Techniques of Artificial Intelligence: Image classification using Convolutional Neural Networks

Chris Adam[1], Charlotte Hendrickx[1] and Olivier Renson[1]

[1] Université Libre de Bruxelles, Masters in Bioinformatics and Modelling

Abstract

Image classification has proven useful in a number of areas like cancer diagnosis, species identification, text recognition,... In this work, we tried to classify images of cats and dogs using a Convolutional Neural Network (CNN). Convolutional Neural Networks are extensively used in computer vision problems and have yielded satisfactory results in various applications. This report presents our implementation of a CNN in order to classify cats and dogs images. We will first describe the data and the methodology we applied as well as different features we implemented in our program. We will then comment our results and compare them with the results obtained by another group that applied Reinforcement Learning on the same dataset.

Introduction

Convolutional Neural Networks (CNN) are an expansion of Neural Networks that have been extensively used in computer vision applications. CNNs are used to capture local features (ex. neighbouring pixels forming a border) and to reduce the complexity of the model.

In opposition to classical neural networks, CNN contain different types of hidden layers: *convolutional layers* and *pooling layers*. The general structure of a CNN is as follows: an input layer connected to a number of convolution and pooling layers, followed by a fully connected layer and an output layer (see Figure 1).

Convolutional layers Convolution is a mathematical operation that enables to extract the local features of an element (e.g. an image) and thereby reduced the dimmensionality of our input. It takes an input I and a kernel K (an array of parameters acting as a filter) as arguments and outputs a feature map $S(i, j)$. With m and n being the number of rows and columns of the input, equation 1 expressed the general idea of a convolution:

$$S(i,j) = (I \cdot K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{1}$$

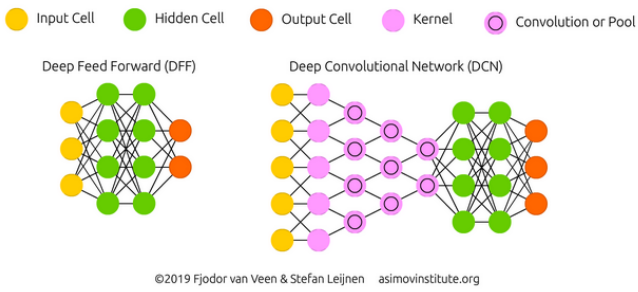

Figure 1: Comparison of a deep neural network (left) and a convolutional neural network (right). Adapted from: The Asimov Institute

Graphically, we could understand a convolution operation as shown in Figure 2. The input is shown in green, the kernel are the red numbers. We slide the kernel over the input and multiply them. We then take the sum (or another function like the mean), which gives the output of the convolution, called the *feature map*.

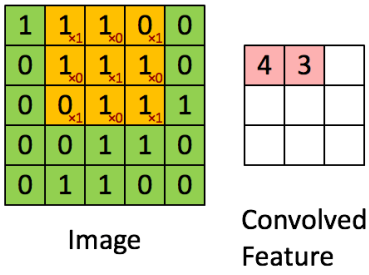

Figure 2: Convolution operation. Reproduced from: Stackexchange

Pooling layers The feature map obtained after a convolution operation can still be very large. We can then apply a pooling function (= downsampling), which aggregates statistics of these features at different locations. For example, in *max-pooling*, we take the maximal value of a set of the feature map. (See Figure 3)

Apart from reducing the dimmension, pooling also re-

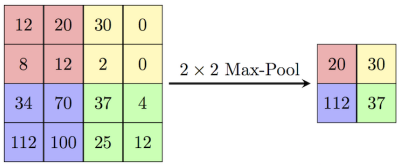

Figure 3: Example of a max-pooling operation. Reproduced from Computerscience Wiki

duces the overfitting. In pooling, we only retain the maximum (or mean) of the feature map, which thus becomes invariant to small translations of the input.

Material and methods

Data

Because different groups of students chose the Dog-Cat classification problem, we had to define a uniform benchmark dataset in order to compare the results. We used a Kaggle dataset composed of 10,000 images: 4,000 images of each class used as training set and 1,000 images of each class used as testing set.

However, we decided to make our own splitting of the data by putting all images together and using a 15% subset of the images, corresponding to 1,500 images, as a validation set, 25% (2,500 images) as an independent testing set and the remaining 6,000 (60%) images were used for training.
In order to obtain a better accuracy on transformed images, we used different techniques of data augmentations. The techniques include: flipping the image horizontally, shifting the image, zooming in, rotating it and shearing the image. These were randomly applied at each epoch on each image in the training and validation set. This data-augmentation was performed using the *ImageDataGenerator* function of the *keras* package.

Construction of the Convolutional Neural Network

The construction of the Convolutional Neural Network has been done empirically in order to maximise the accuracy on the validation set. The model was implemented using the *Tensorflow* library and the code is available here.

The CNN model contains five convolutional layers, with a decreasing amount of 3 by 3 kernels (128-128-64-64-32). The ReLu activation function was used after each layer in order to avoid the vanishing gradient problem. A maxpooling operation was performed after each convolution layer with a 2 by 2 pixel window.
The number of these layers were modified and using five layers proved to give the best accuracy on the validation set.

The image obtained as output of these five convolutions was then flattened out to one dimmension and passed through a fully connected layer of 128 neurons with a ReLu activation function. In order to avoid overfitting, the output of the fully-connected layer was passed through a dropout layer and 20% of the inputs were randomly set to 0. Finally, the final cat-dog classification was achieved by passing the output to a dense layer of only one neuron, with a sigmoid activation function.

The loss function we used is the binary cross-entropy function, because we here face a binary classification problem. The weights were updated using the RMSprop (Root Mean Square propagation) optimizer function with a learning rate of $10 \cdot 10^{-3}$. We experimented some changes in the learning rate. As expected, a higher learning rate lead to faster convergence, but a lower accuracy. Lower learning rates weren't used, because we wanted to obtain a relatively fast convergence.

Training

In order to train our model, we performed 100 training epochs by passing our dataset through the network in batches of 64 images. The training of the network took approximately 5h30 and was performed on the *Dragon2* computing cluster in order to use GPU units. The weights of the network were saved after each epoch in order to load the trained model.

Results

After training for 100 epochs, we were able to obtain a classification accuracy of about 93% on the testing set (figure 4).

Expansions on the project

In order to present a coherent project, we developed a small GUI that enables to input an image and get back a prediction. Because convolutions can be an abstract concept, we also added a visualisation tool that enables to apply a number of convolutions to an image using a chosen kernel.
The GUI has been made in python using the *PyQt* package.

Prediction tool

The main function of the GUI is to enable the user to input an image and get a prediction if it is a cat or a dog. The *Select picture(s)* button enables to select one or multiple images (see Figure 7). The *Select model* button enables to select a neural network to which we will pass the images. The CNN model we constructed is defined as the *Custom* model. Once the image and the model have been selected, we can make a prediction (with the *Predict* button), which will appear under the image. When using the GUI on multiple images, it can be useful to export the prediction results to a ".csv" file. This can be done using the *Export* button.

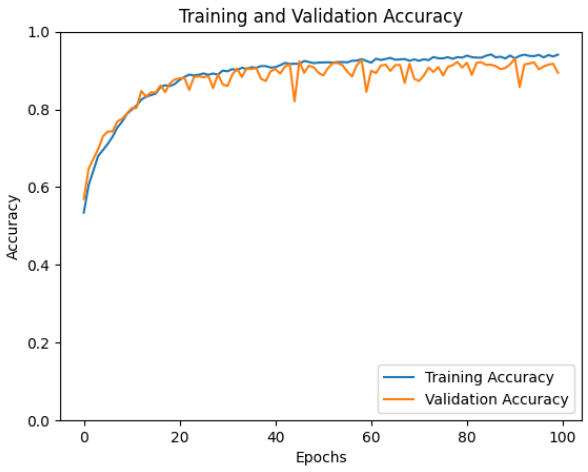

Figure 4: Behaviour of accuracy in the training and the validation sets during the training

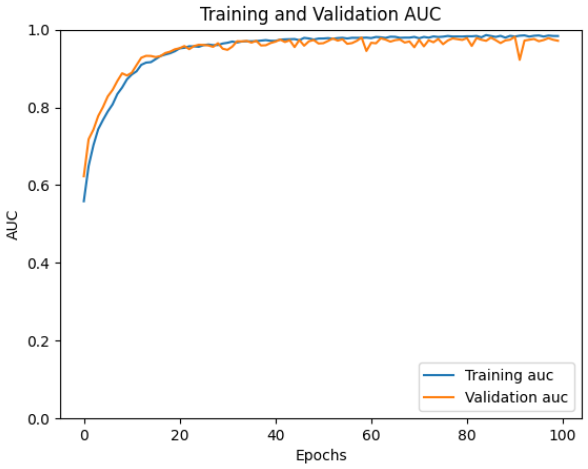

Figure 5: Behaviour of AUC in the training and the validation sets during the training

Visualisation of the convolution operations

We also provide the possibility of using this GUI as a visualisation tool of the convolution operations. Once an image is loaded using the *Load image* button like described previously, we can use the *Apply convolution* button to visualize the output of the convolution of this image. First, one should define the kernel you want to use, the default being the *identity* kernel and the number of convolution and max-pooling steps you want to perform, the default being one. The different possibilities are shown in Figure 8.

Figure 9 shows the result of applying three consecutive convolutions using the "*blur*" kernel and max-pooling. This enables users to have a practical approach of convolutions and play with different parameter settings.

References

Andrew, N., Jiquan, N., Chuan Yu, F., Yifan, M., Caroline, S., Adam, C., Andrew, M., Awni, H., Brody, H., Tao, W., and Sameep, T. Deep learning tutorial. http://ufldl.stanford.edu/tutorial/supervised/Pooling/.

Education, I. C. Convolutional neural networks. https://www.ibm.com/cloud/learn/convolutional-neural-networks.

Education, I. C. Neural networks. https://www.ibm.com/cloud/learn/neural-networks.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Van Veen, F. Leijnen, S. (2019). The neural network zoo. https://www.asimovinstitute.org/neural-network-zoo.

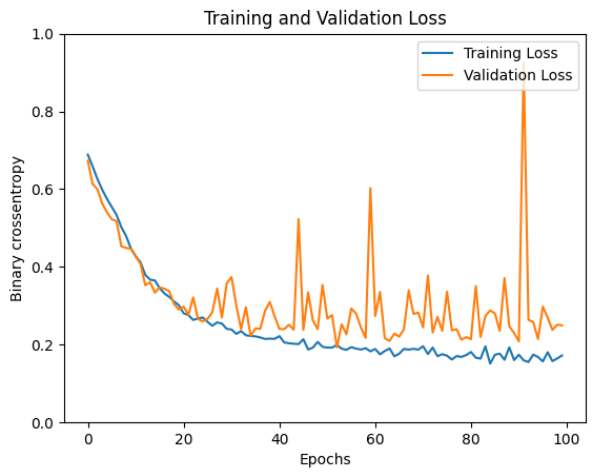

Figure 6: Behaviour of loss in the training and the validation sets during the training

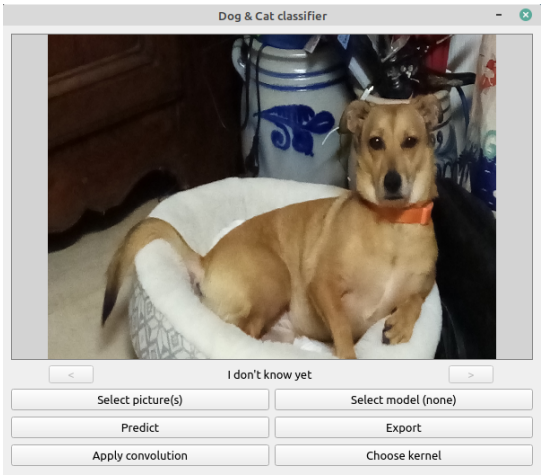

Figure 7: GUI after loading an image (of the wonderful dog of Charlotte)

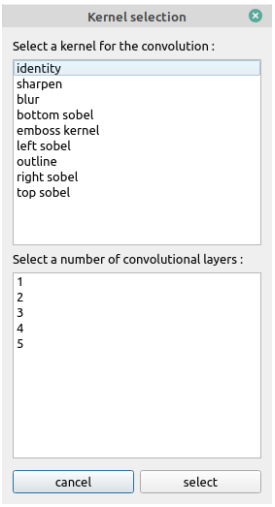

Figure 8: Different choices of kernels and number of convolution + max-pooling operations available.

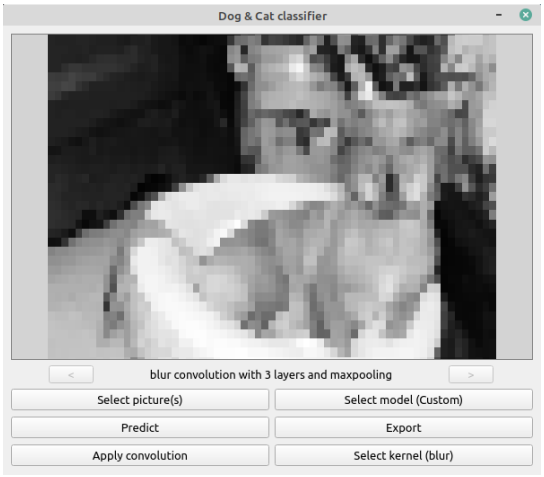

Figure 9: Result of three consecutive convolutions using a 3 by 3 *blur* kernel (stride=(2,2), padding="same") with max-pooling using a 2 by 2 window.