# Improving a CNN based Image Classifier

Aissa Abdelaziz
University of Paris Saclay
Evry, France
aissa.abdelaziz@univ.etud-evry.fr

## I. INTRODUCTION

Convolutional Neural Networks (CNNs) have revolutionized the field of computer vision by achieving state-of-the-art performance in various fields starting from the object recognition, Visual Recognition [1]. in addition, several studies have already been published in many other areas such as segmentation, image reconstruction [2], and natural language processing.

The aim of our project is to design a CNN architecture to be able to perform image classification of two types of animals "Cows and Horses" by improving its accuracy as much as possible and solve the problem of missing data by developing many techniques such as Generative Adversarial Network to perform data augmentation using the available data set.

## II. CNN ARCHITECTURE

In recent years, various improvements in CNN learning methodology and architecture were performed to make CNN scalable to large data such as images. For this reason, the researchers developed different designs such as Wide ResNet, ResNeXt, Pyramidal Net, Xception, PolyNet, and many others. However, most of the architectures typically consists of same components: a series of convolutional layers, followed by pooling layers, and finally fully connected layers. The convolutional layers are responsible for extracting features from the input images by performing convolutions with learnable filters. The pooling layers reduce the spatial dimensionality of the feature maps by performing operations such as max pooling, which selects the maximum value within a local region of the input.

### A. Basic CNN components

*a) Convolutional layer: The convolution layer is composed of a set of convolutional kernels where each neuron acts as a kernel.*

*b) Pooling layer: It is used to reduce the number of parameters and computational complexity of the model. And the most common type of pooling operation is max-pooling*

*c) Activation function: it is used to transform the output of the previous layer into a new representation that can be fed to the next layer.In literature, different activation functions such as sigmoid, tanh, maxout, SWISH, ReLU, and variants of ReLU, such as leaky ReLU, ELU, and PReLU are used to allow the model to learn complex and nonlinear relationships between the input and output data.*

*d) Batch normalization: The technique involves normalizing the activations of the previous layer for each mini batch of data during training, which helps to speed up the training and improve the stability of the model.*

*e) Dropout : It is used to improve the generalization of the network by randomly skipping some units or connections with a certain probability.*

*f) Fully connected layer: It is used in the later stages of a neural network where every neuron in the previous layer connected to every neuron in the current layer.*

## III. IMAGE CLASSIFIER

### A. Tools Used In This Project

Google Collab: it is a cloud-based platform that provides a free environment for writing and executing code in Python, we used it because it provides easy access to powerful hardware resources, such as GPUs.

PyTorch: is an open-source machine learning library for Python that is developed by Facebook's AI Research team. It is used for building and training deep neural networks.

Data set file: Our data set is divided into two file train and test and each file contains 41 images of cows and horses of size 256 by 256 pixel.

Google Drive: It is a cloud-based file storage and synchronization service provided by Google. We used it to store our data.

Python: It is a popular high-level programming language that is widely used in a variety of fields, including artificial intelligence.

### B. Building the model

#### 1) Installing all the packages

In this first step, we prepare our environment to start developing by installing all the packages and import all the libraries needed for this project:

- torch: it contains data structures for multi-dimensional tensors and defines mathematical operations over these tensors.
- torchvision:it consists of model architectures, and common image transformations.
- Matplotlib: it is a famous library to plot graphs in Python.
- Numpy: it is the fundamental package for scientific computing with Python.

#### 2) Labeling and Importing the data

In this step, we will label our images of cows to "1" and horses to "0" by defining a "CustomDataset" class and pass the argument Dataset.

After that, In order to load the images we use a predefind function DataLoader from torch.utils to perform this task by specifying the data set and the batch size and the number of workers .

*3) Designing the architecture of the classifier*

The input to the network is a 3-channel RGB image. The first layer is a 2D convolutional layer with 16 filters of size 3x3, which produces an output volume of size 62x62x16.
The output of the first convolutional layer is then passed through a max pooling layer with a kernel size of 2x2, reducing the spatial dimensions by half to 31x31x16.
The second layer is another 2D convolutional layer with 32 filters of size 2x2, which produces an output volume of size 30x30x32.
A dropout layer with a probability of 0.5 is applied to the output of the second convolutional layer to prevent overfitting. The output of the second convolutional layer is then passed through another max pooling layer with a kernel size of 2x2, reducing the spatial dimensions by half again to 15x15x32.The third layer is another 2D convolutional layer with 8 filters of size 2x2, which produces an output volume of size 14x14x8.
An activation function (ReLU) is applied to the output of each convolutional layer.
The output of the third convolutional layer is flattened into a 1D vector of length 1352, which is then passed through two fully connected layers with 800 and 200 neurons respectively.
A dropout layer with a probability of 0.15 is applied to the output of the first fully connected layer to prevent overfitting.
Another activation function (ReLU) is applied to the output of the first fully connected layer.
The output of the second fully connected layer is then passed through another dropout layer with a probability of 0.5.
Finally, the output of the dropout layer is passed through another fully connected layer with 2 neurons (for binary classification of "Cow" or "Horse") and a SoftMax activation function is applied to produce a probability distribution over the two classes.

$$n_{out} = \left\lfloor \frac{n_{in} + 2p - k}{s} \right\rfloor + 1$$

$n_{in}$: number of input features
$n_{out}$: number of output features
$k$: convolution kernel size
$p$: convolution padding size
$s$: convolution stride size

*Equation 1: Formula for output dims calculation*

ReLU stands for Rectified Linear Unit, which is an activation function commonly used in neural networks. It is defined as:

$$f(x) = \max(0, x)$$

Softmax is an activation function commonly used in the output layer of a neural network. It is used to convert the output of the last layer of neurons into a probability distribution over the possible classes.
The softmax function is defined as follows:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

*Equation 2: The SoftMax function*

The main reason of using convolution neural network is to perform feature extraction from the images and reduce the size of feature map at the end to fit it through an ANN network to apply a classification task.

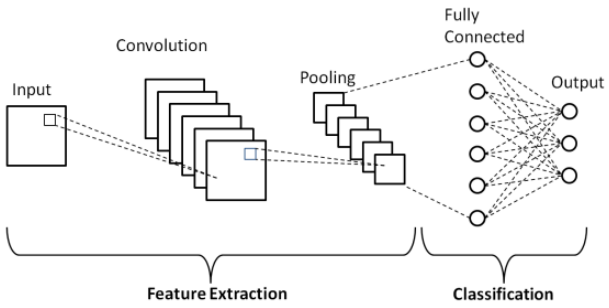


*Figure 2: Architecture of a Convolutional Neural Network (CNN)*

*4) Training the model*

To train a CNN model we must choose the loss function and the optimizer and define some hyper parameters in order to perform the training properly.
For this project we chose Stochastic gradient descent as an optimizer which is a popular optimization technique used in machine learning for finding the optimal parameters (weights and biases) in small batches, based on the loss calculated on each batch.

**Function** SGD:
  Set epsilon as the limit of convergence
  **for** $i = 1, \ldots, m$ **do**
    **for** $j = 0, \ldots, n$ **do**
      **while** $|\omega_{j+1} - \omega_j| < epsilon$ **do**
        $\omega_{j+1} := \omega_j - \alpha \cdot (h_\omega(x^{(i)}) - y^{(i)}) \cdot x_j^{(i)}$;
      **end**
    **end**
  **end**

*Stochastic gradient descent Algorithm*

To calculate the loss at each iteration, we used cross-entropy loss. It measures the difference between the predicted probability distribution and the true probability distribution of the target class.
The formula for cross-entropy loss is:

$$L = -\frac{1}{m} \sum_{i=1}^{m} y_i \cdot \log(\hat{y}_i)$$

*Equation 3: formula for cross-entropy loss*

Finally , we define some hyperparameters for our model which will effect its performance during training and can have a significant impact on the final results. Some hyperparameters include the learning rate to be 0.01,

momentum fixed at 0.99, weight decay value is 0.05, the batch size equal to 4, and the number of epochs set to 20.
After defining all these parameters, we start our training process on the train images and then we verify the accuracy of our model using the test images.

### C. The result

After testing the performance of our model, we end up with results mentioned in the table below:

| Accuracy on the train images | Accuracy on the test images |
|---|---|
| 95% | 55% |

### D. Discussion

We can notice that the model is not performing well, and the accuracy is very low, we assume that the reason is because we do not have enough train data set to train our model and learn more about the features and the difference between the cows and the horses images.
We suggest performing a data augmentation technique in order to solve this problem and improve the performance of the model.
The common techniques are:

#### 1) Random transformations:

We apply many transformations on our set of images using torchvision.transforms module and at each time we fetch randomly from our data set images by activating the parameter shuffle of the DataLaoder .
There are many types of transformations where we can modify the geometry, color, composition, conversion …
The result:
After applying these methods, the accuracy of the model improved by 5%.

#### 2) Develop a Generative Adversarial Network

Generative Adversarial Network is a deep learning model developed by Ian Goodfellow and his colleagues in June 2014 [3]. This model is able to learn from the distribution of the training data how to generate new images from the same distribution. GAN composes of two deep networks, the generator, and the discriminator.
The goal of the generator is to generate new images and try to outsmart the discriminator by generating a realistic image that have the same distribution .in the other hand , the goal of the discriminator is to distinguish between the real and the fake images generated by the generator .after training this model the generator will be able to generate a better images which will be used to train our CNN classifier latter [4].

##### a) Generator

The generator, G, is designed to generate an RGB image starting from a latent vector z. G's architecture contains a series of strided two-dimensional convolutional transpose layers, 2d batch norm layer and a relu activation.

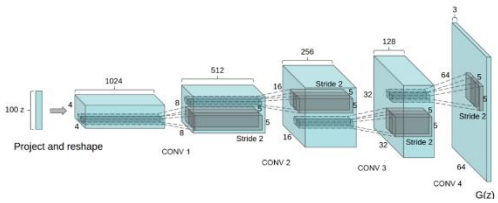


*Figure 3: Architecture of the Generator*

Transposed convolutions are standard convolutions but with a modified input feature map.

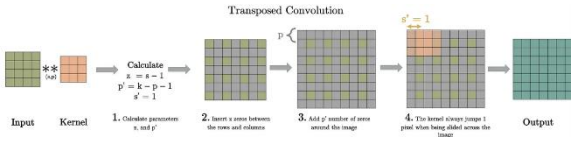


*Figure 4: Transposed convolutions*

##### b) Discriminator

The discriminator, D, is a binary classification network that takes the image as an input and returns if it's real or generated (fake). D architecture contains a series of Conv2d, BatchNorm2d, and LeakyReLU layers, Sigmoid activation function.

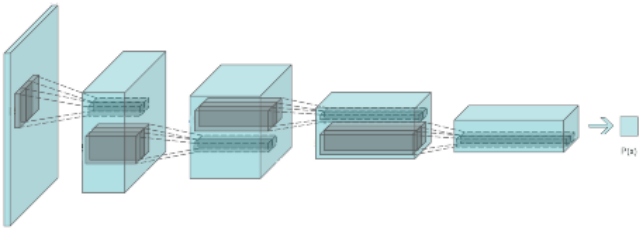

*Figure 5: Architecture of the Discriminator*

The aim of using LeakyReLU layers is to allow a small, non-zero gradient for negative input values.

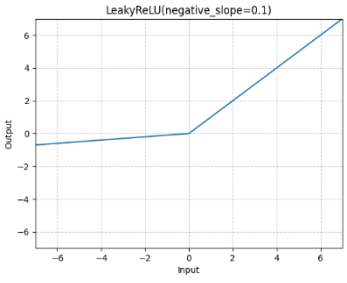


*Figure 6: LeakyReLU Activation Function*

#### 3) Training GAN:

As described in Goodfellow's paper[3], the idea to train the network is by playing minimax game such that, the discriminator will try to maximize the probability of correctly

classifying the a given input as real or fake , Practically, we want to maximize log(D(x))+log(1−D(G(z))) .and the generator will try to minimize the probability of classifying the generated images as fake by the discriminator. Practically, we want to minimize log(1−D(G(z))).

**Algorithm 1** Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, $k$, is a hyperparameter. We used $k = 1$, the least expensive option, in our experiments.

**for** number of training iterations **do**

  **for** $k$ steps **do**

  - Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Sample minibatch of $m$ examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
  - Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[ \log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  **end for**

  - Sample minibatch of $m$ noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  - Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

**end for**

The gradient-based updates can use any standard gradient-based learning rule. We used momentum in our experiments.

the GAN loss function is :

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}\left[log D(x)\right] + \mathbb{E}_{z \sim p_z(z)}\left[log(1 - D(G(z)))\right]$$

*Equation 4: GAN loss function*

*4) Results*

After training the model ,we were able to generate starting from noisy image at the 1st itiration a vey good images at the 30th itiration. However, in some cases the model output a weird image of cow or horse .

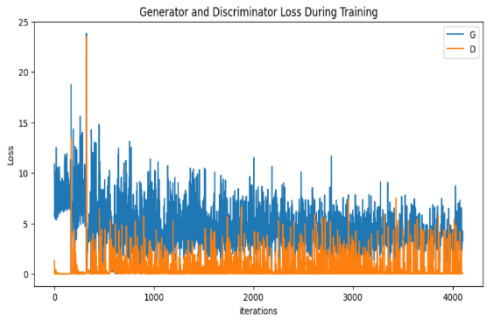


*Figure 7:Generator and Discriminator Loss During Training*

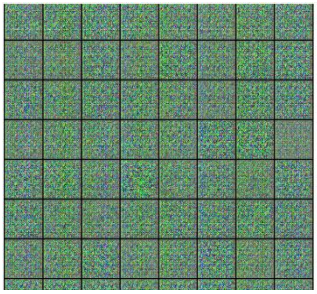

*Figure 7 Images at 1st iteration*

*Figure 8: Images After 200 iterations*

After training the GAN , we generated 500 new images of cows and 500 new images of horses by generating a 500 random latent vector z of length 10 and fit them through our generator .

*5) Perfoming Data Augmentation*

After generating 1000 new images of cows and horses, we saved them in the original train dataset file and retrain the CNN classifier using the new data set.
After finishing the training processes, we tested the accuracy of the model, but we did not notice any change which means that the model is not performing as we want.

## *E. Discussion*

We can observe from the result obtained that the model is not correctly predicting the type of the animal because of the low accuracy of image, or the architecture of our CNN classifier is not built for this type of classification.

## IV. Conclusions and Future Work:

During this project, we developed many types of CNN architecture, and we tried many types of training algorithms and data augmentations techniques. However, we could not improve the accuracy as much as possible due to our limited experience and the available data set.
We can suggest as solution for this problem:

- Generating high resolution images using GAN
- Generating more images and applying more transformations.
- Trying different activation function
- Spend more time tunning the hyperparameters.

## V. Acknowledgments

We would like to express our sincere gratitude to our professor Hedi Tabia for his invaluable guidance and expertise throughout this project. His insightful guidance and explanation have greatly contributed to the success of our work. We are truly grateful for his time and effort in monitoring us.

## VI. References

[1] Deep Learning to Classify Radiology Free-Text Reports Paper, Matthew C Chen .
[2] Very Deep Convolutional Networks for Large-Scale Image Recognition Paper, Karen Simonyan & Andrew Zisserman.
[3] Generative Adversarial Networks Paper, Ian J. Goodfellow
[4] DCGAN tutorial, Nathan Inkawhich