we get 0.05 in train and 0.45 in validate, we have 0.4 of difference, so there is overfitting in this model. We will try another method to avoid overfitting.

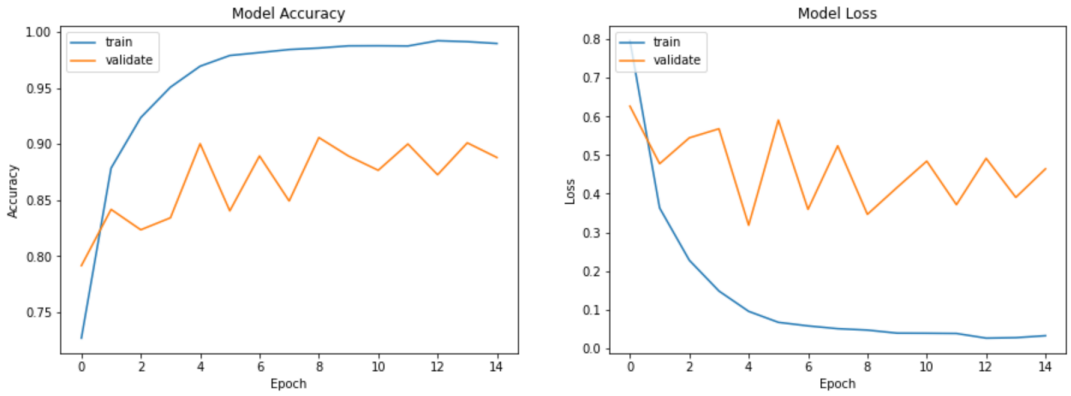

Figure 11 – Loss and accuracy performance in VGG16 - By authors

It is painfully slow to train, the reason is because VGG16 is over 533MB. The network architecture weights themselves are quite large (concerning disk/bandwidth).

3.4.2 ResNet50

In this model we use transfer learning technique with ResNet50 architecture using the pretrained weights with the ImageNet image dataset.

- **Model 1:** Before the base model is done, as is shown in the figure 21 on apendix A, we add 3 layers of upsampling (2,2) to fit in the input size ResNet needs, then we add the base model and apply regularization layers with batch normalization and finally a softmax classifier.

  Since the method is computationally expensive, it trained it for 10 epochs which give us a high training accuracy. After of the training part, we select the type of optimizer and loss-function and learning rate components for model training. For optimizer:[Adam], loss-function:[crossentropy], learning rate:[10e-3], see Figure 12.

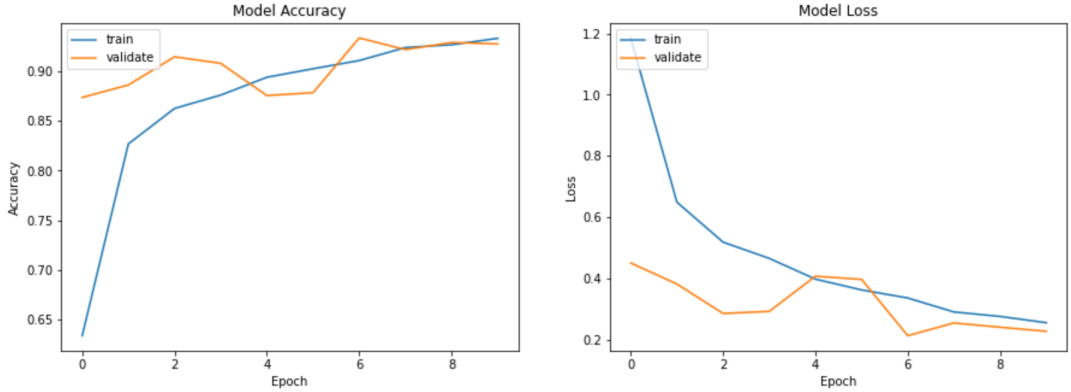

Figure 12 – Loss and accuracy performance in ResNet50 Model 1 - By Authors

- **Model 2:** For the second configuration, it was decided to modify the ResNet50. The main purpose

is to observe what is the difference if we use less layers with respect to the first model, for that as is showed in the pictures 13 and 22 on apendix A,it was added a layer with GlobalAveragePooling2D so that after the convolutional operations that it came with previously, it makes the average of all the values according to the last axis. Following this, we append a layer with Flatten to take a tensor of any shape and transform it into a one-dimensional tensor. Following this, 3 Dense were added, one of them of 1024, another of 512, and finally one of 10 to obtain the classes that we have as an objective, the latter accompanied by a softmax to normalize the output in probabilities.

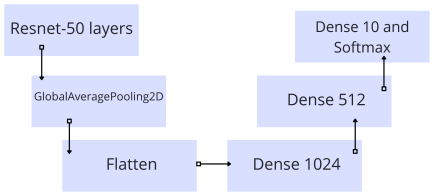

Figure 13 – ResNet50 Architecture Design - Model 2

As this configuration was faster and obtain high accuracy than the model 1, it was enough to use 5 epoch to overcome the results of the first model,

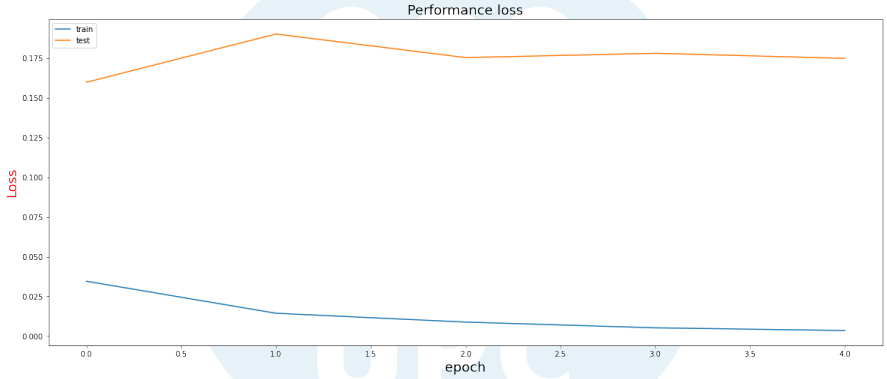

Figure 14 – Loss performance in ResNet50 Model 2 - By Authors

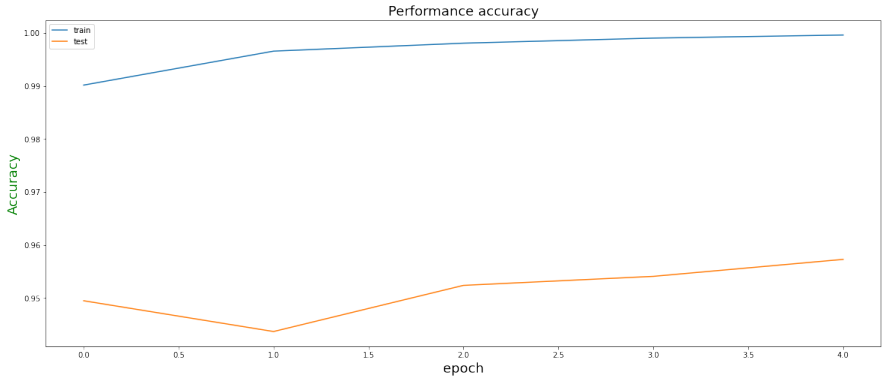

Figure 15 – Accuracy performance in ResNet50 Model 2 - By Authors

Results

As we can observe in table 2, for the first model the loss obtained was 0.2265 (see figure 12), while for the second it was 0.1749, which means decrease 0.0516, in other words 22,78% . On the other hand, in terms of accuracy, model 1 obtained 0.9279, while as is observed in the figure 15, the second obtained 0.9573, which shows an improvement of 3.07%, that is why, it could be concluded

that model 2 presented a little improvement accuracy than model 1. Finally, regarding the test time, model 2 was executed in 26 seconds, while model 2 took 77 seconds, in other words, the test time decrease by 66,2%.

Table 2 – Evaluation ResNet50 Model 1 and Model 2

	Test loss	**Test accuracy**	**Time(Seconds)**
Model 1	0.2265	0.9279	77
Model 2	0.1749	0.9573	26

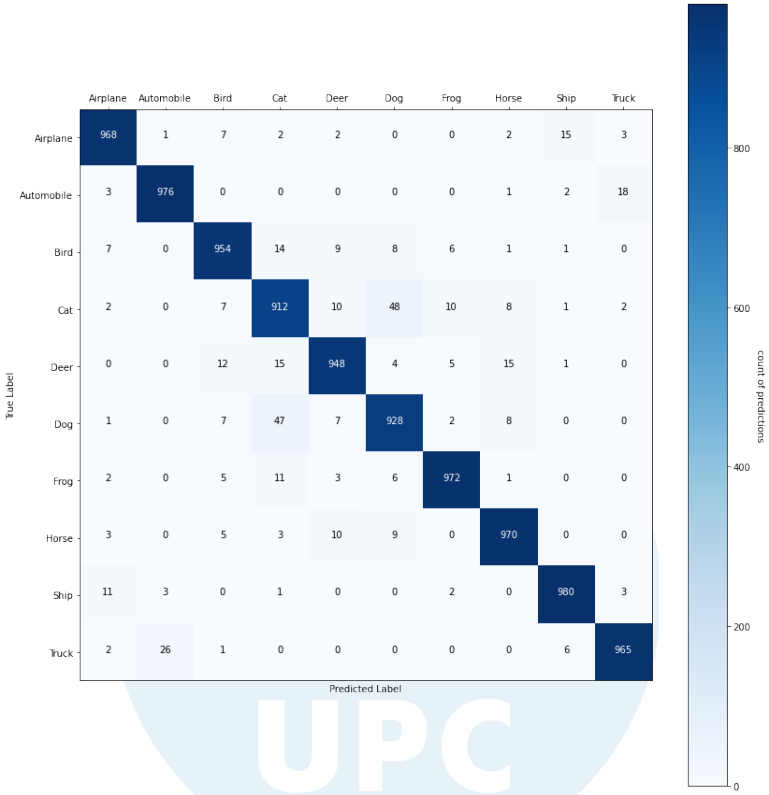

Figure 16 – Confusion matrix, ResNet50 model 2 - by Authors

4 Discussion

- It was possible to verify that the complexity of developing a CNN from scratch can be quite high, as well as its computation time, if we compare it with techniques such as transfer learning. Also, based on the results obtained and as shown in table x, transfer learning obtained better results, for example, using ResNet50 we reduced the loss by 59.3% and increased the accuracy by 11% with respect to our CNN.

Table 3 – Model Result - By Authors

	Validation loss	**Validation accuracy**
CNN from scratch	0.4298	0.8717
VGG16	0.4645	0.8880
ResNet50 - Model 2	0.1749	0.9573

- As was observed in the table 2 and figures 12 14 15, transfer learning improves in a noticeable way

the accuracy of the model, because lets us construct a model in less time and grants the wisdom of models trained previously in big datasets. On another side, based on the results it can be possible to say that is better to use with fewer layers to adapt for our objective, with this the train and test time is lower because the model is more simple.

- **Kaggle Score from CNN and ResNet50 - Model 2** In Kaggle CIFAR-10, the test data is in a 300,000 size: 10,000 images which are the same with official test set, and extra 290,000 noisy images. The best transfer learning model and CNN were chosen to upload the CSV to Kaggle and obtain the following score.

Figure 17 – Kaggle Score for CNN - by Authors

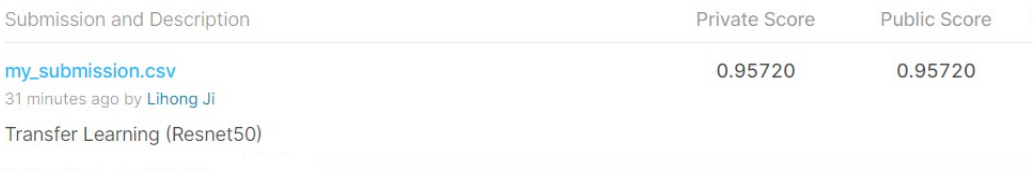

Figure 18 – Kaggle Score for ResNet50 model 2 - by Authors

5 Conclusions

- After developing and evaluating four different CNN architectures for image classification, the models based on transfer learning proved to be a good alternative to creating CNN model from scratch, being the model that showed the best performance the *ResNet50 - Model 2*, with which an accuracy of 0.9573 and a loss of 0.1749 were achieved.
- Based on the knowledge from CI lectures, the architecture of a CNN consists of three main components: convolution stage, pooling stage and dense net stage. Following that structure, we created our own CNN model from scratch and, as a reference, it took us about one week to complete the research and design its architecture (shown in appendix A). In comparison, we were able to conduct the research and the generation of the other models, which are based in transfer learning, in a matter of a few days. In conclusion, because the hyperparameter tuning from scratch requires high level of knowledge and experience to obtain a good performance, it can be very useful to leverage transfer learning with pre-trained architectures as a simpler and faster way to classify images.
- When it comes to model training, with epoch=100 and batch_size=32, each model consumed from about 30 to 40 minutes without data augmentation, and after applying data argumentation the training time consumption increased to about 60 minutes but the accuracy only increased 2%.
- During the hyperparameter setting, initially the number of filters in the convolution stage was set to 32, 64, 128. Activation function is set as 'ReLu'. After these two steps, it is followed by Batch_Normalization, default configuration Maxpooling2D and Dropout, so that we decrease model complexity and probability of overfitting. In the next part of the architecture, we set a 128 dense net. Here when the number of perceptrons exceeds 128, the final accuracy would drop, and in the previous test, 256 dense net gets about 80% accuracy. For optimizer and loss, after comparing the results from different parameter combinations, we draw a conclusion that [optimizer: Adam, loss:cross-entropy, learning rating:10e-4] performs best.

- From experimenting with the ResNet models, we learned that adding too many layers can be detrimental to the performance. Therefore, we concluded that more complexity in the architecture does not necessarily improve the model.
- As a strength of the present work, we would like to emphasize the variety of solutions to the problem that we provided, since in addition to creating our own CNN architecture, we wanted to further explore other alternatives, which we did by implementing transfer learning as well.
- As a weakness, the processing task in deep learning takes huge computational and time-consuming resources to develop CNN models and that is why it is useful to achieve rapid and positive progress in our models, it is also possible to add more layers and adapt to our tasks obtaining improve its performance and thus spend less time and resources.

References

[1] Garnham, Alan. "Introduction". *Artificial intelligence: an introduction.* (Routledge Kegan Paul. 1988): 01–24, 24–44.

[2] Garnham, Alan."Vision, Thinking and reasoning, Language, Learning, ". *Artificial intelligence: an introduction.* (Routledge Kegan Paul. 1988): 57–270.

[3] O'Shea, Keiron Nash, Ryan. (2015). An Introduction to Convolutional Neural Networks - Scientific Figure on ResearchGate. Available from:https://www.researchgate.net/figure/An-simple-CNN-architecture-comprised-of-just-five-layers_fig4_285164623

[4] Haykin, Simon. "Introduction". *Neural networks and learning machines.*(Prentice Hall/Pearson): 11–17.(2009)

[5] K. Simonyan A. Zisserman, "Very Deep Convolutional Networks for Large-scale image recognition", Visual Geometry Group, Department of Engineering Science, University of Oxford, 2015

[6] https://www.image-net.org/

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition," arXiv:1512.03385 [cs], Dec. 2015, Accessed: Nov. 22, 2020. [Online].
Available: http://arxiv.org/abs/1512.03385.

[8] S. Ruder, "An overview of gradient descent optimization algorithms", Insight Centre for Data Analytics, NUI Galway, 2017

[9] Željko Đ., "Classification Model Evaluation Metrics", International Journal of Advanced Computer Science and Applications, Vol. 12, No. 6, 2021

[10] Z. Zhang, M. Sabuncu, "Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels", Meinig School of Biomedical Engineering- Cornell University, NeurIPS 2018 [Online].

[11] https://www.cs.toronto.edu/ kriz/cifar.html

[12] C. Shorten and T. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning", Journal of Big Data, 2019 [Online].

Appendix A. Implementation details

- **Summary of the CNN model**

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 32, 32, 32)        896

 activation (Activation)     (None, 32, 32, 32)        0

 batch_normalization (BatchN  (None, 32, 32, 32)       128
 ormalization)

 conv2d_1 (Conv2D)           (None, 32, 32, 32)        9248

 activation_1 (Activation)   (None, 32, 32, 32)        0

 batch_normalization_1 (Batc  (None, 32, 32, 32)       128
 hNormalization)

 max_pooling2d (MaxPooling2D  (None, 16, 16, 32)       0
 )

 dropout (Dropout)           (None, 16, 16, 32)        0

 conv2d_2 (Conv2D)           (None, 16, 16, 64)        18496

 activation_2 (Activation)   (None, 16, 16, 64)        0

 batch_normalization_2 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 conv2d_3 (Conv2D)           (None, 16, 16, 64)        36928

 activation_3 (Activation)   (None, 16, 16, 64)        0

 batch_normalization_3 (Batc  (None, 16, 16, 64)       256
 hNormalization)

 max_pooling2d_1 (MaxPooling  (None, 8, 8, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 8, 8, 64)          0

 conv2d_4 (Conv2D)           (None, 8, 8, 128)         73856

 activation_4 (Activation)   (None, 8, 8, 128)         0

 batch_normalization_4 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 conv2d_5 (Conv2D)           (None, 8, 8, 128)         147584

 activation_5 (Activation)   (None, 8, 8, 128)         0

 batch_normalization_5 (Batc  (None, 8, 8, 128)        512
 hNormalization)

 max_pooling2d_2 (MaxPooling  (None, 4, 4, 128)        0
 2D)

 dropout_2 (Dropout)         (None, 4, 4, 128)         0

 flatten (Flatten)           (None, 2048)              0

 dense (Dense)               (None, 128)               262272

 activation_6 (Activation)   (None, 128)               0

 batch_normalization_6 (Batc  (None, 128)              512
 hNormalization)

 dropout_3 (Dropout)         (None, 128)               0

 dense_1 (Dense)             (None, 10)                1290

 activation_7 (Activation)   (None, 10)                0

=================================================================
Total params: 552,874
Trainable params: 551,722
Non-trainable params: 1,152
_________________________________________________________________
```

Figure 19 – Summary of CNN architecture - By authors

- **Summary of the VGG model**

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_3 (InputLayer)        [(None, 32, 32, 3)]       0

 up_sampling2d_1 (UpSampling  (None, 224, 224, 3)      0
 2D)

 vgg16 (Functional)          (None, 7, 7, 512)         14714688

 global_average_pooling2d_1   (None, 512)              0
 (GlobalAveragePooling2D)

 flatten_1 (Flatten)         (None, 512)               0

 batch_normalization (BatchN  (None, 512)              2048
 ormalization)

 classification (Dense)      (None, 10)                5130

=================================================================
Total params: 14,721,866
Trainable params: 14,720,842
Non-trainable params: 1,024
_________________________________________________________________
```

Figure 20 – Summary of VGG16 architecture with transfer learning - By authors

- **Summary of ResNet50 - Model 1**

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 up_sampling2d_10 (UpSamplin  (None, 64, 64, 3)        0
 g2D)

 up_sampling2d_11 (UpSamplin  (None, 128, 128, 3)      0
 g2D)

 up_sampling2d_12 (UpSamplin  (None, 256, 256, 3)      0
 g2D)

 resnet50 (Functional)       (None, 8, 8, 2048)        23587712

 flatten_3 (Flatten)         (None, 131072)            0

 batch_normalization_3 (Batc  (None, 131072)           524288
 hNormalization)

 dense_3 (Dense)             (None, 10)                1310730

 activation_3 (Activation)   (None, 10)                0

=================================================================
Total params: 25,422,730
Trainable params: 25,107,466
Non-trainable params: 315,264
_________________________________________________________________
```

Figure 21 – Summary ResNet50 Architecture with transfer learning model 2 - By authors

- **Summary of ResNet50 - Model 2**

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 32, 32, 3)]       0

 up_sampling2d (UpSampling2D  (None, 224, 224, 3)      0
 )

 resnet50 (Functional)       (None, 7, 7, 2048)        23587712

 global_average_pooling2d (G  (None, 2048)             0
 lobalAveragePooling2D)

 flatten_1 (Flatten)         (None, 2048)              0

 dense_2 (Dense)             (None, 1024)              2098176

 dense_3 (Dense)             (None, 512)               524800

 classification (Dense)      (None, 10)                5130

=================================================================
Total params: 26,215,818
Trainable params: 26,162,698
Non-trainable params: 53,120
_________________________________________________________________
```

Figure 22 – Summary ResNet50 architecture with transfer learning model 2 - By authors