Building A Cat and Dog Image Classifier Using Convolutional Neural Network (CNN)

Evgeny Vendrov, Tal Noam, Amitai Zamir,

January, 2020,

Ariel University.

Abstract:

CNN is a very popular class of deep learning used mainly for analyzing of visual imagery, with emphasis on image classification.

In this paper, we'll describe our attempt to build an accurate CNN classifier for dog and cat images using a pretty small data set[1] (6K sized train set, 2K sized validation set and 2K sized test set), and a relatively weak computer system (laptops with basic GPU and only 8GB RAM), and compare its results to basic Logistic Regression and MLP models which we created on the same environment.

The CNN itself is composed of VGG blocks[2], using data augmentation[3] and Dropout as regularization methods.

The coding itself is done in python 3.7, using TF(2.0).Keras framework.

Our best results (on not previously seen images) are little above 92% (test - set accuracy) and 0.92 F-measure.

Introduction:

In most cases, trying to solve a problem using Deep Learning methods requires a lot of data to begin with, or at least this is the common believe, train sets are hopefully as big as possible[4], while size and variety of data used for training the model is believed to be crucial to the "strength" of the model[5], this leads most data scientist's to work with "super-computers" which are built to this purpose[6].

But, in our case, we work with laptops which are not strong enough (GPU and RAM wise) to process a really large data bundles at once, in addition – training the model (even on this not-so-big data set) is VERY slow (even while using data batches and multi-threaded processing), so our data set has to be quite small.

The point of our research is to show that building efficient CNN's is possible even with small data sets and relatively weak

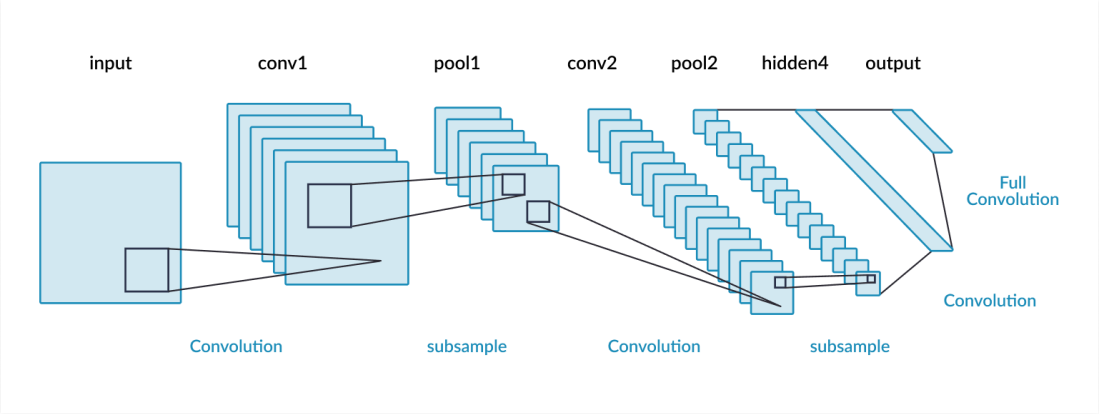
computers, in addition, we compare such CNN's with basic Logistic Regression and MLP.

Lucky for us, CNN models which composed of VGG blocks and use data augmentation, don't require this much of train data[7], so this architecture is perfect for our situation.

VGG blocks emerged for the first time from "Visual Geometry Group" at Oxford University[8], and are now very popular in CNN's, as they believed to be very efficient.

Classic VGG block is composed of a convolutional layer (with padding to maintain the resolution) with 3X3 sized kernel, nonlinear activation function (as the classic "ReLU" for example), and finally 2X2 max pooling with stride of 2 (for halving the resolution after each block)[8].

Typical CNN architecture using VGG blocks:



In our research we found that best solutions to cat and dog classification problems, using VGG blocks – are around 97% of accuracy[10], we haven't got "this good", and stopped around 92% because as already mentioned – time and computation limits.

Methodology:

We used trial and error strategy to improve our model, we tried many architectures based on the basic VGG architecture, combined with dropout and data augmentation, while comparing loss and accuracy graphs (as shown below in Results section), we studied when there's overfitting – by noticing when validation and train graphs are separating, and tried to maximize our train accuracy based on our observation.

We then compared between models by their accuracy percentage on the test set, and F-measure.

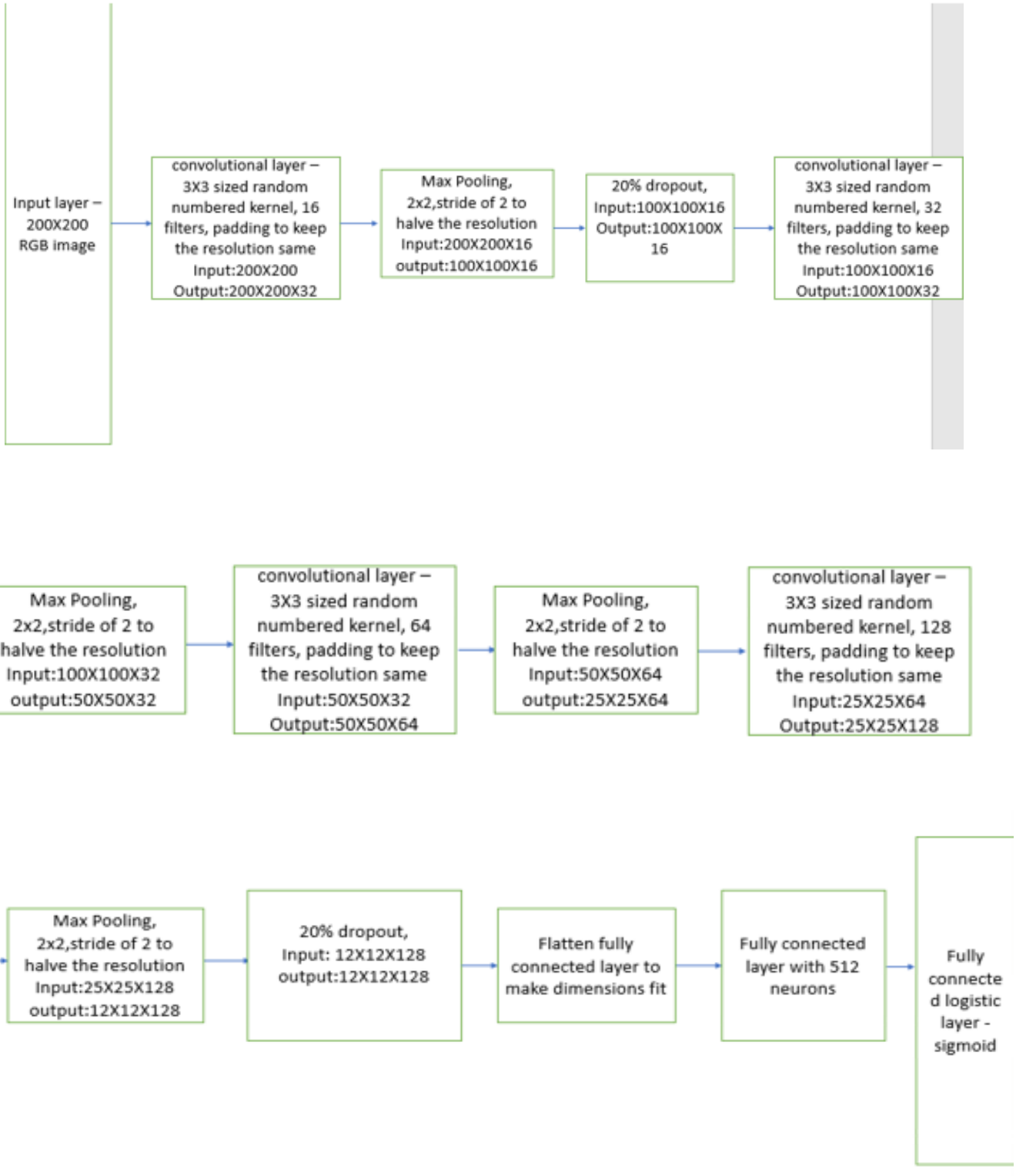
We found that its quite hard avoiding overfitting with CNN, while on previous models we barely even got to small fitting to train set, with this model we overfitted after as much as 10 epochs, even while using relatively intense dropout[11], while too much dropout made us stuck on same value of train loss, we discovered that adding a noise to the images in train set should prevent

overfitting and make our model "stronger" as it's trained on "harder to understand images"[12], which led us using data augmentation, which means – adding noise to the phots, example of regular dog image and "noised" one:

our model is trained on images like the one on the left but is tested on regular ones like the one on the right.

Finally, we chose the next architecture and ran it for 250 epochs:

*please notice that first convolution's output is 100X100X16 – not 32 as written



```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 200, 200, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 100, 100, 16)      0
_________________________________________________________________
dropout (Dropout)            (None, 100, 100, 16)      0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 100, 100, 32)      4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 50, 50, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 25, 25, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 25, 25, 128)       73856
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 128)       0
_________________________________________________________________
flatten (Flatten)            (None, 18432)             0
_________________________________________________________________
dense (Dense)                (None, 512)               9437696
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 9,535,649
Trainable params: 9,535,649
Non-trainable params: 0
_________________________________________________________________
```

Results:

* Our Previous works:

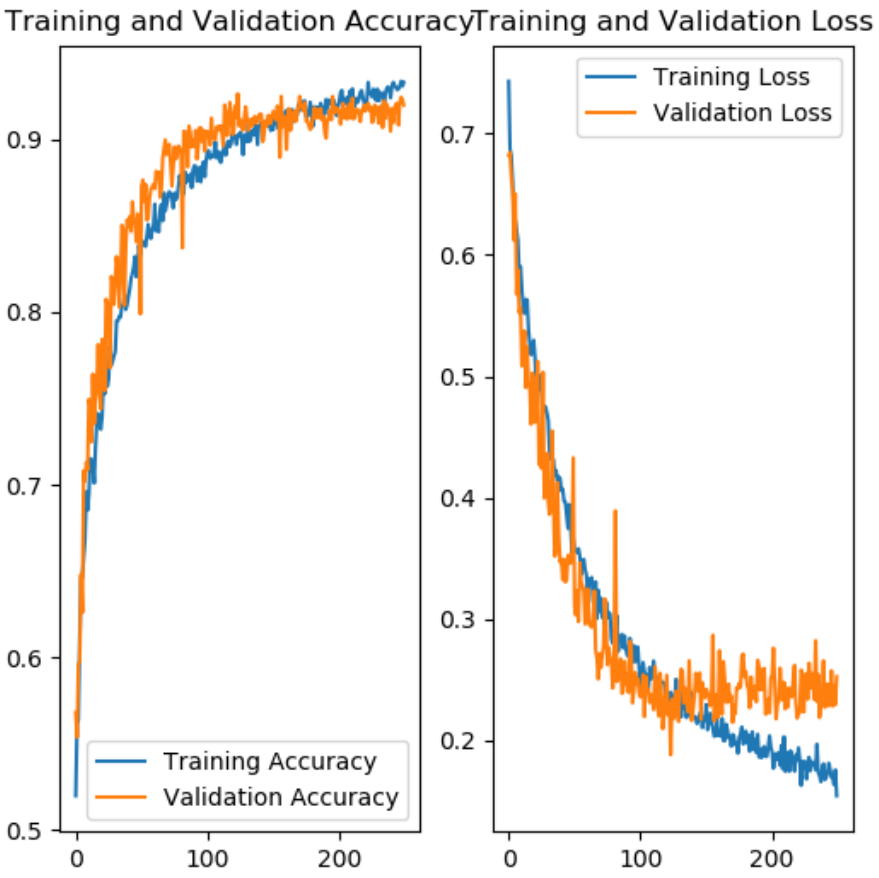
We previously tried to tackle the same problem (but with grayscale 20X20 images) using only input layer and Logistic Regression layer with ADAM optimizer – this haven't worked, after 100,000 epochs, which took over 10 hours, we got only 75.8%[12] of accuracy to train set – which simply means that the model is not "strong" enough.

With MLP, which architecture was: input layer, fully connected to 25 neurons, fully connected to 10 neurons, which is fully connected to logistic layers which includes logistic regression, while every layer has ReLU activation function and were using ADAM optimizer.

we got very strange results, after around 170K epochs – loss value got zeroed, but the fit to train was still not good enough[12], we checked it on a smaller train set (50 cat images and 50 dog images), got zero train loss

and 100% accuracy on train – so this isn't a bug, we guess that basic MLP is just not "smart" enough to handle this mission, it can't even fit itself to a big train set.

But CNN VGG block architecture with dropout and data augmentation did work:



After as much as 250 epochs (much less than 250K), we got to great fitting to train set and over 90% accuracy on validation set (92.14% accuracy for test, 0.92 F-measure).

Conclusion:

Easy to see that CNN is much better in classifying images than basic MLP or LR, in addition, it is a more suitable solution for the problem for weaker computers and smaller data sets, as its more efficient epoch and accuracy wise, while LP and MLP are not good enough for image classification – CNN's are GREAT!

Please notice, that our model isn't the best we could output, as we encountered time and computation problems, we could train the model for more time – to zero the train loss which should make our test accuracy even better, or, we could make our model more complex (maybe VGG16 architecture [14]), which could make our fitting to train set faster.

References:

[1] data set used: https://www.kaggle.com/tongpython/cat-and-dog/data

[2] https://www.robots.ox.ac.uk/~vgg/practicals/cnn-reg/

[3] https://towardsdatascience.com/data-augmentation-for-deep-learning-4fe21d1a4eb9

[4] https://towardsdatascience.com/does-deep-learning-really-require-big-data-no-13890b014ded

[5] https://machinelearningmastery.com/impact-of-dataset-size-on-deep-learning-model-skill-and-performance-estimates/

[6] https://towardsdatascience.com/supercomputing-the-heart-of-deep-learning-and-artificial-intelligence-49218c6bdee5

[7] https://bair.berkeley.edu/blog/2019/06/07/data_aug/

[8] https://d2l.ai/chapter_convolutional-modern/vgg.html

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

[11] https://github.com/EvgenyVendrov/DeepLearningCourse/tree/master/CNN/saved_models

[12]https://github.com/EvgenyVendrov/DeepLearningCourse/blob/master/MLP%2BLogistic%20Regression/Description.docx

[13] https://bair.berkeley.edu/blog/2019/06/07/data_aug/

[14] https://neurohive.io/en/popular-networks/vgg16/