Comparing two supervised machine learning algorithms

This project will present a description of ***Convolutional Neural Network*** *and* ***Naive Bayes*** *algorithms implemented in python for image classification and comparison of their performance*

Introduction

Image classification is the process of taking an **input** (like a picture) and outputting a **class** (like “cat”) or a **probability** that the input is a particular class (“there’s a 90% probability that this input is a cat”). But how can a computer learn to do that?

There are many algorithms in order to achieve this task, but in this report I will present the CNN – which is the state of art when it comes to image classification – and I will also present the Naive Bayes approach – which does not perform that good on image classification, but on any other probabilistic prediction with strong independence between variables. Naive Bayes is algorithm is simply applying Bayes’s theorem of probabilities.

Description of data set

The data set used for this task is a well known base of 25000 images in 3 channels with dogs and cats from Kaggle. The dataset has already been labeled and almost prepared for feeding in a CNN. The processing I have done on data was to iterate with a for loop over each item in order to reduce them to gray scale and from their original size to a stretched resolution of 50x50.

After all the processing on data I saved everything in a numpy file which contains an array with all the data images and labels in it for future usage.

Convolutional Neural Network

The first approach I choose to talk about is Convolutional Neural Network (CNN). I used Keras library for creating this image classifier model. Keras is a powerful python library that is built on top of tensorflow, which is a deep learning library. Keras purpose is to make easier the process of handling neural networks with tensorflow.

Convolutional Neural Network is just a class of deep neural networks that is used for analyzing visual imagery and computer vision. Its arhitecture consists of multiple layers. A convolutional layer is constructed by many neurons. Because computers sees images as pixels expressed as matrix with 3 dimensions (width X height X channels no.). Each neuron has assigned to it value from this matrix. For my project I have processed images to have only one gray channel due to my computational power.

The convolutional layer makes use of filters. The filter is used to detect presence of different patterns within the image matrix. The filter dimension is smaller than the image’s, but with the same depth (no. of channels). The filter it convolved (or slided) across the width and height of the image matrix, and a dot product is computed to give an activation map (another matrix). Different filters which detect different features are convolved on the input file and a set of activation maps is outputted which is passed to the next layer in the CNN.

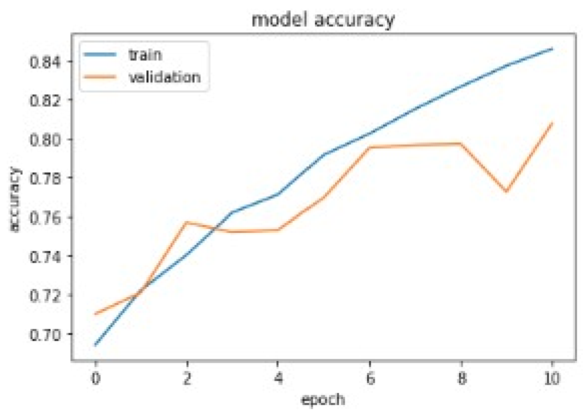
The first layer has 2500 neurons, because my input image has a 50x50 resolution with a depth of 1 as I said before. The second layer has mentioned that it needs 32 filters fo 3x3, a default padding and stride. The output if this will be some activation maps. Those feature maps are passed through an activation layer called ReLU. I have used ReLU because it is the most used activation function in this type of tasks and it helps to pass negative inputs so it will fire only a part of the neurons on the next layer.

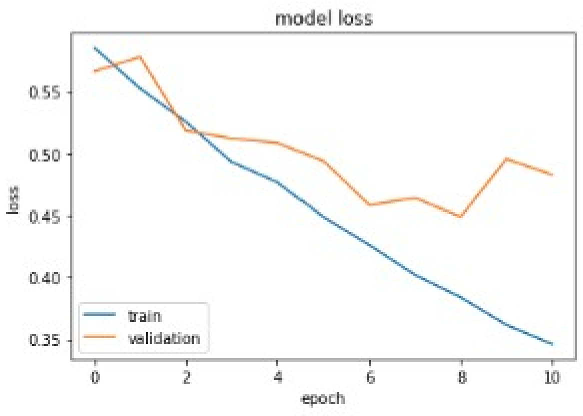
Between convolutional layers, the pooling layer can be seen. They basically are used for reducing the number of useless parameters and computation in the network. I have used MaxPooling2D with a dimension of 2x2. It makes use of filters that slide through the input and it extracts only the maximum parameter in that filter.

The arhitecture consists of 4 convolutional layers and one dense layer for output. To prepare the model for output I first fallten the feature map to 1 dimension. The dense function uses softmax activation function for normalizing the outputs between 0 and 1.

My model is also defined by the optimizer and loss function. The optimizer in case in 'Adam' with a learning rate of 0.001 and the categorical cross entropy loss function. Loss function measure the distance between the computed probability distribution of classes and the distribution of the output we expect.

After I have trained the model for 11 epochs with a train data of 23000 images and a validation data with 2000 images I got the following results:





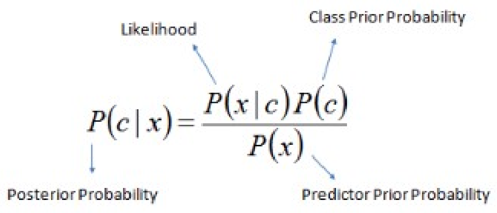
Time to train	593 seconds
Real Accuracy	0.8075
Real Loss	0.4828

After many attempts with different arhitectural design, different optimizers and by tunning all the parameters in the neural network I got the above results which I think they are pretty good for classifing not very tricky pictures with cats or dogs.

The model can be improved by having more images or more patience for training the model as many epochs as it needs. Also I would introduce a function to stop the training when these 2 parameters shown above start going downwards, also the function I would introduce would also change the learning rate dynamically in order to get a much better result.

Naive Bayes

This approach is based on Bayes’ theorem with the '*naive*' assumption of conditional independence between every pair of features given the value of the class variable.


Likelihood
Class Prior Probability
$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$
Posterior Probability
Predictor Prior Probability

Let X be a feature vector containing all the 2500 pixels of each image - $X=\{x_1,x_2,\ldots,x_{2500}\}$ and y_k be the class variable, so the predicted class will have the following formula:

$$y_{\text{predicted}} = argmax_{y_k} P(y_k) \prod_{i=1}^{i=n} P(x_i|y_k)$$

The likelihood of the features is assumed to be Gaussian:

$$P(x_i|y_k) = \frac{1}{\sqrt{2\pi\sigma_{y_k}^2}} exp(-\frac{x_i - \mu_{y_k}}{2\sigma_{y_k}^2})$$

This algorithm has been fed with the same data as the CNN was with the same gray scale at 50x50 resolution and got the following performance:

	precision	recall	f1-score	support
0	0.57	0.52	0.55	2481
1	0.57	0.62	0.59	2519
accuracy			0.57	5000
macro avg	0.57	0.57	0.57	5000
weighted avg	0.57	0.57	0.57	5000

Comparison table

The following table compare the performing results of the two approaches on 50x50 resolution having 1 channel depth

	Naive Bayes	CNN
Time (seconds)	insignificant	593
Accuracy	0.569	0.808 (real accuracy)
		0.846
Loss	14.859	0.4828 (real loss)
		0.3460

Conclusion

By far this implementation of Naive Bayes cannot compete with the implementation of Convolutional Neural Network on the specified set of images with cats and dogs. In order to improve Naive Bayes implementation I would use Local Binary Pattern (LBP) for feature extraction because it is one of the most widely used feature extraction methods for describing image textures including points, lines, and surfaces because of its texture representation capability and computational simplicity. I consider that Discrete cosine transform (DCT) as a frequency converter would bring a great improvement for image classification by decomposing the image signal into the underlying spatial frequencies.

I enjoyed implementing both of these models and tweaking their parameters around to get the “magic” prophet alive.