U.PORTO

FACULDADE DE CIÊNCIAS
UNIVERSIDADE DO PORTO

Professor Inês Dutra

Neural Networks to Distinguish Images

Advanced Topics in Artificial Intelligence

December, 2022

Pedro Leite - 201906697

Pedro Carvalho - 201906291

Index

1. Introduction

The main objective of this project is to find the best computacional method to distinguish pictures of animals. We'll start with a neural network model to distinguish pictures of dogs and cats. We'll then, respecting the tasks proposed by the teacher, add more methods to find out, if they'll help us find a more accurate model.

2. Background

The dataset has about 25000 pictures of dogs and cats (~12500 for each), where 75% is used for training and 25% is used for testing. The first model that we used is the VGG-16 model. After that, we added to the initial dataset, about 1000 pictures with noise, and analyzed the results. We then tried a different model from VGG: Resnet. And added a new class of animals to find if that would hurt the predictive ability of the model. Afterwards we used image description to train the model. And we concluded by applying reinforcement learning.

3. Materials

For this project we used Python3 with the following libraries: numpy, matplotlib, keras, h5py, opencv, pillow and bing-image-downloader.

The results are being run in a PC with the following specs: AMD Risen 5 2600X Six.Core Processor 3.6 GHz, 16GB RAM, Asus GeForce GTX 1660 Ti.

The files are: "taia.py" that creates a VGG-16 model (This file contains the creation of the model with and without the panda class, to generate one we comment the model.fit that concerns the other), "resnet.py" is the Resnet model, "ex7.py" is the model that uses the [5] dataset, "gaussian.py" generates the pictures with gaussian noise, "create_pandas.py" generates the pictures of pandas, "tester.py" gives us a model's accuracy.

Each model will run during 5 epochs. To analyze each method we calculated the accuracy in order to compare them, with the following formula:

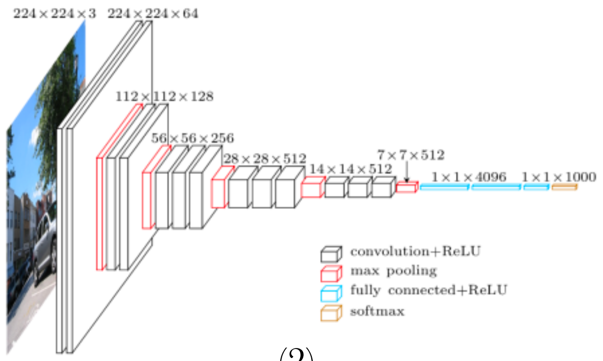
$$\text{Accuracy} = \frac{\text{Number of correct predictions}}{\text{Total number of predictions}} \tag{1}$$

4. Methods and Results

4.1. Our Firsts Models

The first proposed task, is to build a deep convolutional neural network model to classify pictures of dogs and cats, following a tutorial by Jason Brownlee [1].

The baseline VGG architecture, consists in stacking convolutional layers with filters followed by a max pooling layer, together, these layers form a block. Each layer will use the "reLU"activation function and the "He" weight initialization. It will use stochastic gradient descent, 0.001 learning rate and 0.9 momentum.



(2)

The first, second and third model are, respectively: the "one block VGG", the "two block VGG" and the "three block VGG", each is an extension of the previous, where the convolutional layer have, respectively: 32, 64 and 128 filters, followed by a max pooling filter. The accuracy of the models are, respectively: ~72%, ~76% and ~80%.

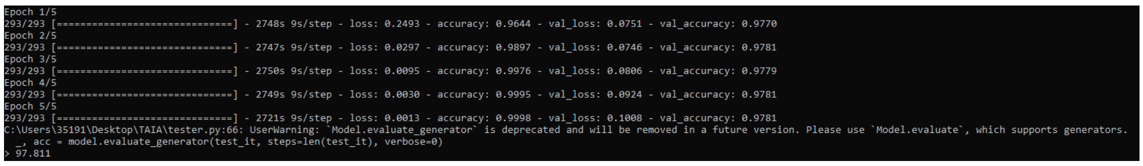
With this VGG models we can evaluate, that more blocks, means more accuracy. But it's clear, that the models show "overfitting", which means that we may have too much parameters. To regulate that we can use: "dropout regularization" and "data augmentation".

Dropout regularization, works by removing inputs to a layer. It shows improvement in accuracy with ~81%, and reduces overfitting.

Data augmentation, works by expanding the size of a training dataset by creating modified versions of images in the dataset. It shows improvement in accuracy with ~85%, and reduces overfitting.

The fourth (and the best) model, will explore "transfer learning" that uses all or parts of a model trained on a related task. For this method, we'll be using

VGG-16 (VGG model with 16 layers). The model is comprised of two main parts, the feature extractor part of the model that is made up of VGG blocks, and the classifier part of the model that is made up of fully connected layers and the output layer. The model removes the fully connected layers from the output end of the model, then adding the new fully connected layers to interpret thee model output and make a prediction: “include_top = False”, “input_shape = (224, 224, 3)”. Not a lot of training is required, as only the new fully connected and output layer have trainable weights, so the number of training epochs will be 5. The input pictures need to have 224x224 pixels. The pictures need to be centered, so e use: “featurewise_center = True” and “mean = [123.68, 116.779, 103.939]”. This model has an accuracy of ~97%, and loss of ~0.1%.



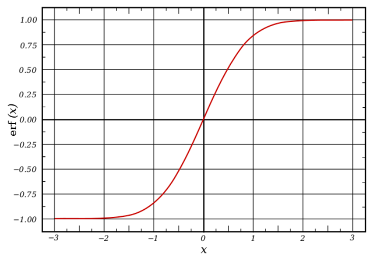
```
Epoch 1/5
293/293 [==============================] - 2748s 9s/step - loss: 0.2493 - accuracy: 0.9644 - val_loss: 0.0751 - val_accuracy: 0.9770
Epoch 2/5
293/293 [==============================] - 2747s 9s/step - loss: 0.0297 - accuracy: 0.9897 - val_loss: 0.0746 - val_accuracy: 0.9781
Epoch 3/5
293/293 [==============================] - 2750s 9s/step - loss: 0.0095 - accuracy: 0.9976 - val_loss: 0.0806 - val_accuracy: 0.9779
Epoch 4/5
293/293 [==============================] - 2749s 9s/step - loss: 0.0030 - accuracy: 0.9995 - val_loss: 0.0924 - val_accuracy: 0.9781
Epoch 5/5
293/293 [==============================] - 2721s 9s/step - loss: 0.0013 - accuracy: 0.9998 - val_loss: 0.1008 - val_accuracy: 0.9781
C:\Users\35191\Desktop\TAIA\tester.py:66: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
> 97.811
```

(3)

4.2. Using Images with Noise

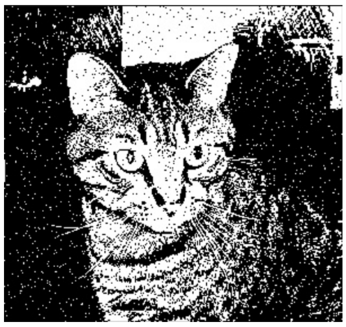
For this task, we had to generate images with noise. To generate this noise, we’ll use “Gaussian noise”, that consists in a probability density function equal to normal distribution.

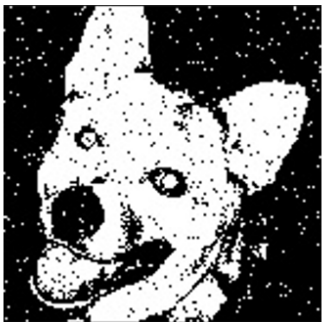
We converted the input image to binary. After that we created a (Gaussian random) normal distribution, with mean 0, and sigmoid 1, and converted it to binary. If the value of a pixel is above 2, we set the value of that pixel as 1. If not, we set the value of that pixel as -1. Just like the sigmoid function:



(4)

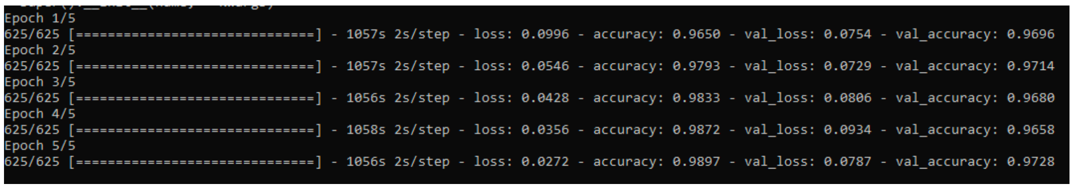
We generate the final picture using the newly generated pixels. Here are 2 examples:


(5)


(6)

4.3. Test the Model with the Pictures Generated with Noise

We added 1000 pictures to the dataset. This model has an accuracy of ~97%, and loss of ~0.1%.



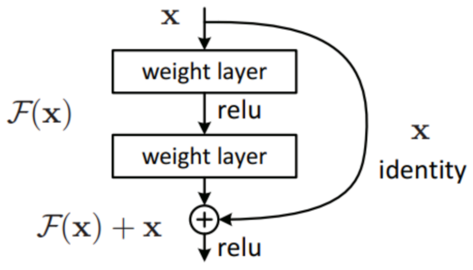
```
Epoch 1/5
625/625 [==============================] - 1057s 2s/step - loss: 0.0996 - accuracy: 0.9650 - val_loss: 0.0754 - val_accuracy: 0.9696
Epoch 2/5
625/625 [==============================] - 1057s 2s/step - loss: 0.0546 - accuracy: 0.9793 - val_loss: 0.0729 - val_accuracy: 0.9714
Epoch 3/5
625/625 [==============================] - 1056s 2s/step - loss: 0.0428 - accuracy: 0.9833 - val_loss: 0.0806 - val_accuracy: 0.9680
Epoch 4/5
625/625 [==============================] - 1058s 2s/step - loss: 0.0356 - accuracy: 0.9872 - val_loss: 0.0934 - val_accuracy: 0.9658
Epoch 5/5
625/625 [==============================] - 1056s 2s/step - loss: 0.0272 - accuracy: 0.9897 - val_loss: 0.0787 - val_accuracy: 0.9728
```

(7)

By the results, we can conclude that, if we add the noisy pictures, to the initial dataset with the "normal" pictures, the final accuracy will be very similar to the previous, this might be because the number of noisy pictures compared to the normal ones, is relatively low.

4.4. Train the Model with Networks Different from VGG

To train the model, with networks different from VGG, we'll use "Resnet" (specifically resnet50). Resnet or "residual network", introduced the concept called "residua blocks". We skip connection connects activations of a layer to further layers by skipping some layers in between. This forms a residual block. Resnets are made by stacking these residual blocks together. Instead of layers learning the underlying mapping, we allow the network to fit the residual mapping.



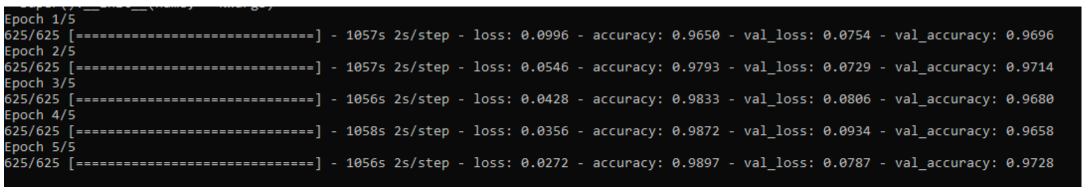
(8)

For the training data, we start by reformatting the pictures with the scale 200x200 pixels, the validation split is 0.2, this means that 80% of data will be reserved for training, while 20% will be use for validation, the batch size is 32, is the number of training example utilized in one iteration, subset is firstly a training subset. We do the same for the validation data, the only change is the subset attribute.

While importing the resnet50, we use “include_top = False”, this ensures that we can add our on custom input and output layers, “weights = ‘imaginet’” it will use the weights he learned from “imagenet”, and “layer.trainable = False”, this ensures that the model doesn’t learn the weights repeatedly.

Then we added, a fully connected and output layer, where we use 512 neurons and the “relu” activation function and 5 neurons and the “softmax” activation function, respectively.

This model has an accuracy of ~97%, and loss of ~0.7%.



```
Epoch 1/5
625/625 [==============================] - 1057s 2s/step - loss: 0.0996 - accuracy: 0.9650 - val_loss: 0.0754 - val_accuracy: 0.9696
Epoch 2/5
625/625 [==============================] - 1057s 2s/step - loss: 0.0546 - accuracy: 0.9793 - val_loss: 0.0729 - val_accuracy: 0.9714
Epoch 3/5
625/625 [==============================] - 1056s 2s/step - loss: 0.0428 - accuracy: 0.9833 - val_loss: 0.0806 - val_accuracy: 0.9680
Epoch 4/5
625/625 [==============================] - 1058s 2s/step - loss: 0.0356 - accuracy: 0.9872 - val_loss: 0.0934 - val_accuracy: 0.9658
Epoch 5/5
625/625 [==============================] - 1056s 2s/step - loss: 0.0272 - accuracy: 0.9897 - val_loss: 0.0787 - val_accuracy: 0.9728
```

(9)

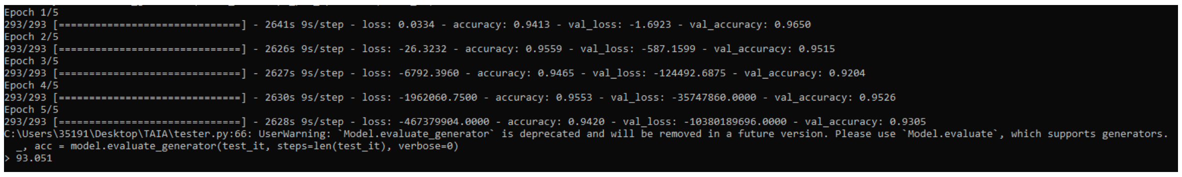
4.5. Add a New Class of Animals

To add a new class of animals (pandas), to the already existing dataset of pictures of dogs ands cats, we used a library that uses the Bing search engine. With the following conditions: “limit = 200”, “adult_filter_off = True”, “force_replace = False” and “timeout = 60.

4.6. Train the Model Again

4.6.1. Can you classify well when you add other classes of animals?

Adding another class to the dataset, the accuracy lowered from ~97% to ~93%. So it makes the pictures less predictive.



```
Epoch 1/5
293/293 [==============================] - 2641s 9s/step - loss: 0.0334 - accuracy: 0.9413 - val_loss: -1.6923 - val_accuracy: 0.9650
Epoch 2/5
293/293 [==============================] - 2626s 9s/step - loss: -26.3232 - accuracy: 0.9559 - val_loss: -587.1599 - val_accuracy: 0.9515
Epoch 3/5
293/293 [==============================] - 2627s 9s/step - loss: -6792.3960 - accuracy: 0.9465 - val_loss: -124492.6875 - val_accuracy: 0.9204
Epoch 4/5
293/293 [==============================] - 2630s 9s/step - loss: -1962060.7500 - accuracy: 0.9553 - val_loss: -35747860.0000 - val_accuracy: 0.9526
Epoch 5/5
293/293 [==============================] - 2628s 9s/step - loss: -467379904.0000 - accuracy: 0.9420 - val_loss: -10380189696.0000 - val_accuracy: 0.9305
C:\Users\35191\Desktop\TAIA\tester.py:66: UserWarning: `Model.evaluate_generator` is deprecated and will be removed in a future version. Please use `Model.evaluate`, which supports generators.
  _, acc = model.evaluate_generator(test_it, steps=len(test_it), verbose=0)
> 93.051
```

(10)

4.6.2. What if the classes are imbalanced?

When downloading the pictures, a problem that we faced, is that we also downloaded different images not related with pandas directly (like, movies, restaurants, shoes, etc), so a manual filtering was needed. Our code that downloaded the panda images also took a very long time to download after ~20 images and after that, if we ran it again we would end up downloading the same ones we did before. We tried bypassing this by searching for "pandas", "panda animal", etc… to gather more images. Making the dataset imbalanced (we have much less pictures of pandas than dogs/cats).

4.6.3. Does it matter to humans? Does it matter to the computacional models?

An imbalanced dataset doesn't really matter to humans, because humans only need a small sample size to identify the class object. In the other hand, computacional models need large sample sizes, so the imbalanced classes affects them in a negative way as we can see from the accuracy reduction our model suffered.

4.7. Train the Model Using Image Description

We used the dataset from [5], that consists in a set of pictures and a CSV file, with an ID for each picture, a set of attributes and how popular the animal would be to be eventually adopted.

For this type of data we created, unsuccessfully, a new sequential model, with 4 layers, where the first 3 layers has: 128 neurons and uses the activation function “tanh”, and the last uses 1 neuron and the activation function “softmax”.

4.8. Apply Reinforcement Learning

4.8.1. How would you represent it and what is the reward function?

We also couldn’t implement this task, but we would use a similar approach to [6] where they create a Reinforcement Learning agent which adaptively selects the resolution of every image provided by the detector. They train the agent with double rewards by choosing lower resolution images for a coarse level detector in case of the image being dominated by large objects, and higher resolution images for a fine level detector in case of the image being dominated by small objects. Regarding the image detection the approach would be the same as suggested by [7] where instead of the classical methods such as the “Sliding Window Formulation” that consists in splitting the image in a large set of rectangles and the problem is solved by exhaustive search which is very consuming (time and computationally wise), they suggest we follow a more biological and natural to humans approach, where a small set of scene locations are investigated sequentially, in order to accumulate sufficient evidence on the target location. This is clearly more efficient than the prior approach because because only a few regions are explored. This is also much more biologically plausible because as humans we don't need to search every region of an image when trying to detect an object contained in it. They achieve this by operating with delayed rewards, which rules out supervision at each step.

5. Discussion and Conclusion

We can conclude, that the two main models that we developed: VGG-16 (with dropout regularization and data augmentation) and Resnet, have very similar accuracy results, around ~97%, but the VGG model showed better results in terms of loss. Adding pictures with noise will worsen the predicative ability of the dataset, but because we generated a low sample of noisy pictures, in our testing, it didn't affect much the dataset. When we added a new class of animals, composed by pictures of pandas, it hurt the accuracy of the model from ~97% to ~93%, also because the dataset becomes very imbalanced with 12500 pictures of dogs, 12500 pictures of cats and just 1000 pictures of pandas. We weren't successful in executing the last two tasks.

In the future, to distinguish and identify pictures, we would use the VGG-16 model for his high accuracy and low loss results while keeping the number of classes as low as possible, and the pictures in the dataset as clear as we can.

6. Bibliography

[1] https://machinelearningmastery.com/how-to-develop-a-convolutional-neural-network-to-classify-photos-of-dogs-and-cats/

[2] https://www.geeksforgeeks.org/residual-networks-resnet-deep-learning/

[3] https://chroniclesofai.com/transfer-learning-with-keras-resnet-50/

[4] https://pypi.org/project/bing-image-downloader/

[5] https://www.kaggle.com/competitions/petfinder-pawpularity-score

[6] Image Classification by Reinforcement Learning with Two-State Q-Learning, Abdul Mueed Hafiz

[7] Reinforcement Learning for Visual Object Detection, Stefan Mathe, Aleksis Pirinen, Cristian Sminchisescu