CAPSTONE

0 – Introduction

Convolutional Neural Networks (CNN) are widely used in the field of AI and machine learning. Specifically they get fully into the field of deep learning.

Within this area it is common to solve problems related to computer vision; that is, the treatment of images in some way. That is why among the projects offered by Udacity to complete this "Machine Learning Engineer" nanodegree, there is one related to the aforementioned CNNs.

The scope of this project is clearly defined since it is one of the pre-established options that Udacity provides. In this sense, the project consists, broadly speaking, in developing a neural network model that, from an image given by the user, is capable of identifying the dog breed that appears in the image.

It also needs to be able to identify if a human is detected in the image, and then it will provide an estimate of the dog breed that is most resembling.

Obviously there are as many solutions as there are programmers. The following pages describe in detail the process followed to arrive at a possible solution implementation. It is not the best; but its operation is more than acceptable and amply meets the objectives of the project.

1 – Starting data

As starting data there is a graphical database of people's faces and a graphical database of dogs.

1.1-Human faces data set.

The human faces data set can be downloaded from the following link:

https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/lfw.zip

The data set of human faces consists of 13,233 images in which human faces appear. Each image file is named in a way that indicates the name of the person in the photo. Also, since in many cases there are several photos of the same person, the data set is structured in folders for each person. For the purposes of this work, who is who is not relevant, since it is not intended to identify a specific person. The only fact that is relevant is that in all these images the face of at least one person appears.

1.2-Dogs data set.

The dog images data set can be downloaded from the following link:

https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip

The dog image data set consists of a total of 8,351 images of different dog breeds. These images are labeled indicating the dog breed. There are a total of 174 different breeds labeled in 118 groups (some groups, due to the great similarity, encompass more than one breed)
The dog data set is already provided stratified and separated for training (80%), validation (10%) and test (10%), so the main folder contains three folders (one for trainining, one for validation and one for test).

Each of these folders consists of exactly 133 elements; that is, one for each dog breed. These elements are nothing more than folders containing within them a variable number of image files of the dog breed in question.

In the validation and test folders each dog breed has between 3 and 10 photos. Since most of the data is used for training, in these folders there can be more than 70 photos for each breed of dog.

The name of each file explicitly identifies the breed of dog seen in the image, followed by a serial number.

Note that the data structure of the dog images is already prepared to do this job. Otherwise it would be necessary to do this pre-processing of the data.

2 – Detection of contents.

2.1-Human face detection.

First of all, to implement the desired functionality it is necessary to be able to identify if a human's face appears in a photo.

For this, although there are other options, it's has been used the python library called OpenCV. This library has a face detector. This way it is easy to implement a function that identifies whether or not there is a face in a given image.

This face detector has been tested with the first 100 images of human faces, achieving a hit rate of 98%. On the other hand, curiously it has been tested with the first 100 images of dogs and has found human faces in 17% of these.

2.2-Dog detection

The second detection task is to identify dogs in the images. For this we will use a pre-trained model called VGG16.

The VGG-16 it's a model that have been trained on ImageNet, a very large, very popular data set used for image classification and other vision tasks. ImageNet contains over 10 million URLs, each linking to an image containing an object from one of 1000 categories.

This model, as well as others, is contained in the torchvision library and can be easily downloaded.

The way to identify if there is a dog in the image is done as follows. You look at the Imagenet dictionary and it is verified that of the 1000 elements that you can identify there are 118 that correspond to dog breeds. In this way, it will be enough to check that the identified label corresponds to one of those 118 values. The labels of interest are those that go from position 151 to 268 (both included). It can be checked at https://gist.github.com/yrevar/942d3a0ac09ec9e5eb3a

In summary, for the detection of dogs a pre-trained model is used that identifies up to 1000 different elements. If the output of this model is one of those corresponding to a dog, then we consider that there is a dog in the image.

What has just been described has been done with three different models (VGG-16, Inception-V3 and ResNet-50). The first 100 pictures of dogs have been checked. The results are shown below:

	Percentage of dogs detected in...	
	...human files (%)	***...dog files (%)***
VGG-16	0	100
Inception-V3	0	99
ResNet-50	0	100

3 - Dog breeds classification with CNN from scratch.

Now, having a way to detect human and dog faces in the images, it is time to design and deploy a neural network from scratch.

First of all, it must be said that as a first approach to the problem I have taken into account two papers:

1. *Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)*
2. *Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in Neural Information Processing Systems, pp. 1097–1105 (2012)*

Nevertheless, my design will always be a freely and very simplified version of the original models stated in those papers.

3.1-Transformations.

After defining the dataloaders, it's time for the transformations. Even if you have a huge database, it is advisable to do some augmentation, since this will affect the randomness of the data and the independence of the scale, rotation and translation of the image.

Another point to consider is the dimensions of each image. In this case I have selected an image dimension of 224x224. See that these are the same dimensions that are used in reference 1.

In my case, since the database is not huge, I did make some transformations. Specifically, first I make a random rotation of a few degrees (10º), then I crop to the desired dimension (224x224) and finally I horizontally flip a certain percentage of the samples, as it was made in the paper referenced as 1..

Finally, in all cases the values between 0 and 1 are normalized.

The above is valid for the training data set. I do it this way because it is in training where I want to obtain the maximum variability of the data in order for the network to extract the general features of the images. For validation and testing the transformations are not the same. The only thing I do is scale to 256x256 and then crop the central part to the desired dimensions (224x224).

3.2-Architecture.

I started by defining quite ambitious networks. Specifically, I started with a network that had 5 convolutional layers and 3 fully connected layers. The system quickly ran out of memory and I was unable to run the workout.

I had to try making some modifications to the value of the batch and reduce the network it self. After several attempts I have finished with a network that has 3 convolutional layers and 2 fully connected layers. The following graph shows what exactly is the network model with all its layers, and indicating the dimensions of one single image.

Layer	*Input size*	*Output size*
Convolutional 1	224 x 224 x 3	224 x 224 x 32
MaxPool 1	224 x 224 x 32	112 x 112 x 32
Convolutional 2	112 x 112 x 32	112 x 112 x 64
MaxPool 2	112 x 112 x 64	56 x 56 x 64
Convolutional 3	56 x 56 x 64	56 x 56 x 128
MaxPool 3	56 x 56 x 128	28 x 28 x 128
Fully connected 1	28 x 28 x 128 = 100,352	1330
Fully connected 2	1330	133

All convolutional layers have a structure such that they multiply the depth by 2, except for the first one.
In the first layer, 32 3x3 filters are applied with stride=1 and padding=1. In the other layers the kernel size 3x3 is maintained with stride=1 and padding=1, which causes the depth to be multiplied exactly by 2. After each layer, a Relu-type activation is applied. After each activation a maxpool with a 2x2 kernel and stride=2 are also applied. This exactly divides the height and width of the image by 2.

In addition, prior to the fully connected layers, a dropout is applied to avoid overfitting. In the original paper maxpool is not applied after each convolutional layer; but in that case they have many more layers, so in my case I have considered it appropriate to apply them. Relu is not applied to the last layer because by default it already comes with softmax, which is more convenient for the final output from the system.

3.3-Algorithm.

The training algorithm is, in principle, quite standard. Naturally, the basic steps are respected (zero grad, forward pass, compute loss, back propagation, optimization step).

On the other hand, different criteria have been experimented in the case of the learning rate and early stop.

In principle the best fit criterion is taken based on the minimum validation error. In this way, even if the model continues training, and as much as it falls into overfitting, if the result does not improve in the validation, that model will not be saved.

This in itself is already enough to avoid overfitting to a great extent.

Even so, trying to optimize training time, it was decided to incorporate an early stop. At first I tried an early stop consisting of stopping training after 4 periods without improvement in validation. This system has clearly turned out to be worse than allowing a large number of epochs to train without anything else but merely keeping the model that best fits the validation data.

Probably, the problem with this approach has been the choice of the early stop criterion. Later, I was able to verify that with a less restrictive criterion, similar results were obtained to those obtained by running a large number of epochs (always maintaining the criterion of taking the best fit in the validation).

I have also experimented with the learning rate. In the first attempts the system converged very slowly. I have made a manual adjustment after several attempts. I have also tried with a variable learning rate, with results similar to the best manual adjustment.

3.4-Results.

The from scratch model described above offers more than acceptable results. In fact they are noticeably better than expected and requested in the initial problem description.

Specifically, it has been tested with 6680 images matching 3200. This means that the accuracy has been 47%, when the expected minimum was 10%.

It is evident that the network works. Of course it is clearly improvable and we will see this in more detail in the next step, when using transfer learning; but getting it right in almost half the cases is certainly a great first approximation.

4 - Dog breeds classification with pre-trained CNN (transfer learning).

Training an extensive model like this, with the scarce resources that a student can count on is not the best way to approach the solution to the problem.

From a pedagogical point of view it is amazing to design a neural network from scratch; but from a practical point of view the best option is to use a pre-trained network that can be easily adapted to the problem to be solved.

As already anticipated with the work of dog detection, VGG-16 is a pre-trained model on an extensive image database, which includes up to 118 tags intended for dog breeds.

A good approach to solving the problem is to take this network, which is already trained with thousands of images, and replace the classifier (the latest fully-connected layers) to adapt it to the requirements of the current problem.

In this case this can be done without more than modifying the last output layer so that it has 133 outputs (as many as different dog breeds we are showing the model)

In this way, when training, the weights of the convolutional layers will not be modified at all, keeping the deep learning stored in them intact. The only thing to change is the classifier weights (the last fully connected layers).

This approach, as we will see in the analysis of the results, offers a noticeably better and faster adjustment to the design from scratch.

4.1-Transformations.

In order to be able to make a comparison of both systems, I have kept the same dataloaders and the same transformation, augmentation and normalized criteria.

The criteria previously established on the dimensions of the images have also been maintained.

4.2-Architecture.

In this case it is convenient to speak of a mixed architecture. The first part is the one described for the VGG-16 model that has 13 convolutional layers. Between them there are activation and pool layers, but we will not emphasize them.

The second part of the model, the classifier, instead of taking the VGG-16 model, consists of two fully connected layers that progressively adapt the output dimensions to the 133 dog breeds with which we feed the model.

Layer	*Input size*	*Output size*
13 Convolutional layers	224 x 224 x 3	7 x 7 x 512
Fully connected layer 1	7 x 7 x 512 = 25,088	1,330
Fully connected layer 2	1,330	133

In order to adapt the convolutional layers of the VGG-16 model to the particular classifier designed, what has had to be done is to explore the details of the dictionary type variable resulting from importing the model.

This variable has two parts: features and classifier. Below is the classifier as it appears at runtime.

```
(classifier): Sequential(
  (0): Linear(in_features=25088, out_features=4096, bias=True)
  (1): ReLU(inplace)
  (2): Dropout(p=0.5)
  (3): Linear(in_features=4096, out_features=4096, bias=True)
  (4): ReLU(inplace)
  (5): Dropout(p=0.5)
  (6): Linear(in_features=4096, out_features=1000, bias=True)
```

See that to adapt my classifier to the network, it would be enough to change the out_features of the last layer to 133.

Even so, in order to minimize size, it has been verified that with a smaller network the final results were not significantly altered. For this reason, a two-layer network has been chosen, progressively adapted to the output features.

4.3-Algorithm.

The training algorithm is, the same used in the network created from scratch.

Again, different criteria have been experimented in the case of the learning rate and early stop, but the conclusions do not differ from those explained in point 3.3 for the neural network from scratch.

The best fit criterion is taken based on the minimum validation error.

4.4-Results.

As expected, the results are much better with this pre-trained network than with the from scratch network. This time an accuracy of 82% has been achieved. This practically doubles the percentage achieved with the network from scratch.

The minimum goal required was to achieve at least 60% of accuracy,

The output is even better than I expected. Even so I think that if I spend a lot of time with it, I could get even a better result (maybe not too much better). Some of the points that I could use to improve the fitting could be as follow:

- I could try with any parameter variation (such as different learning rate updatting, number of epochs, initial values...)
- I could also try with a different loss function. See that I'm using CrossEntropyLoss, but there are other options.
- I could use a different optimizer. I use SGD; maybe I could have tried with Adam.
- I could change completely my classifier structure. I could add a third layer (or more) and try with different number of output features in between.
- I could make more transformations to the initial set of images, or even get more images from other databases.

- See that I've used VGG16 as a pretrained model. I could check how does it work with InceptionV3, Resnet 50 or GoogleNet.
- I also could improve the face detector and dog detector algorithms. This could be done by using again CNN with transfer learning from a good model.
- There is always the possibility of developing a better, more complex model that requires long hours of training. Unfortunately my resources do not go as far; but certainly that possibility exists. You could even start from the architecture of a VGG-16 and train it from scratch exclusively with images of dogs.

There is always room for improvement and artificial intelligence is currently booming, experiencing a time of great expansion and research.

5 – Conclusion

The first comparison between the two models is immediate. It can be done simply by comparing the accuracies of each of them.

The from scratch model achieves an accuracy of 47% while the model made through transfer learning reaches 82%.

Clearly, although both models far exceed the success by simple guessing, the second is much better, offering very good results.

Another way to compare both models is by having a look to the validation loss. It's just to get an idea of the order of magnitude at first glance. To avoid a biased situation with initial values we compare loss value in 10th epoch. In the first case (from scratch) we have a value of 4,121162, while the value in the pre-trained model is 0.84020.

In fact, the first value in the pre-trained model (1,60229) is much better than the best value in the scratch model (3,231425).

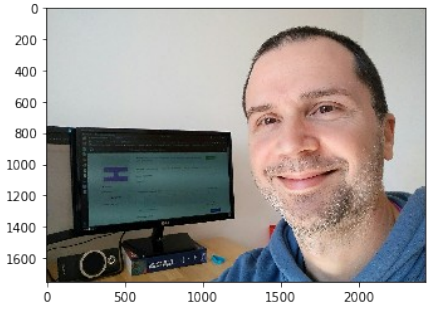
In the case of the from scratch model, increasing the complexity of the network is no guarantee of success. In fact, I have tested a network with 4 convolutional layers and 3 fully connected layers (instead of 3 cv and 2 fc of the chosen model) with approximately same results than those shown here with fewer layers.

In conclusion, it is a field in which you can potentially do a lot of research, and the range of possibilities is such that it is difficult to cover them all.

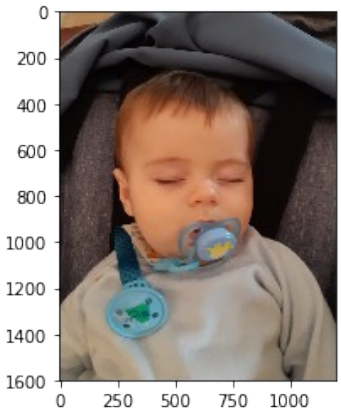
To finish we can see graphically some of the results.

```
Error!!! No dog or human has been detected in this image.
```

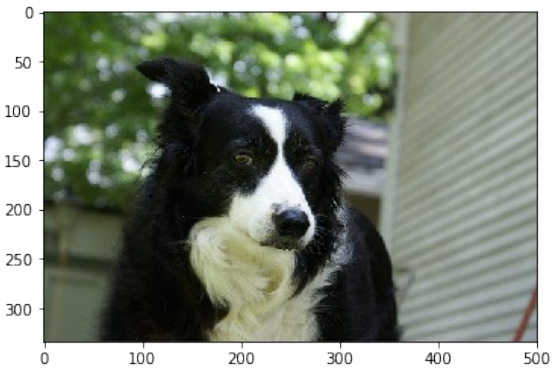

```
Hi human. You look like a Pharaoh hound
```



Hi human. You look like a Italian greyhound



Border collie



Dalmatian