FEDERAL STATE AUTONOMOUS EDUCATIONAL INSTITUTION OF HIGHER EDUCATION
ST. PETERSBURG NATIONAL RESEARCH UNIVERSITY OF INFORMATION TECHNOLOGIES, MECHANICS AND OPTICS

Faculty of Information Technologies and Programming
Program track 01.04.02 Applied Mathematics and Informatics
Machine Learning and Data Analysis

REPORT
On the Course Project for Image Analysis

Topic:
Classify Images of dogs and cats using
Deep Convolutional Neural Networks

Mohamed Saber
M42332

Saint Petersburg
2020

Introduction

There are many attempts to build a model that is able to classify cats and dogs with a high accuracy rate. The ratio of .82 with a model can be reached classifier is a combination of support vector machine classifiers trained on color and texture features extracted from images [1] and it found another try with exploits Interest aligned manual image categorization [2] but OverFat model presents an integrated framework for using Convolutional Networks for classification, localization, and detection. it's called a novel deep learning approach to localization by learning to predict object boundaries. Bounding boxes are then accumulated rather than suppressed in order to increase detection confidence. We show that different tasks can be learned simultaneously using a single shared network. [3]

But the model is in this paper using DCNN to classify images 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. While the dataset is effectively solved, it can be used as the basis for learning and practicing how to develop, evaluate, and use convolutional deep learning neural networks for image classification. this includes how to develop a robust test harness for estimating the performance of the model, explore improvements to the model, and to save the model and later load it to make predictions on new data.

Dataset

The dogs vs cats dataset refers to a dataset (Figure 1) used for a Kaggle machine learning competition. The dataset is comprised of photos of dogs and cats provided as a subset of photos from a much larger dataset of 3 million manually annotated photos. The 2,000 images used in this exercise are excerpted from the "Dogs vs. Cats" dataset available on Kaggle, which contains 25,000 images. Here, we use a subset of the full dataset to decrease training time for educational purposes.

Figure 1 Sample Dataset

Approach

The images that will go into our convnet are 200 x 200 color images (in the next section on Data Preprocessing, I'll add handling to resize all the images to 200 x 200 before feeding them into the neural network).

Let's code up the architecture. We will stack 3 {convolution + relu + maxpooling} modules. Our convolutions operate on 3x3 windows and our maxpooling layers operate on 2x2 windows. Our first convolution extracts 32 filters, the following one extracts 64 filters, ,the following one extracts 128 filters and the last one extracts 256 filters.

From reviewing the learning curves for the model during training, the model showed strong signs of overfitting. So I will using two approaches to attempt to address this overfitting: dropout regularization and data augmentation.

I build a classifier model from scratch that is able to distinguish dogs from cats. I will follow these steps:

1. Explore the example data
2. Building ConvNet Models:
3. Plot Evaluating Accuracy and Loss for the My Model
4. Make Prediction

Experiments

Model A:

It's one block cnn has single convolutional layer with 32 filters followed by a max-pooling layer. so I created a function name ModelA() that will define a model and return it ready to be fit on the dataset. This function can then be customized to define different baseline models, e.g. versions of the model with 1 VGG style blocks. The model will be fit with stochastic gradient descent and start with a conservative learning rate of 0.001 and a momentum of 0.9. it appears that the accuracy and validation accuracy is low and estimated at .050 together (Figure 2)

```
Model: "sequential_16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_34 (Conv2D)           (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d_34 (MaxPooling (None, 100, 100, 32)      0
_________________________________________________________________
flatten_15 (Flatten)         (None, 320000)            0
_________________________________________________________________
dense_29 (Dense)             (None, 128)               40960128
_________________________________________________________________
dense_30 (Dense)             (None, 1)                 129
=================================================================
Total params: 40,961,153
Trainable params: 40,961,153
Non-trainable params: 0
```

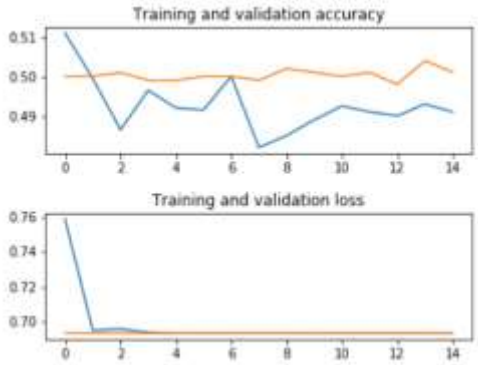

Figure 2 Architecture and Accuracy of Model A

Model B:

it's two block cnn model extends the one block model and adds a second block with 64 filters. so define function name ModelB() that will define a model and return it ready to be fit on the dataset. This function can then be customized to define different baseline models, e.g. versions of the model with 2 VGG style blocks, it appears that the accuracy is .98 and validation accuracy is .66 (Figure 2)

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 100, 100, 64)      18496
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 50, 50, 64)        0
_________________________________________________________________
flatten_8 (Flatten)          (None, 160000)            0
_________________________________________________________________
dense_15 (Dense)             (None, 128)               20480128
_________________________________________________________________
dense_16 (Dense)             (None, 1)                 129
=================================================================
Total params: 20,499,649
Trainable params: 20,499,649
Non-trainable params: 0
```

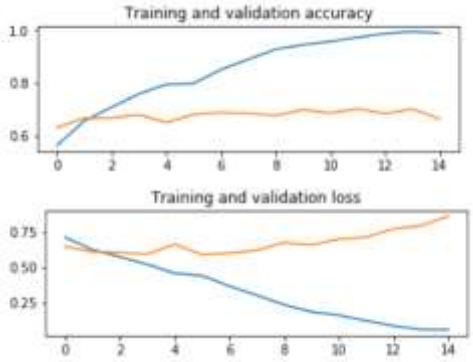

Figure 3 Architecture and Accuracy of Model B

Model C:

It’s three block cnn model extends the two block model and adds a third block with 128 filters. so I define function name ModelC() that will define a model and return it ready to be fit on the dataset. This function can then be customized to define different baseline models, e.g. versions of the model with 3 VGG style blocks, it appears that the accuracy is .96 and validation accuracy is .69 (Figure 3)

```
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 100, 100, 64)      18496
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 50, 50, 128)       73856
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 25, 25, 128)       0
_________________________________________________________________
flatten_10 (Flatten)         (None, 80000)             0
_________________________________________________________________
dense_19 (Dense)             (None, 128)               10240128
_________________________________________________________________
dense_20 (Dense)             (None, 1)                 129
=================================================================
Total params: 10,333,505
Trainable params: 10,333,505
Non-trainable params: 0
```

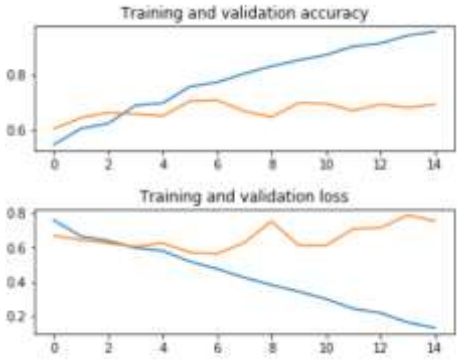

Figure 4 Architecture and Accuracy of Model C

Model D:

It’s had four block cnn model with extends the three block model and adds a fourth block with 256 filters. e.g. versions of the model with 4 VGG style blocks. it appears that the accuracy is .93 and validation accuracy is .75 (Figure 4)

```
Model: "sequential_12"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_23 (Conv2D)           (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 100, 100, 64)      18496
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 50, 50, 128)       73856
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 25, 25, 128)       0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 25, 25, 256)       295168
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 12, 12, 256)       0
_________________________________________________________________
flatten_12 (Flatten)         (None, 36864)             0
_________________________________________________________________
dense_23 (Dense)             (None, 128)               4718720
_________________________________________________________________
dense_24 (Dense)             (None, 1)                 129
=================================================================
Total params: 5,107,265
Trainable params: 5,107,265
Non-trainable params: 0
```

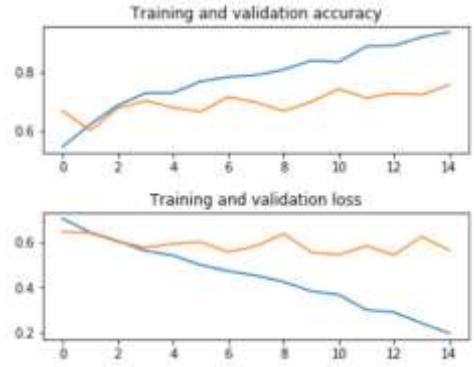

Figure 5 Architecture and Accuracy of Model D

Model E:

It’s three block cnn model extends the two block model and adds a third block with 128 filters. so I define function name ModelE() that will define a model and applying dropout regularization technical to avoid overfitting, it appears that the accuracy is .67 for 15 echo and .96 for 50 echo and validation accuracy is .66 for 15 echo and .76 for 50 echo (Figure 6)

```
Model: "sequential_21"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_47 (Conv2D)           (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d_47 (MaxPooling (None, 100, 100, 32)      0
_________________________________________________________________
dropout_13 (Dropout)         (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_48 (Conv2D)           (None, 100, 100, 64)      18496
_________________________________________________________________
max_pooling2d_48 (MaxPooling (None, 50, 50, 64)        0
_________________________________________________________________
dropout_14 (Dropout)         (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_49 (Conv2D)           (None, 50, 50, 128)       73856
_________________________________________________________________
max_pooling2d_49 (MaxPooling (None, 25, 25, 128)       0
_________________________________________________________________
dropout_15 (Dropout)         (None, 25, 25, 128)       0
_________________________________________________________________
flatten_20 (Flatten)         (None, 80000)             0
_________________________________________________________________
dense_39 (Dense)             (None, 128)               10240128
_________________________________________________________________
dropout_16 (Dropout)         (None, 128)               0
_________________________________________________________________
dense_40 (Dense)             (None, 1)                 129
=================================================================
Total params: 10,333,505
Trainable params: 10,333,505
Non-trainable params: 0
```

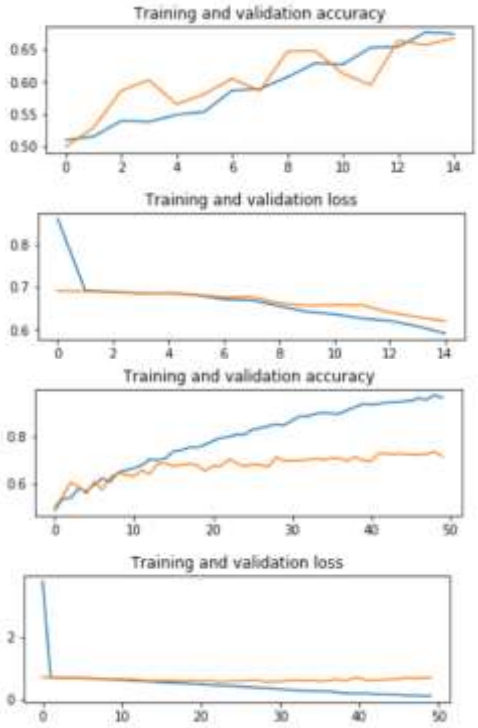

Figure 6 Architecture and Accuracy of Model E

Model F:

The first I will define a two variables train and test data generation (Figure 7) to applying data generation technical in Model F to avoid overfitting and increase accuracy

```
[ ] # create data generators
    train_datagen = ImageDataGenerator(rescale=1.0/255.0, width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True)
    test_datagen = ImageDataGenerator(rescale=1.0/255.0)

    # prepare iterators
    train_it = train_datagen.flow_from_directory(
              train_dir,
              target_size=(200, 200),
              batch_size=64,
              class_mode='binary')


    test_it = test_datagen.flow_from_directory(
              validation_dir,
              target_size=(200, 200),
              batch_size=64,
              class_mode='binary')
```

Figure 7 Create data generation

It’s three block cnn model extends the two block model and adds a third block with 128 filters. so I define function name ModelF() the same model Model C but with applied train_datagen, test_datagen, it appears that the accuracy is .51 and validation accuracy is .2 (Figure 8)

```
Model: "sequential_43"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_113 (Conv2D)          (None, 200, 200, 32)      896
_________________________________________________________________
max_pooling2d_113 (MaxPoolin (None, 100, 100, 32)      0
_________________________________________________________________
conv2d_114 (Conv2D)          (None, 100, 100, 64)      18496
_________________________________________________________________
max_pooling2d_114 (MaxPoolin (None, 50, 50, 64)        0
_________________________________________________________________
conv2d_115 (Conv2D)          (None, 50, 50, 128)       73856
_________________________________________________________________
max_pooling2d_115 (MaxPoolin (None, 25, 25, 128)       0
_________________________________________________________________
flatten_42 (Flatten)         (None, 80000)             0
_________________________________________________________________
dense_83 (Dense)             (None, 128)               10240128
_________________________________________________________________
dense_84 (Dense)             (None, 1)                 129
=================================================================
Total params: 10,333,505
Trainable params: 10,333,505
Non-trainable params: 0
```

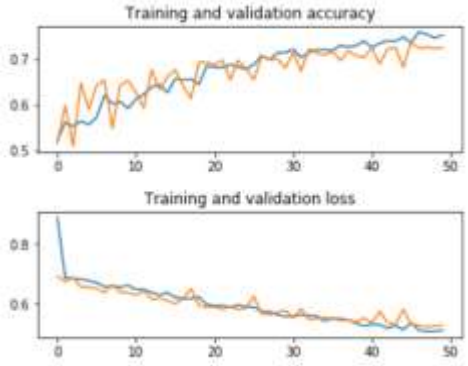

Figure 8 Architecture and Accuracy of Model F

Model V:

The first I will define a one variable for mean data generation for centerin (Figure 9) to applying data generation technical in Model V to avoid overfitting and increase accuracy

```
[ ]   # Create data generator
      datagen = ImageDataGenerator(featurewise_center=True)

      # specify imagenet mean values for centering
      datagen.mean = [123.68, 116.779, 103.939]

      # prepare iterator
      train_it = datagen.flow_from_directory(train_dir, class_mode='binary', batch_size=64, target_size=(224, 224))
      test_it = datagen.flow_from_directory(validation_dir, class_mode='binary', batch_size=64, target_size=(224, 224))
```

Figure 9 Create data generation

Transfer learning is one of the VGG models, such as VGG-16 with 16 layers that at the time it was developed, The model is comprised of two main parts, the feature extractor part of the model that is made up of VGG blocks, and the classifier part of the model that is made up of fully connected layers and the output layer (Figure 10)

```
Model: "model_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_11 (InputLayer)        (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten_61 (Flatten)         (None, 25088)             0
_________________________________________________________________
dense_121 (Dense)            (None, 128)               3211392
_________________________________________________________________
dense_122 (Dense)            (None, 1)                 129
=================================================================
Total params: 17,926,209
Trainable params: 3,211,521
Non-trainable params: 14,714,688
```

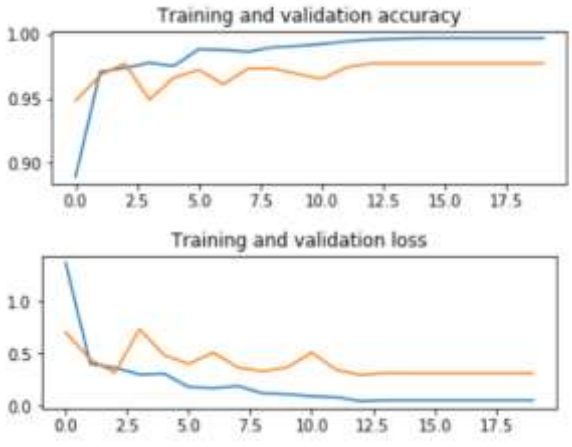

Figure 10 Architecture and Accuracy of Model V

Results

Finally, this (Model V) by fitting a model on the entire training dataset and saving the model to file for later use. then load the saved model and use it to make a prediction on a single image (Figure 11).

```
# load and prepare the image
def load_image(filename):
  # load the image
  img = load_img(filename, target_size=(224, 224))
  # convert to array
  img = img_to_array(img)
  # reshape into a single sample with 3 channels
  img = img.reshape(1, 224, 224, 3)
  # center pixel data
  img = img.astype('float32')
  img = img - [123.68, 116.779, 103.939]
  return img

# load an image and predict the class
def run_example():
    # load the image
    img = load_image(img_path)
    # load model
    model = load_model('final_model.h5')
    # predict the class
    result = model.predict(img)

    if result == 0:
      print ('This is CAT')
    if result == 1 :
      print ('This is DOG')

img_path = train_cats_dir + '/' + 'cat.6.jpg'
img = mpimg.imread(img_path)
plt.imshow(img)

# entry point, run the example
run_example()
```

```
This is CAT
```

Figure 11 Load Image and run Model V

Conclusion

In the previous models, it was noted that the best model leads to high accuracy while avoiding overfitting (Table 1) , it's Model F with accuracy .99 without overfitting. a final model configuration must be chosen and adopted. In this case, to keep things simple and use the VGG-16 transfer learning approach as the final model.

Model	Echo	Loss	Acc	Val oss	Val acc
Model A	15	0.6932	0.5000	0.6931	0.5010
Model B	15	0.0548	0.9895	0.8607	0.6640
Model C	15	0.1337	0.9605	0.7546	0.6930
Model D	15	0.1987	0.9350	0.5619	0.7560
Model E	15	0.5911	0.6745	0.6199	0.6680
	50	0.1060	0.9630	0.6793	0.7160
Model F	50	0.5104	0.7539	0.5282	0.7260
Model EF	50	0.6673	0.5943	0.6750	0.6080
Model V	20	0.0550	0.9966	0.3134	0.9770

Accuracy Models (Table 1)

References

[1] Asirra: A CAPTCHA that Exploits Interest-Aligned Manual Image Categorization, 2007. https://www.microsoft.com/en-us/research/publication/asirra-a-captcha-that-exploits-interest-aligned-manual-image-categorization/

[2] Machine Learning Attacks Against the Asirra CAPTCHA, 2007. https://dl.acm.org/doi/10.1145/1455770.1455838

[3] OverFeat: Integrated Recognition, Localization and Detection using CNN, 2013. https://arxiv.org/abs/1312.6229