DOG BREED IDENTIFICATION WITH CNN

1. Definition

1.1.Project Overview

From the moment we open our eyes, we gradually learn how to classify all the objects around our living space. For some people, this ability is unique to humans and most of the living beings. ¿What if I told you that Machine Learning allows machines to do the same classification with an accuracy even superior to human eyes? The arise of modern CNN (Convolutional neural network) models in the recent years had allows us to apply this technology in a variety of fields. In this project, I will address the task of dog breed identification. My motivation is to demonstrate how powerful is CNN for a multi-class classification problem.

1.2. Problem Statement

The main goal of this project is to deploy a model than can classify a dog breed from an image. The first step is to distinguish whether the image contains a human face, a dog, or neither. After that, the next step is to detect the dog breed with at least 60% of accuracy. Our final solution must distinguish a dog breed when a dog is present in the image or suggest the most similar dog breed when the image contains a human face. The final part is done just for fun.

This project follows these steps:

Step 0: Import Datasets:
Step 1: Detect Humans:
Step 2: Detect Dogs
Step 3: Create a CNN to Classify Dog Breeds (from Scratch)
Step 4: Create a CNN to Classify Dog Breeds (using Transfer Learning)
Step 5: Write Your Algorithm
Step 6: Test Your Algorithm

1.3. Metrics

Two metrics are considered in this project. The Log loss values in the train and validation data and the accuracy, which is defined as the ratio of correct predictions to the total size of the test data. Specifically, for the two CNN models deployed:

- Test accuracy greater than 10% for the model from scratch.
- Test accuracy greater than 60% for the transfer model.

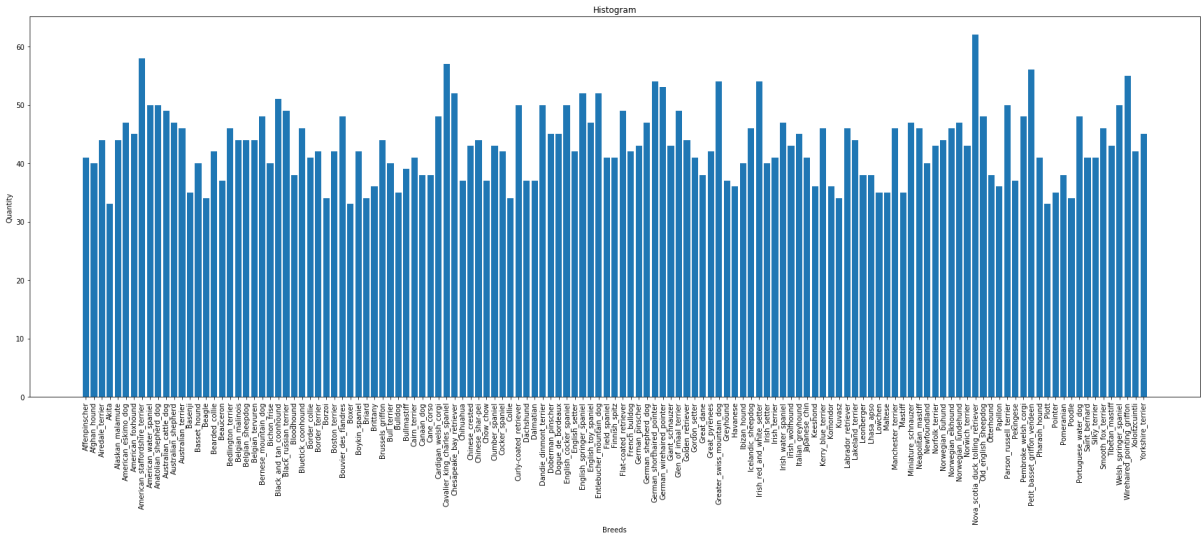
2. Analysis

2.1.Data Exploration

The dataset for this project is provided by Udacity machine learning nanodegree program. The information is shared in a repository that contains two sub-datasets.

- Dog images dataset: This dataset contains a total of 8351 dog pictures divided in the following subfolders: train, valid and test that are also divided in 133 subfolders which represent the dog breeds.
- Human images dataset: This dataset contains 13233 of total human pictures divided in 5750 subfolders. All the images are of size 250x250.

The distribution of the dog dataset is calculated counting the number of files in each sub-folder specifically inside the train subfolder.



As you can see, the dataset is not perfectly balanced (where the number of examples in each class are between 33 and 62). However, it is considered that this dataset is (almost) a uniform distribution. On the other hand, when the labels of the original dataset are unbalanced, which is usually the case in real-world classification problems, naively shuffling and splitting the dataset could lead to a major problem: infrequent classes end up in one partition and not in the other. In such a case, it is recommended to use advanced sampling techniques, such as stratified sampling.

2.2.Exploratory Visualization

It is always a good practice to visualize some random examples of the dataset before the training process. In this case, the following images have been transformed and are part of the train iterator. Therefore, the images are showed with its torch size (10,3,224,224) and batch size (10).

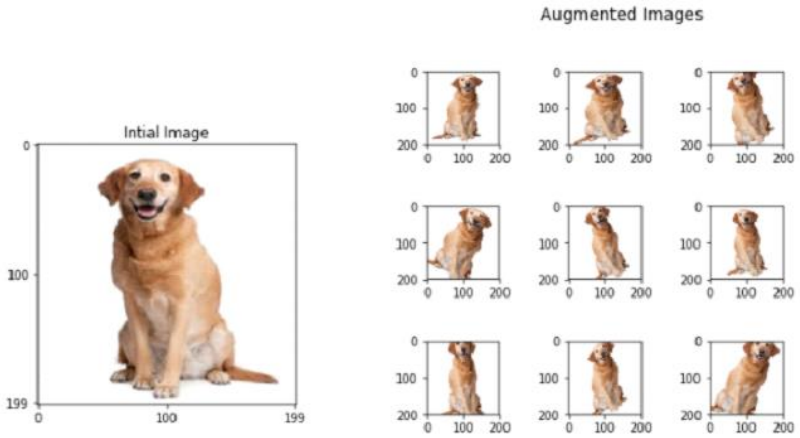
```
Inputs size: torch.Size([10, 3, 224, 224])
Classes size: torch.Size([10])
```

We can observe that in effect there are different dog breeds in the dataset. Also, the angles from which the pictures were taken are different. The same applies to the background colors which are diverse. Furthermore, it can be observed that most objects are centered, making the classification task easier.

2.3.Algorithms and Techniques

Data augmentation:

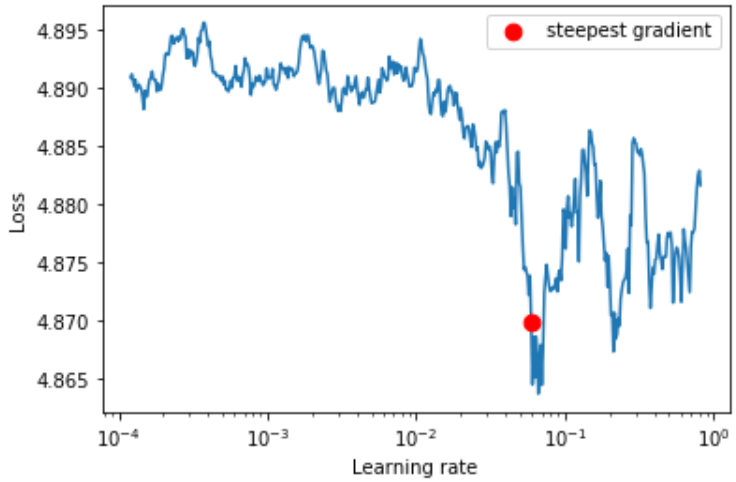
Data augmentation is a trick which can be used to augment the effective size of your training data. This can, in turn, help reduce overfitting. It consists in creating new data by transforming available training data. For example, for images, we can resize, stretch, rotate, mirror them to obtain additional images. Here are some examples of image augmentations (right) from an original image (left):



Learning Rate Finder

This is a useful technique to find the learning rate. There is an implementation for Pytorch available on GitHub with more information. The next image shows an example:


```
Stopping early, the loss has diverged.
Learning rate search finished. See the graph with {finder_name}.plot()
LR suggestion: steepest gradient
Suggested LR: 5.94E-02.
```



Transfer Learning

Sometimes is not useful to train an entire Convolutional Network from scratch because it is relatively rare to have a dataset of sufficient size. Instead, it is common to use a pre-trained CNN on a very large dataset such as: ImageNet which contains 1.2 million images with 1000 categories, and then use our CNN either as an initialization or a fixed feature extractor for the task of interest. This is referred to as transfer learning.

2.4.Benchmark

Model from Scratch

Due to the model from scratch requires at least 10% of accuracy, it is necessary to implement a model in pytorch framework designed for image classification. A good choice is a model with at least with 2-dimensional CNN (Conv2d).

Model Transfer

Nowadays there are a good number of models pretrained with ImageNet. To choose one of these models it is always possible to review some analysis about them in pytorch.org and other support sites. Alexnet model was chosen for the proposal. However, for this report, ResNet-18 will be implemented as a form of comparing both models.

Learning Rate

Choosing a learning rate is very important step. The idea is to find de maximum learning rate for which model will keep on converging faster. The Learning Rate finder has two approaches: Tweaked version from fastai and Leslie Smith's.

3. Methodology

3.1.Data Preprocessing

Due to the images in dog dataset does not have the same size, it is necessary to apply some transformations. Pytorch have the module torchvision.transforms for this purpose. Resize to 256x256 is applied and then CenterCrop to 224x224. These transformations are applied to all the dataset (train, validation, and test).

We take advantage of this transformation to add data augmentation to the dataset. RandomHorizontalFlip is applied and RandomPerspective(0.5,0.5,2,0) only to the train data. RandomRotation(5) was discard because it decreases the accuracy in some tests.

Finally, the sub-datasets (train, validation, and test) are converted to tensors and then to dataloaders using their specific transformations as showed in the next figure:

```
pretrained_means = [0.485, 0.456, 0.406]
pretrained_stds= [0.229, 0.224, 0.225]

train_transforms = transforms.Compose([
                    transforms.Resize(256),
                    transforms.RandomPerspective(0.5,0.5,2,0),
                    transforms.RandomHorizontalFlip(),
                    transforms.CenterCrop(224),
                    transforms.ToTensor(),
                    transforms.Normalize(mean = pretrained_means, std = pretrained_stds)
                ])
valid_transforms = transforms.Compose([
                    transforms.Resize(256),
                    transforms.CenterCrop(224),
                    transforms.ToTensor(),
                    transforms.Normalize(mean = pretrained_means, std = pretrained_stds)
                ])
test_transforms = transforms.Compose([
                    transforms.Resize(256),
                    transforms.CenterCrop(224),
                    transforms.ToTensor(),
                    transforms.Normalize(mean = pretrained_means, std = pretrained_stds)
                ])
```

3.2.Implementation

Model from Scratch

First, the learning rate (LR) is calculated based on a set of tests with transfer learning finder technique. It is very important to choose a random seed to fairly compare the two approaches mentioned early and some transforms-compose scenarios.

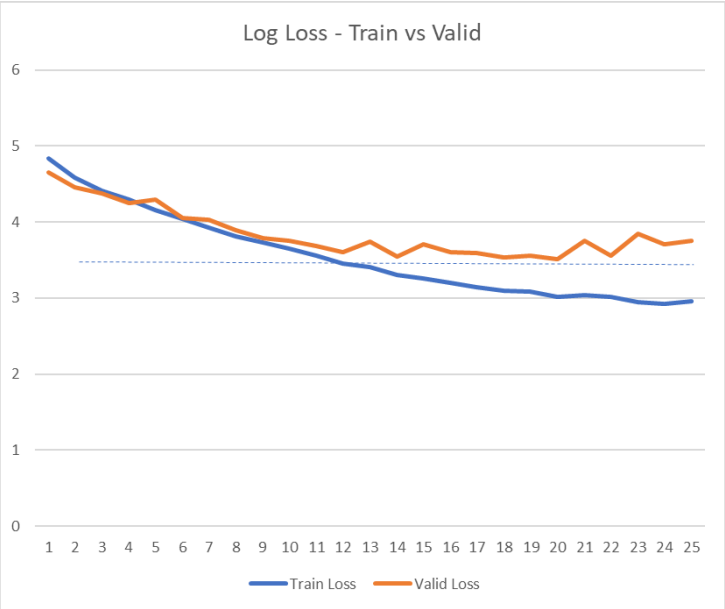
LR Finder approach	Random Seed	Batch Size	Transforms Compose	Learning Rate Suggested	# Iter	% Iter	Time proces	Loss
Leslie Smiths	1000	10	Horizontal(0),Perspective	2.64E-02	40	100%	7min	4.890
Tweaked Fastai	1000	10	Horizontal(0),Perspective	7.85E-04	400	94%	1min	3.500
Tweaked Fastai	1000	10	Horizontal(0),Rotation	1.56E-01	400	100%	1min	4.890
Leslie Smiths	1000	10	Horizontal(0)	5.35E-02	50	100%	8min	4.890
Tweaked Fastai	1000	10	Horizontal(0)	7.85E-02	400	100%	1min	4.881
Tweaked Fastai	1000	10	Horizontal(0),Perspective,Rotation	1.56E-01	400	100%	1min	4.896
Tweaked Fastai	**1000**	**10**	**Horizontal(0),Perspective**	**5.75E-02**	**350**	**100%**	**1min**	**4.875**

It is important to note in the previous table that the number of iterations can change the LR suggested for the LR finder.

The next step is to make some training on the model from scratch with 10 epochs and different Learning Rates. After that, the process can be repeated with more epochs, such us 25. The results are showed in the next table.

# Ephocs	Batch size	LR	Transforms Compose	Time	Loss in Data			Acc
					Train	Valid	Test	
10	10	0.02640	Horizontal(0),Perspective	17min 30s	3.766	3.804	3.797	12%
10	10	0.00079	Horizontal(0),Perspective	16min 27s	3.219	3.645	3.633	16%
10	10	0.15600	Horizontal(0),Rotation	15min 41s	4.211	4.273	4.234	5%
25	**10**	**0.05750**	**Horizontal(0),Perspective**	**37min 18s**	**2.959**	**3.751**	**3.566**	**17%**

One interesting finding was the result with 10 epochs and LR 0.00079 that achieves an accuracy of 16%. However, in order to demonstrate the divergence of performance between the training and the validation datasets (see next figure), for this report, 0.0575 was chosen as LR. Moreover, the model achieves **17% of accuracy** and 3.566 of log loss in the test data.



After the 10th epoch, we start to observe an increase in the divergence of performance between the training and the validation datasets. The loss on the training dataset keeps improving, while the one on the validation dataset is deteriorating. When this situation occurs, we are likely "overfitting" our model on the training dataset. Instead of learning relevant features, we see that the network only memorized the training examples.

Model Transfer

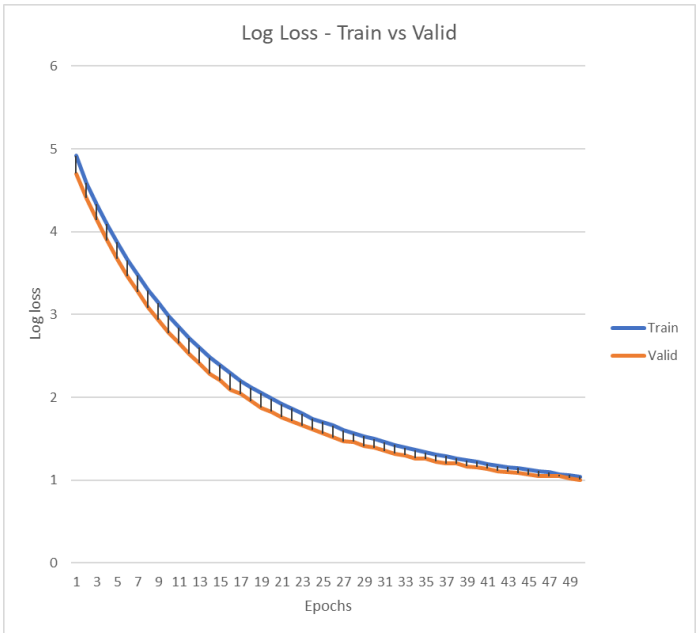
For this step, it is necessary to choose a pre-trained model. In this case, Resnet18 is implemented and then some tests are made changing batch size and epochs.

Resnet18, batch size=10, epochs=50:

- Train Loss: 0.834308
- Validation Loss: 0.703117

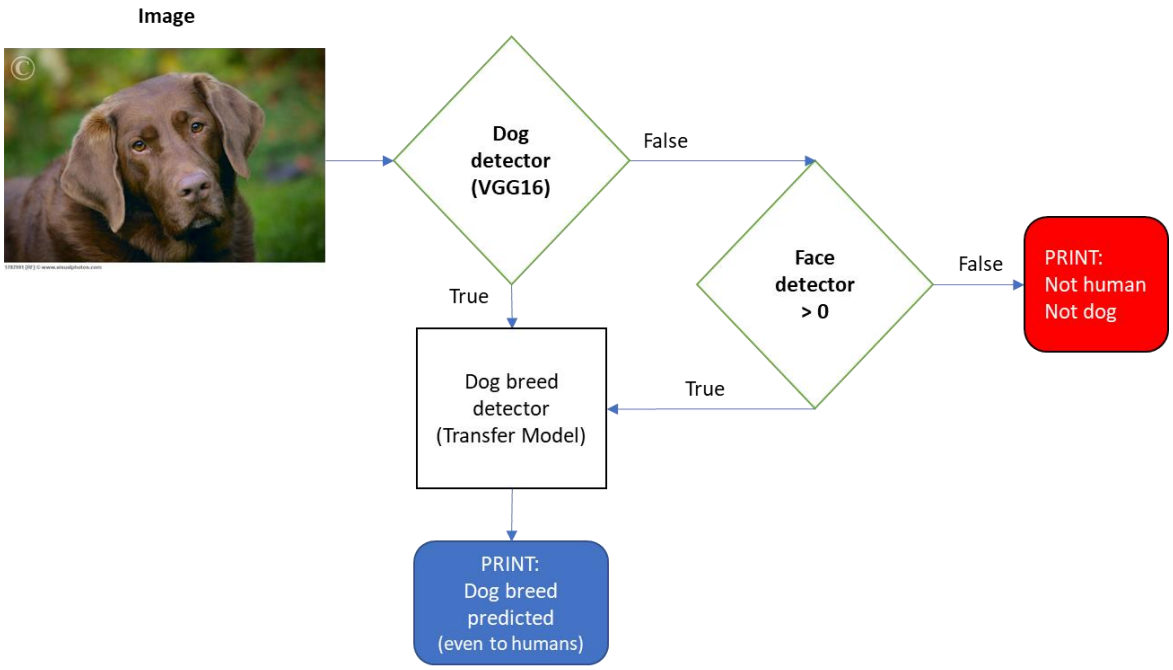
Resnet18, batch size=20, epochs=50:

- Train Loss: 1.044906
- Validation Loss: 1.015536



Predict Dog Breed

The final model for dog breed identification uses two detectors: dog detector and face detector as a filter to any given image. Both were previously implemented as part of the Dog project workspace. Then, the pre-trained model implemented is used when the image corresponds to a dog or a human. In any other case a sentence of error is printed.



3.3.Refinement

Model from scratch

Based on the results, there is overfitting after training the model. It is mainly due to the combination of many factors, such as:

- the size of the training dataset which is small.
- the architecture of the network.

To prevent overfitting, common solutions suggested are:

- increasing the size of the training set.
- early stopping.
- regularization.
- reducing the complexity of the network.

Batch Size

Training models for image classification takes some amount of time. Therefore, is very important to choose a batch size that converges the model faster along with accurate results. *For some reason Transform perspective did not work on the workspace at Udacity, so most of these calculations were made in a personal computer. As showed in the next table, after some tests, the best batch size for the model and train loaders is 20.

# Ephocs	Batch size	LR	Transforms Compose	Time	Loss in Data			Acc
					Train	Valid	Test	
10	10	0.0500	Horizontal(0),Perspective	**23min 22s**	**3.602**	**3.747**	**3.644**	**14%**
10	20	0.0500	Horizontal(0),Perspective	22min 56s	3.846	3.99	3.972	8%
10	**20**	**0.0500**	**Horizontal(0),Perspective**	**15min 1s**	**3.809**	**3.838**	**3.834**	**12%**
10	40	0.0500	Horizontal(0),Perspective	22min 3s	4.045	4.082	4.053	7%
20	5	0.0500		50min 28s	4.133	3.925	3.842	11%
20	10	0.0500		39min 54s	3.73	3.673	3.535	17%
20	10	0.0500	Horizontal(0),Perspective	43min 47s	3.895	3.54	3.555	14%
20	10	0.0500	Horizontal(0),Perspective	47min 37s	3.378	3.663	3.536	16%
20	10	0.0500	Horizontal(0),Perspective	44min 4s	2.862	3.519	3.433	18%
20	10	0.0600	Horizontal(0),Perspective	43min 38s	3.066	3.608	3.494	16%
20	**20**	**0.0500**	**Horizontal(0),Perspective**	**30min 36s**	**2.905**	**3.719**	**3.565**	**17%**
40	10	0.0500		1h 36min 37s	3.518	3.381	3.323	20%
40	10	0.0500	+perspective	1h 42min 26s	2.542	3.801	3.524	19%
80	10	0.0500		2h 40min 26s	3.397	3.44	3.16	23%

4. Results

4.1.Model Evaluation and Validation

The resnet18 pretrained model gives better results than the previous Alexnet pre-trained model mentioned in the proposal related to this report. Here are the results:

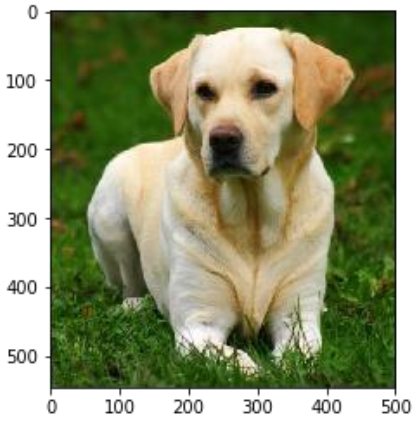
- Test Loss: 0.821012
- **Test Accuracy: 78% (653/836)**

78% of accuracy is very good achievement considering that the more epochs could be run. Note that this model is simple compared to others like Resnet101 or Resnet 152.

In the next table, other results changing epochs and batch size are showed:

# Ephocs	Batch size	LR	Transforms Compose	Time	Loss in Data			Acc
					Train	Valid	Test	
50	10	0.0010	Horizontal(0)	1h 31min 16s	0.834	0.703	0.821	78%
50	20	0.0010	Horizontal(0)	1h 21min 6s	1.045	1.016	1.130	74%
40	40	0.0010	Horizontal(0)	1h 2min 41s	1.819	1.795	1.925	67%
15	40	0.0010	Horizontal(0)	23min 56s	3.201	3.122	3.196	42%

Now, it is time to make some predictions using the algorithm showed in the implementation section.




```
Great news! Dog detected!
I think it is a Labrador retriever.
```


```
Well, Human detected
But as a dog you could be a Kerry blue terrier.
```

4.2.Justification

Training and evaluating models for multi-classification takes a great amount of time. The results shows that it is possible to have an accuracy above 75% with a simple pre-trained transfer model like Resnet18 and make some tests to compare results in different scenarios. Tunning a model also demands a great amount of time because the computation cost is very high even for this project with less than 10,000 images. It makes sense to use cloud computing for a large dataset.

The results of this project demonstrate the power of CNN, especially using transfer models, for multi classification of images. It really opens a new era of applications that can be designed for image classification in other fields.

References

- Capstone Project Example: https://github.com/udacity/machine-learning/blob/master/projects/capstone/report-example-3.pdf (Shibuya,2020).
- CNN-Benchmarks: https://github.com/jcjohnson/cnn-benchmarks (Johnson, 2017).
- Model from Scratch reference: https://medium.com/analytics-vidhya/classification-of-dog-breed-using-deep-learning-343f98ebebf0 (Agarwal, 2020).
- EDX. Deep Learning Essentials. UMontrealX, Mila and IVADO - IVADO-DL-101.
- GITHUB. Pytorch Learning Rate Finder. https://github.com/davidtvs/pytorch-lr-finder
- Batch Size. https://towardsdatascience.com/implementing-a-batch-size-finder-in-fastai-how-to-get-a-4x-speedup-with-better-generalization-813d686f6bdf
- MEDIUM. Transfer learning. https://medium.com/swlh/image-classification-tutorials-in-pytorch-transfer-learning-19ebc329e200