Università degli Studi di Verona

Computer Engineering for Robotics and Smart Industry

Binary classification with different CNN and SVM

July 13, 2021

Alberto Falezza
alberto.falezza@studenti.univr.it
VR462704

Contents

MOTIVATION AND RATIONALE

It is always been fascinating how our brain can easily recognize and distinguish the objects in an image. Nowadays, machines can easily distinguish between different images, detect objects and faces, and even generate images of people who don't exist. I will try to show how to develop a deep Convolutional Neural Network to classify photographs of dogs and cats. Although the problem sounds simple, it was only effectively addressed in the last few years using deep learning convolutional neural networks. I will try to get the maximum accuracy in estimating which pets they are. This is just a nuance of what we could do, but it is good to always start from the beginning. The dataset that has been used is taken from `Kaggle`, a dataset with 12500 pictures of dogs and 12500 pictures of cats.

Figure 1: Dataset of 25000 images labeled

Figure 2: After binary classification, desired results

Next two images are not treated in this document, but it's good to know how powerful it could be with a deeper implementation.

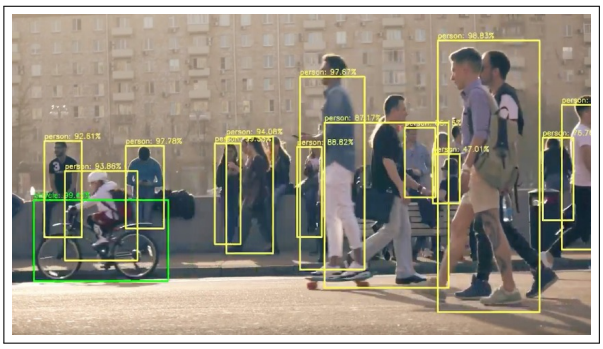


Figure 3: Detection of people with bounding box

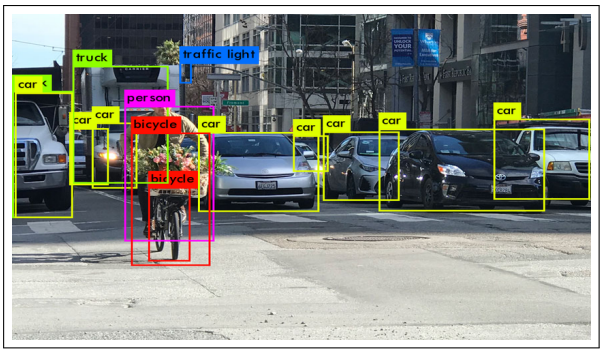


Figure 4: Object detection with bounding box and class label outputs

This type of problem absolutely dominates machine learning and artificial intelligence. Binary classification, the predominant method, sorts data into one of two categories based on a classification rule. We could see this as a True or False problem. The machine has to decide whether an element is in the first class or in the second. We use binary classification in various problems, like medical testing, to determine if a patient has a certain disease or not. Other usage is in industrial quality control, deciding if we have met a certain specification, is even used in information retrieval, for deciding whether a page should be in the result set of a search or not. In our case we have two symmetrical groups, but in many practical binary classification problems, the two groups are not symmetric, and rather than overall accuracy, the relative proportion of different errors is of interest. For example, in medical testing, detecting a disease when it is not present is

considered differently from not detecting a disease when it is present. When the classification function is not perfect, false results will appear. We can use 4 different metrics to express the 4 different probable outcomes.

- **False positives:** False positives result when a test falsely (incorrectly) reports a positive result. For example, a medical test for a disease may return a positive result indicating that the patient has the disease even if the patient does not have the disease.
- **False negatives:** On the other hand, false negatives result when a test falsely or incorrectly reports a negative result. For example, a medical test for a disease may return a negative result indicating that patient does not have a disease even though the patient actually has the disease.
- **True positives:** True positives result when a test correctly reports a positive result. As an example, a medical test for a disease may return a positive result indicating that the patient has the disease. This is shown to be true when the patient test confirms the existence of the disease.
- **True negatives:** True negative result when a test correctly reports a negative result. As an example, a medical test for a disease may return a positive result indicating that the patient does not have the disease. This is shown to be true when the patients test also reports not having the disease.

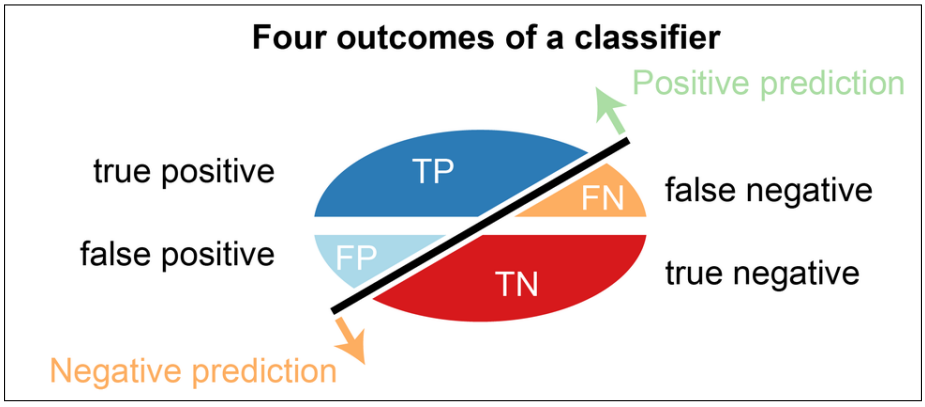


Figure 5: Basic evaluation measures from the confusion matrix

In the next section, I'm going to talk about all the Nets that are going to be used in this document. I choose these Nets because are the most interesting in terms of results, and the most promising for these tasks. We are going to learn the structure of some CNN, a SVM and understand how they work.

STATE OF THE ART

In this article, I will cover in total 3 models for image classification that are widely used in the industry as well. I built a CNN from scratch, used one of the top pre-trained models and as last a SVM that uses HOG features.

CNN built from scratch

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural network, most commonly applied to analyze visual imagery. CNN is a type of neural network model which allows us to extract higher representations for the image content. Unlike the classical image recognition where you define the image features yourself, CNN takes the image's raw pixel data, trains the model, then extracts the features automatically for better classification.

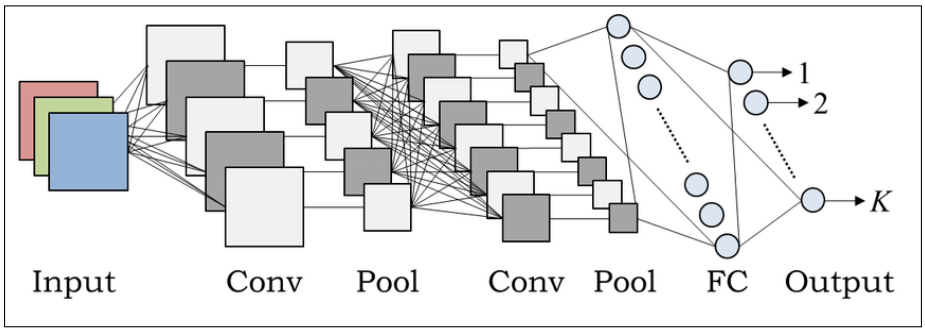


Figure 6: An example of CNN architecture — `Source`.

Convolution Layer

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs, such as image matrix and a filter or kernel.

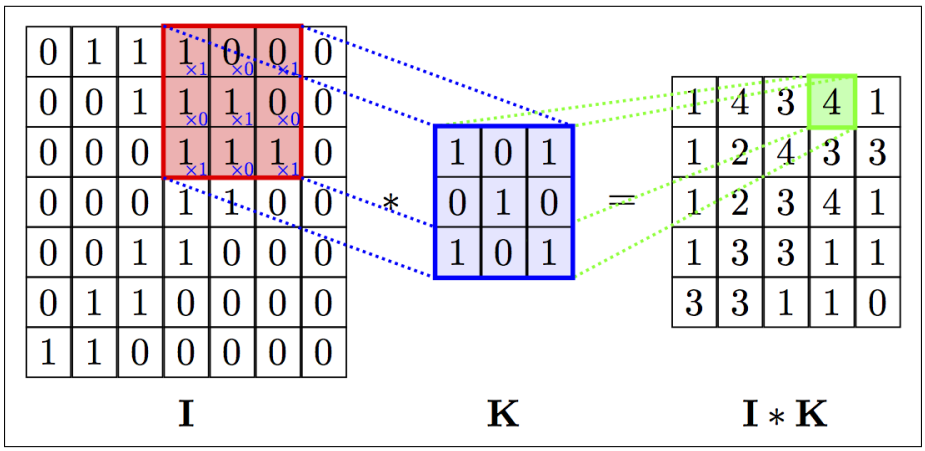


Figure 7: An example of CNN architecture — `Source`.

With this computation, you detect a particular feature from the input image and produce feature maps (convolved features) which emphasize the important features. Convolution of an image with different filters can perform operations such as edge detection, blur and sharpen by applying filters.

Stride

Stride is a component of convolutional neural networks, or neural networks tuned for the compression of images. For example, if a neural network's stride is set to 1, the filter will move one pixel, or unit, at a time. Stride is a parameter that works with padding, the feature that adds blank, or empty pixels to the frame of the image to allow for a minimized reduction of size in the output layer. Roughly, it is a way of increasing the

size of an image, to counteract the fact that stride reduces the size. Padding and stride are the foundational parameters of any convolutional neural network.

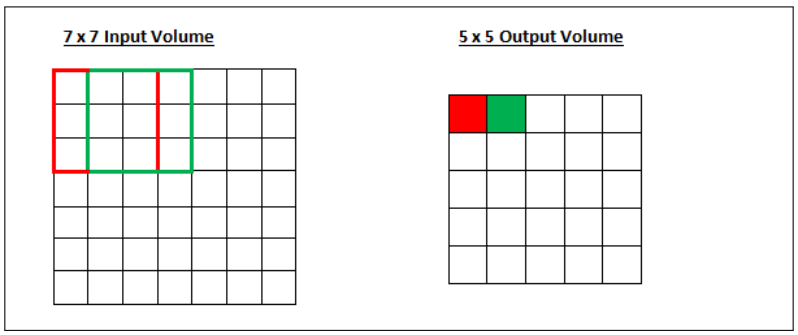


Figure 8: An example of stride with a 7x7 matrix — `Source`.

Padding

Sometimes filter does not fit perfectly the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits
- Drop the part of the image where the filter did not fit. But we are going to lose corner or side values, and this are important information.

Essentially, these convolution layers promote weight sharing to examine pixels in kernels and develop visual context to classify images. CNN's weights are attached to the neighboring pixels to extract features in every part of the image.

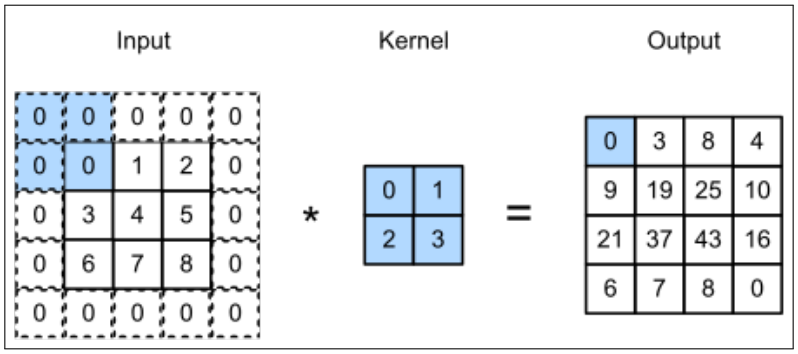


Figure 9: An example of padding and stride — `Source`.

Pooling Layer

The pooling operation involves sliding a two-dimensional filter over each channel of feature map and summarising the features lying within the region covered by the filter. Pooling layers are used to reduce the dimensions of the feature maps. Thus, it reduces the number of parameters to learn and the amount of computation performed in the network. The pooling layer summarises the features present in a region of the feature map generated by a convolution layer. So, we perform further operations on summarised features instead of precisely positioned features generated by the convolution layer. This makes the model more robust to variations in the position of the features in the input image.

- **Max Pooling:** Max pooling is a pooling operation that selects the maximum element from the region of the feature map covered by the filter.

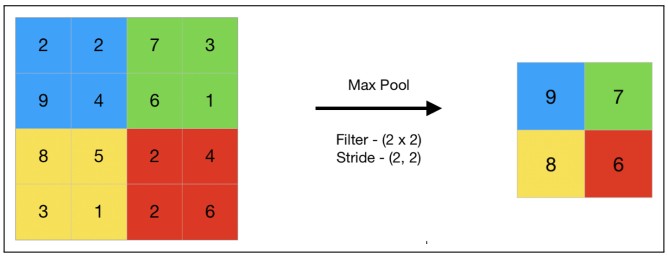


Figure 10: An example of max pooling layer — `Source`.

- **Average Pooling:** Average pooling computes the average of the elements present in the region of feature map covered by the filter.

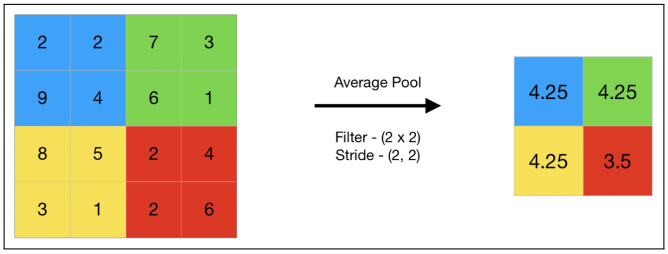


Figure 11: An example of average pooling layer — `Source`.

Activation Function (ReLU and Sigmoid)

ReLU's purpose is to introduce non-linearity in our ConvNet. The ReLU function mimics our neuron activations on a "big enough stimulus" to introduce nonlinearity for values $x > 0$ and returns 0 if it does not meet the condition. ReLu is faster to compute than the sigmoid function, and its derivative is faster to compute.

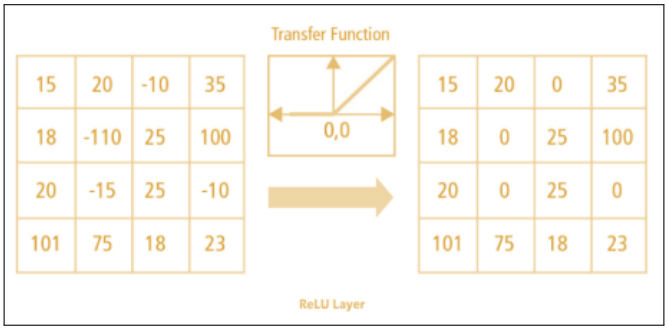


Figure 12: An example of ReLu layer — `Source`.

Fully Connected Layer

Fully Connected layers in a neural network are those layers where all the inputs from one layer are connected to every activation unit of the next layer. The input to the fully connected layer is the output from the final Pooling or Convolutional Layer, which is flattened and then fed into the fully connected layer. Flattened means that the output from the final Pooling and Convolutional Layer is a 3-dimensional matrix, we unroll all its values into a vector.

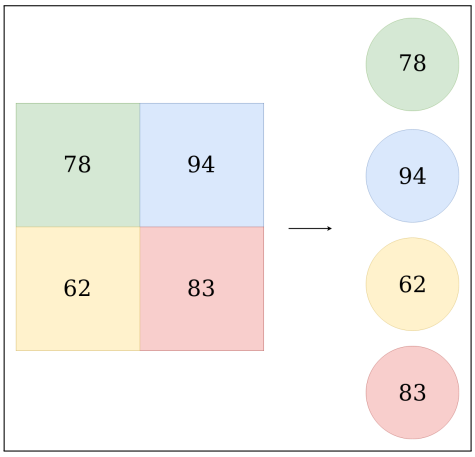


Figure 13: Flattening

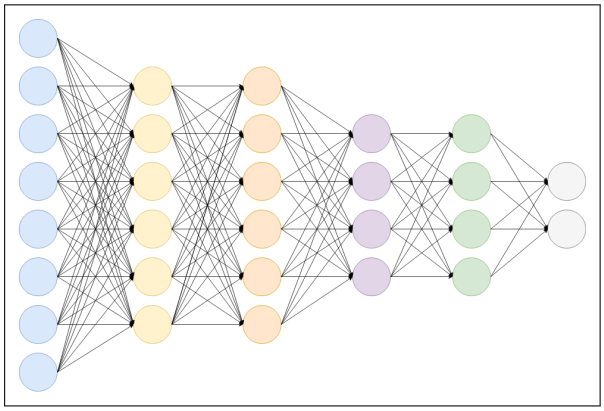


Figure 14: Fully Connected Network — `Source`.

VGG16 (Very Deep Convolutional Networks for Large-Scale Image Recognition)

Now that we understand how a CNN works, we can move on to something more powerful. It is called transfer learning, and I'm going to show one pre-trained model that has to fulfill this Image Classification task. "Transfer learning, used in machine learning, is the reuse of a pre-trained model on a fresh problem. In transfer learning, a machine exploits the knowledge gained from a previous task to improve generalization about another." [1]

The VGG-16 is one of the most popular pre-trained models for image classification. As we can see, the model is sequential and uses lots of filters. There are variations of the VGG16 model, which are basically improvements to it, like VGG19.

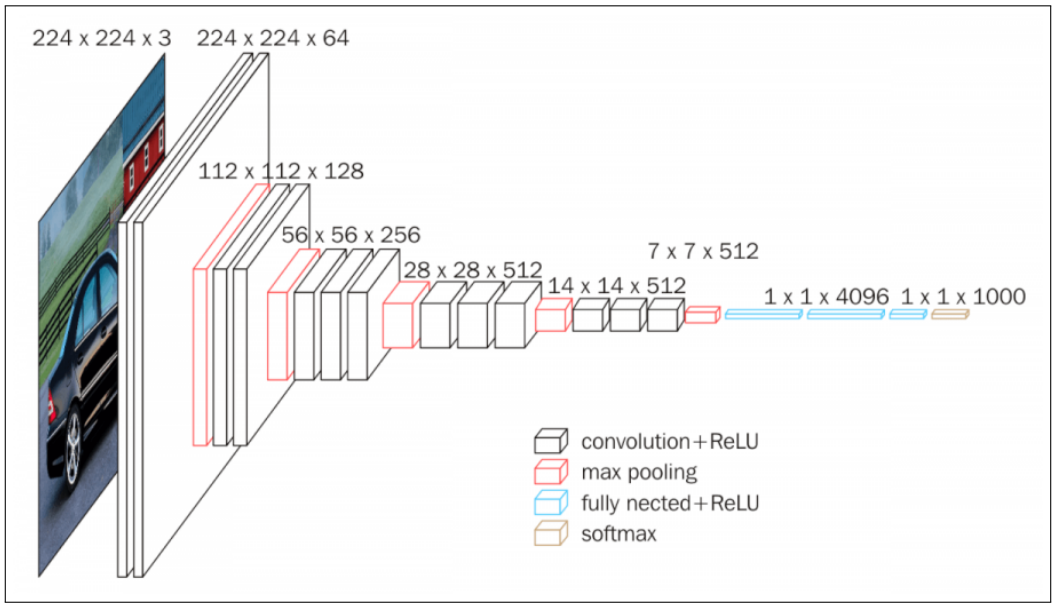
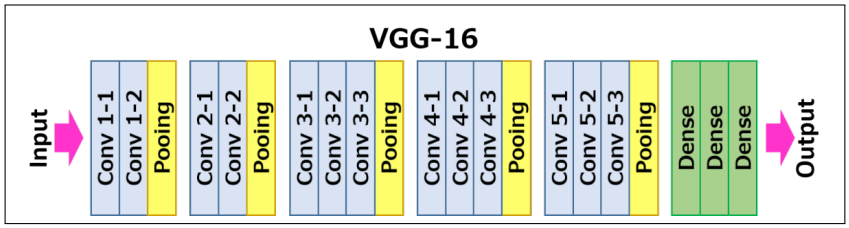


Figure 15: Architecture of VGG-16 — `Source`.

Here is a more intuitive layout of the VGG-16 Model.



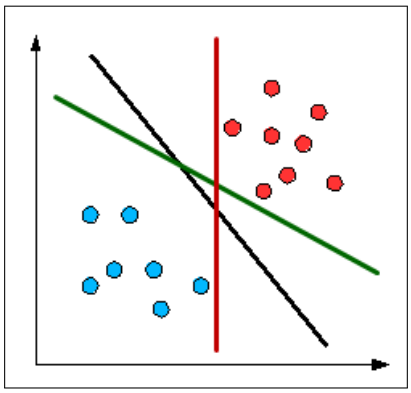
SVM (support-vector machines)

The Support Vector Machine aims to identify a hyperplane that best divides the support vectors into classes. To do this, it performs the following steps:

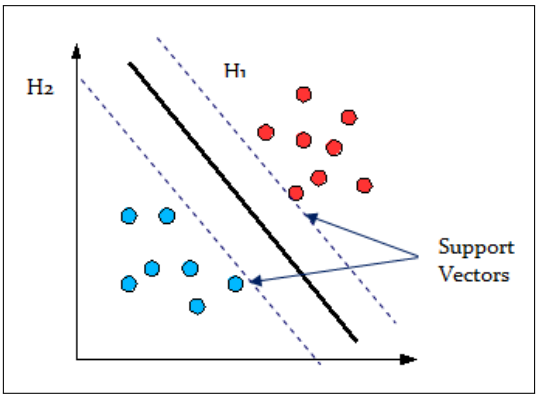
- Look for a linearly separable hyperplane or a decision limit that separates the values of one class from the other.
- If such a hyperplane does not exist, SVM uses a non-linear mapping to transform the training data into a higher dimension (if we are in two dimensions, it will evaluate the data in 3 dimensions). In this way, a hyperplane, which will be chosen for splitting the data, can always separate the data of two classes.

Linear datasets

In the following image, it is possible to find more than one linearly separable hyperplane. The problem is to find which of the infinite lines is optimal, the one that generates the least classification error on a new observation.



In fact, the further away from the hyperplane our data points are, the more confident we are that they have been classified correctly. Therefore, we want our data points to be as far as possible from the hyperplane, while remaining in the correct part. According to what has been stated, the black straight line is the line that maximizes the distance between the support vectors and the hyperplane itself.



Non linear datasets

When we can easily separate data with hyperplane by drawing a straight line is Linear SVM. When we cannot separate data with a straight line, we use Non–Linear SVM. In this, we have Kernel functions. They transform non-linear spaces into linear spaces. It transforms data into another dimension so that the data can be

classified. It transforms two variables x and y into three variables, along with z. Therefore, the data have plotted from 2-D space to 3-D space. Now we can easily classify the data by drawing the best hyperplane between them.

OBJECTIVES

This project classifies whether there is a dog or a cat in the image. I will create a CNN, use transfer learning methods and create an SVM to show the network architecture that gives the best result. To do this, I tried different architectures shown above and several known techniques to improve training.

METHODOLOGY

Dataset

The dataset that I have used, as said in beginning, is taken from Kaggle, a dataset with 12500 pictures of dogs and 12500 pictures of cats. Each class is divided into a folder, dog folder which has images of dogs and cat folder which has images of cats. The images have different size and not always the pet is well visible at first sight. I used the folder name as labels, because the images file name are only numbered, with no other specification. Two images, one of a dog and one of a cat have to be removed because corrupted when trying to resize them.

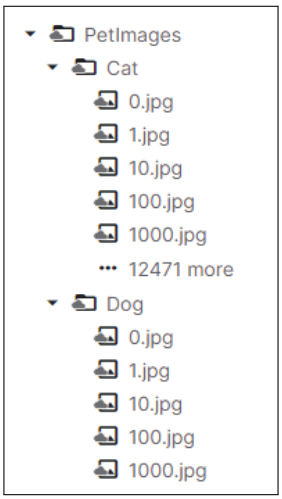


Image Augmentation

In this project, I have used Image Augmentation; I had to build a generator with this functionality. Data augmentation can help reduce the manual intervention required to develop meaningful information and insight of data, as well as significantly enhance data quality. Image augmentation parameters that are used to increase the data sample count are zoom, shear, rotation and so on. Usage of these parameters results in generation of images having these attributes during training of Deep Learning model. Image samples generated using image augmentation on existing data samples increased by the rate of nearly 3x to 4x times.

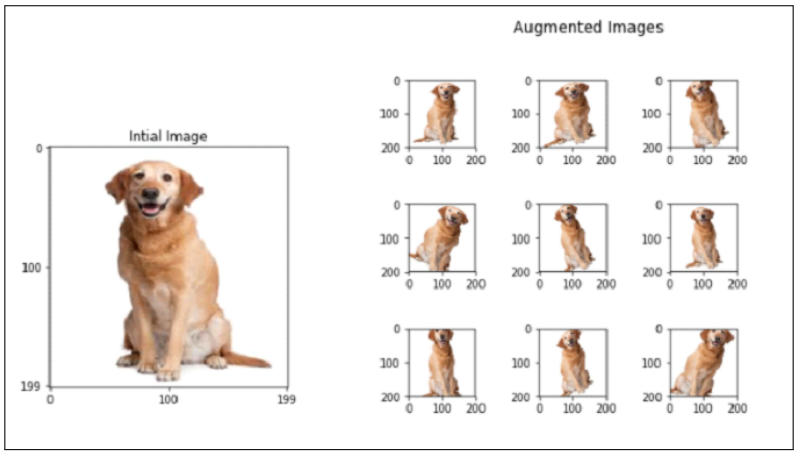


Figure 16: Data Augmentation — `Source`.

Tools

Computational

I built all these programs using Google Colab, so everyone could start the program and change it as he likes. So in this way there is no need to download or inspect the code from third part programs, someone could just look at it and test it on spot. This is the most cool thing of Google Colab, even if you are a starter you just need to press play to see some results and study the code already completed.

Library

I used the standard libraries that are usually imported when talking of machine learning (matplotlib, numpy, random ...). Other than that I have used:

- **PIL:** For resizing images.
- **Keras**
- **Kaggle**
- **Sklearn**
- **Skimage**
- **Cv2**

Methods

Image resize for feed the CNNs

Just after have downloaded the images in Google Colab we have to prepare them to be fed in the Neural Network. For doing so we have to resize all the images in the same dimension, because the dataset has images of different size. So I built a resize function that takes in input all the images of the path chosen, resize them, and replace them in the original dataset, I show it how under.

```
from PIL import Image

def resize():
    for item in dirs:
        if os.path.isfile(path+item):
            im = Image.open(path+item).convert('RGB')
            f, e = os.path.splitext(path+item)
            imResize = im.resize((img_height, img_width), Image.ANTIALIAS)
            imResize.save(f + '.jpg', 'JPEG', quality=100)
            print(path+item)

path = "PetImages/Cat/"
dirs = os.listdir( path )

resize()

path = "PetImages/Dog/"
dirs = os.listdir( path )

resize()
```

Figure 17: Resize function

Architectures

To do this image classification I have used a CNN produced by me, VGG16 and SVM. All these models are already described in the beginning of this document, and all the implementations are visible in every Google Colab link. I have tested them multiple times to see the best result that I could get; I saved all the models with the best settings, so anyone could try for himself and change things for his own.

Augmented data

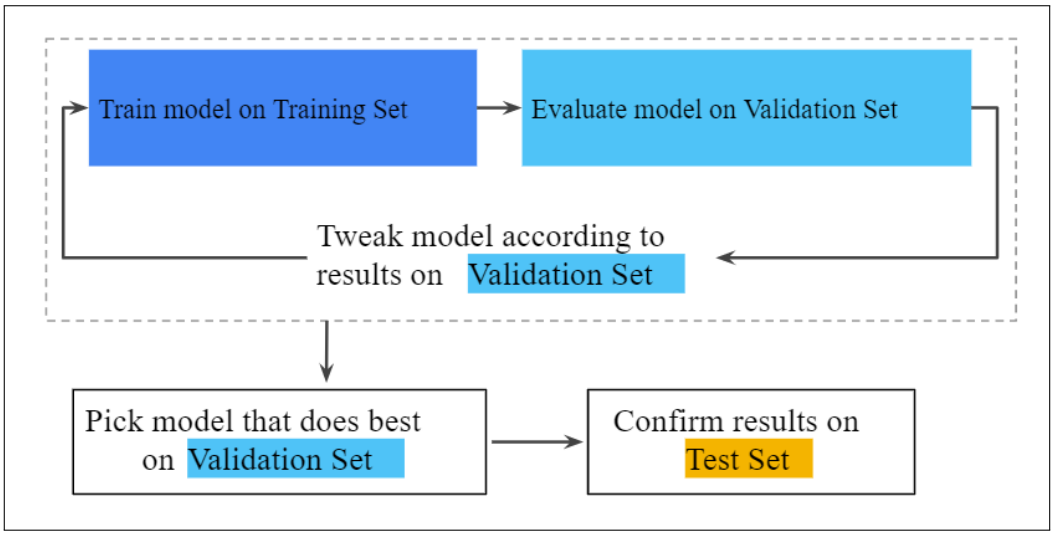
For the image augmentation, during the training, I used Keras library which has a method called ImageDataGenerator. In this method you can set all the change that an image could do, and then recall it so during the training it does the augmentation.

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

train_augmentation = ImageDataGenerator(
                     rescale=1./255,
                     rotation_range=45,
                     width_shift_range=.15,
                     height_shift_range=.15,
                     #brightness_range = (0,50),
                     shear_range = 0.2,
                     horizontal_flip=True,
                     #vertical_flip=True,
                     validation_split=0.2
```

EXPERIMENTS & RESULTS

Let's talk about some results, so we can see the differences between all the machines. So, how did I evaluate these models? We train our models on the training set and we evaluate our models on the validation set and tune the parameters of the models according to the validation loss and accuracy and we repeat this process until we get the model that best fit on the validation set. And after choosing the best model we test or confirm our results on testing set to get the correct accuracy or how well our model is generalised.



The test set were I tested my models is always taken from Kaggle, is a dataset of 2000 images split equally between cats and dogs. In this way we should have a dataset of images which, our model, has never seen before. At the beginning I reached an accuracy of 83% with Adam optimizer, 74% with SGD, and 50% with RMSprop,and I was not happy at all. I upgraded the model, and I got a way better results.

CNN with Adam optimizer

The first model that I have tested is the CNN I built. As the first test, I choose Adam because is a popular algorithm in deep learning and it achieves excellent results fast. "The authors describe Adam as combining the advantages of two other extensions of stochastic gradient descent. Specifically: AdaGrad - that maintains a per-parameter learning rate that improves performance on problems with sparse gradients. RMSProp - that also maintains per-parameter learning rates that are adapted based on the average of recent magnitudes of the gradients for the weight". [2]

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 112, 112, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 56, 56, 32)        9248
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 28, 28, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 14, 14, 128)       73856
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 7, 7, 128)         0
_________________________________________________________________
flatten (Flatten)            (None, 6272)              0
_________________________________________________________________
dense (Dense)                (None, 128)               802944
_________________________________________________________________
dense_1 (Dense)              (None, 64)                8256
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 65
=================================================================
Total params: 917,953
Trainable params: 917,953
Non-trainable params: 0
_________________________________________________________________
```

Figure 18: Summary of my CNN model, where is possible to see all the layers.

With this model I reached an accuracy of 93.4%, starting from 60%.

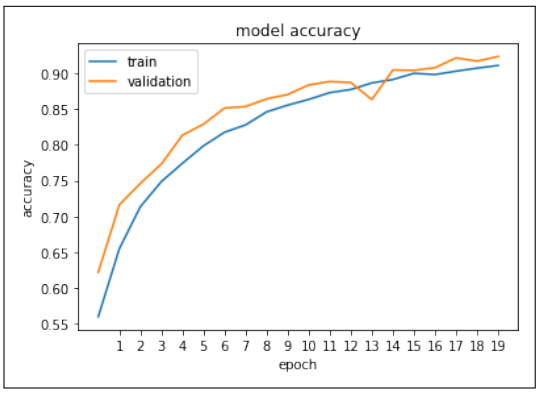


Figure 19: Accuracy of the model during the 20 epochs

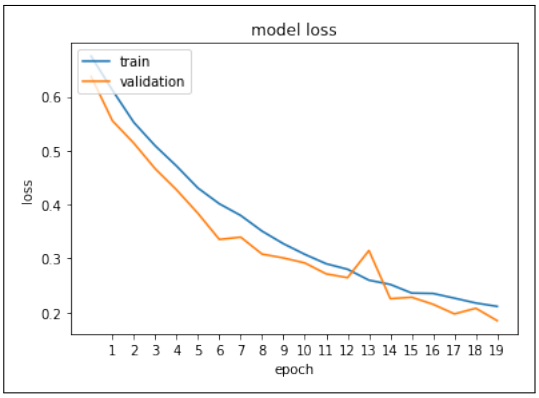


Figure 20: Loss of the model during the 20 epochs

In the end I produced a confusion matrix based on the test set, I uploaded the test set as a zip file in the GitHub. But it is possible to try any dataset since it contains photos of cats and dogs. Just remember to resize the images as I did in the code. Here the result of the confusion matrix,

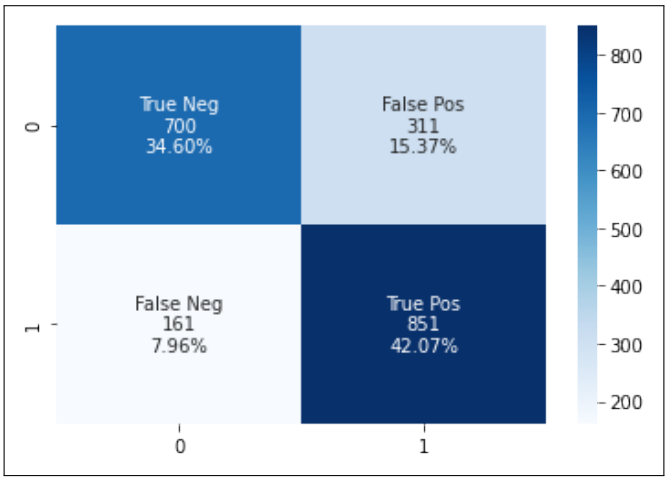


Figure 21: The model got right 76.67% of the images, in a dataset that has never seen before.

CNN with SGD optimizer

To test, I tested the same model with the SGD optimizer, and the results were not so great while training. In fact, we reached an accuracy of 75.6% while training, but when it was tested in the unseen dataset, the result is not much farther from the Adam optimizer.

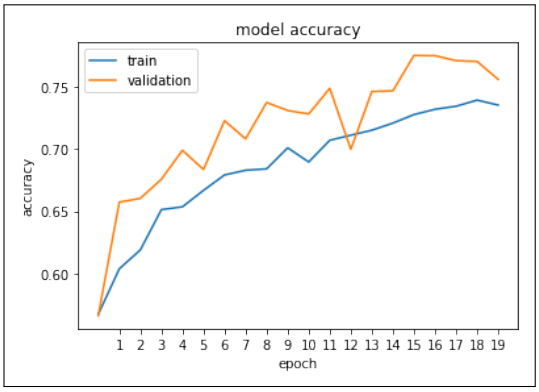


Figure 22: Accuracy of the model during the 20 epochs

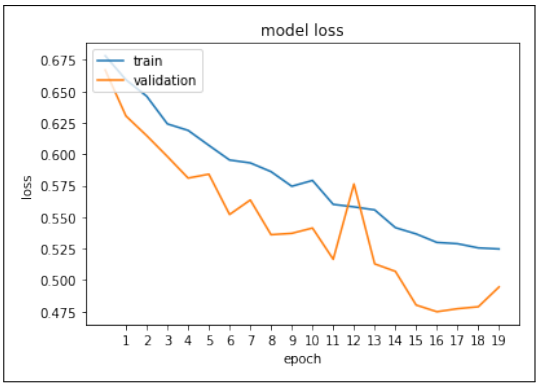


Figure 23: Loss of the model during the 20 epochs

"SGD is a variant of gradient descent. Instead of performing computations on the whole dataset SGD only computes on a small subset or random selection of data examples. SGD produces the same performance as regular gradient descent when the learning rate is low." [3]

But now lets see how this model performed in the unseen test set. The test set is the same as the one used for the Adam optimizer.

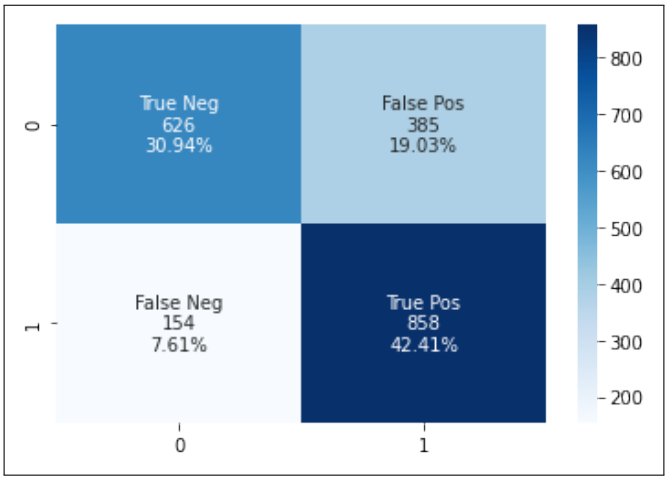


Figure 24: The model got right 73.35% of the images, in a dataset that has never seen before.

CNN with RMSprop optimizer

For the last test on my CNN I choose the RMSprop optimizer, and its performance is not bad at all. The first time I trained with this optimizer I got the worst result ever, like it didn't even train. After I did some changes to the model and added some layers the story changed, I got an accuracy of 87% while training.

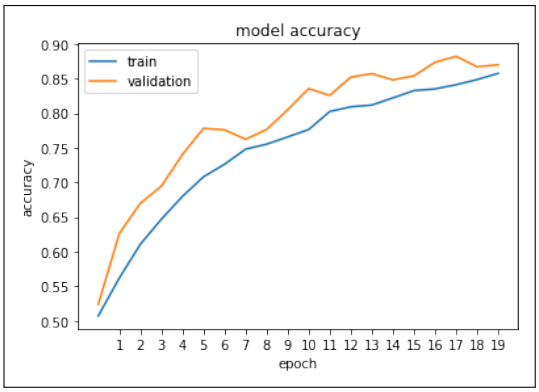


Figure 25: Accuracy of the model during the 20 epochs

Figure 26: Loss of the model during the 20 epochs

"The RMSprop optimizer is similar to the gradient descent algorithm with momentum. The RMSprop optimizer restricts the oscillations in the vertical direction. Therefore, we can increase our learning rate and our algorithm could take larger steps in the horizontal direction converging faster. The difference between RMSprop and gradient descent is on how the gradients are calculated". [4]

In conclusion, to see which of the three optimizers worked better, we have to see the last confusion matrix.

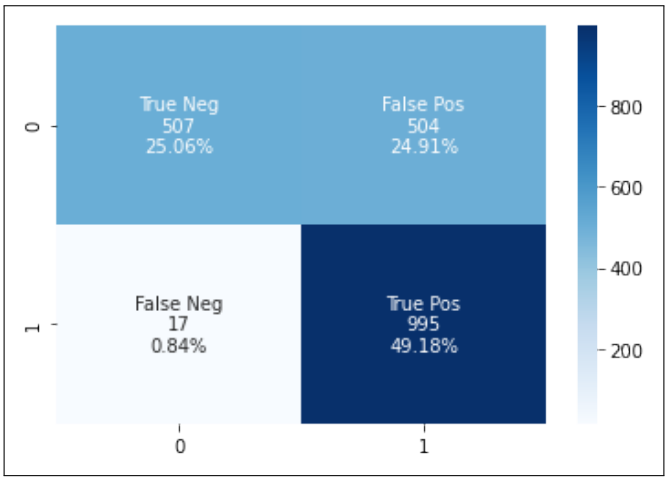


Figure 27: The model got right 74.24% of the images, in a dataset that has never seen before.

In the end, this model is not the most powerful, but can do a decent classification. Under we will see other two architecture and how they acted in the same circumstances. Below, I summarize the result I got from these tests:

	val accuracy	test accuracy	precision	recall
Adam	93.4%	76.7%	0.73	0.84
SGD	75.6%	73.4%	0.69	0.85
RMSprop	87%	74.2%	0.66	0.98

VGG16 transfer learning CNN

Finally, here we are, in the promising section of transfer learning. For this kind of classification, and even for multiclass classification, transfer learning is always the right choice. I chose VGG16, which is powerful and already pre-trained in this camp. I've already explained above how is it built the VGG16, so now we are going to talk about results, and they are mind blowing.

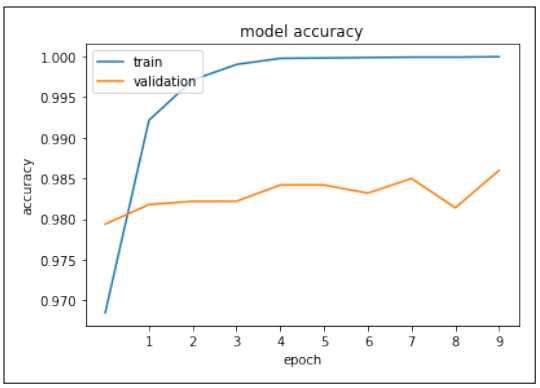


Figure 28: Accuracy of the model during the 10 epochs

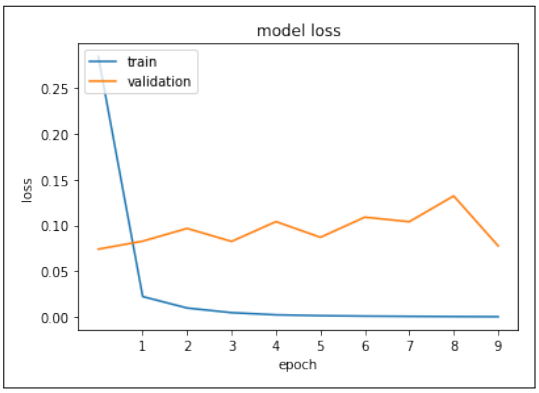


Figure 29: Loss of the model during the 10 epochs

As we can see, this CNN for our problem is exaggerated, already in the beginning it was at 98% accuracy. This CNN is used for more complex problem, as you have seen above how many layers it has. But to test, and showing how it works and his power, I used it anyway, so people could see how useful is the transfer learning. Moving on, here it is the confusion matrix:

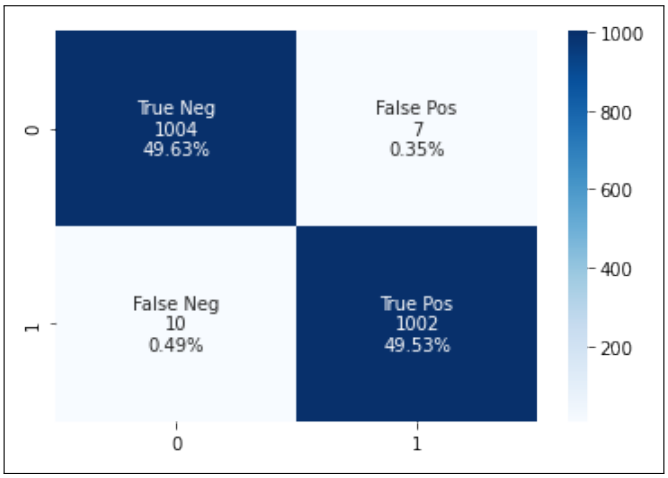
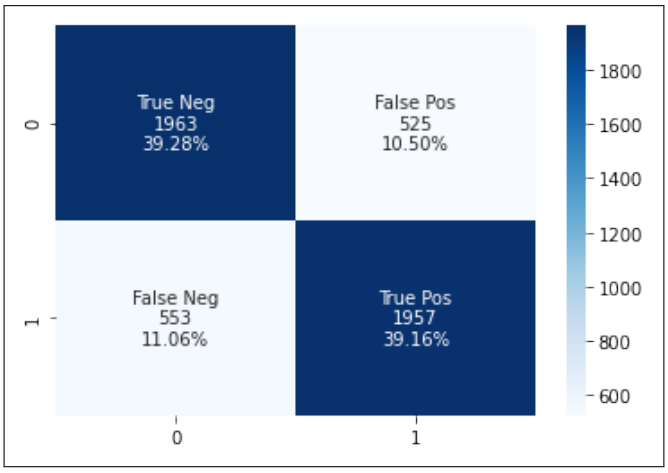


Figure 30: The model got right 99.16% of the images, in a dataset that has never seen before.

The results are outstanding, but predictable. At the end of the epochs it got a validation accuracy of 98.34%, and, as shown in the image above, a test accuracy of 99.16%! The precision and recall are both 0.99.

Support-vector machine

Last but not least, we have our SVM, which applies to our images with HOG features. In the beginning I didn't expect it to act well, but it got decent results. This kind of machine works better when we have to detect if there is something or there isn't in our picture. But I wanted to try it, and here are the results, we got an accuracy of 78.4%, which is not great but neither terrible. I trained the model on 20000 images; I have used the rest 5000 for testing the prediction. Here is the confusion matrix I got from this SVM.

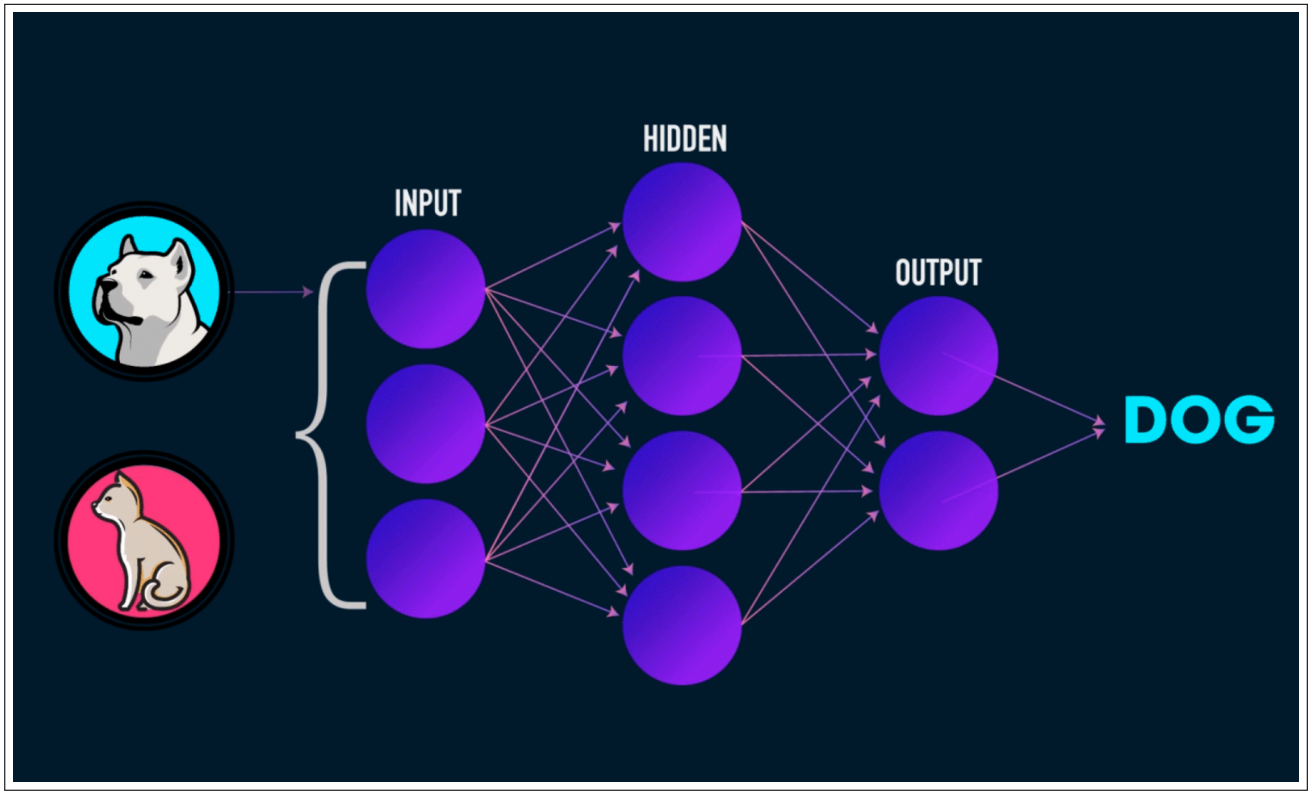


We can see that we got a precision of 0.79 and a recall of 0.78.

CONCLUSIONS

In conclusion, I've covered every common part of CNN and SVM. It should now be clear how these neural networks work and how SVM behaves. My CNN results aren't the best, but it's always possible to add more layers and make the network even deeper. The SVM worked fine, but I imagine it will perform even better in a detection problem instead of classification. The VGG-16 model, as already said before, is much powerful. It is possible to use other Neural Networks with transfer learning, but with this kind of problem we would have to get nearly the same results in every test. In base of the problem someone could pick a Neural Network instead of another, and train or adapt the last layers for his scope. I'll leave here my links to the Google Colab code, so anyone could access and look at them.

- `CNN with 3 different optimizer.`
- `VGG-16.`
- `SVM.`



Bibliography

[1] Niklas Donges. *WHAT IS TRANSFER LEARNING? EXPLORING THE POPULAR DEEP LEARNING APPROACH*. URL: https://builtin.com/data-science/transfer-learning.

[2] Jason Brownlee. *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning*. URL: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/.

[3] Synced Tony Peng. *Fast as Adam Good as SGD*. URL: https://medium.com/syncedreview/iclr-2019-fast-as-adam-good-as-sgd-new-optimizer-has-both-78e37e8f9a34.

[4] Rohith Gandhi. *A Look at Gradient Descent and RMSprop Optimizers*. URL: https://towardsdatascience.com/a-look-at-gradient-descent-and-rmsprop-optimizers-f77d483ef08b.