Using Deep Learning to classify Dogs and Cats

Ravish Chawla
Conghui Fu
Qiqi Ai
Tianyu Li
Objective

- Learn and use a Deep Learning Library to build a Neural Network for classifying images
- Understand how to optimize neural network in either accuracy or efficiency side
- Participate in the Kaggle competition (dogs vs cats)
Classification

Classification: build a machine learning algorithm capable of detecting the correct cat or dog in new unseen images.
Difference

Traditional algorithms: hand-craft the features.

Deep learning algorithms: done automatically by the algorithm.
CNN Architecture

Convolutional neural networks are a special type of feed-forward networks. These models are designed to emulate the behaviour of a visual cortex. CNNs perform very well on visual recognition tasks.

Special layers: convolutional layers and pooling layers.
The Data

- 25,000 images, with 12,500 cat images and 12,500 dog images
Tools Used

- Computer Vision Library
  - OpenCV

- Deep Learning Libraries
  - TensorFlow
  - Keras with TensorFlow backend

- Systems
  - Amazon Web Services EC2
    - M4-2xlarge, 8 vCPU, 32 GB Memory
  - Macbook Pro with GPU
    - Tensorflow library does not natively support Apple GPU however
OpenCV

- A library of programming functions for real time computer vision.
- Speed: Library of functions written in C/C++, about 30 frames/second, resulting in real time detection.
- Free open source.
- Compatibility: Work with C, C++, Python and so on.
- Not easy to use: Not good module.
Keras

- High-level neural network library capable of using either Tensorflow or Theano as backend
- Allows for easy and fast prototyping
  - Modularity
  - Minimalism
  - Extensibility
- Effective analysis on fewer data
Keras - Power image classification with small data set

- Real world data size challenge
  - Expensive data collection (e.g. medical imaging)
  - Small targets

- Model challenge
  - Overfitting

- Disadvantages of Keras library
  - Less flexibility of setting parameters for model

- Advantages of Keras library
  - Advanced logging for epochs
  - Code is transferable between Theano and Tensorflow without a lot of changes
Keras - Power image classification with small data set

- Modulate entropic capacity - Dropout
  - Data highly correlated

```python
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='rmsprop',
              metrics=['accuracy'])
```
TensorFlow

- Open source software library for machine learning in various kinds of perceptual and language understanding tasks supported by Google.
- Great number of users other than Google developer contribute to this library.
- Easy to use
  - Accessible from C++ applications
  - Support interface to Python, including Ipython/Jupyter
- Compatibility
  - Run on multiple GPUs and CPUs.
  - Run on 64-bit Linux or Mac OS X desktop or server systems
TensorFlow -- Why and Why not

- **Start with TensorFlow:**
  - Easy to obtain,
  - compatible with Python
  - Build in machine learning and neural network functions
  - easy to use compared with other Machine Learning libraries

- **Trouble with TensorFlow:**
  - Require assistance of Computer Vision library.
  - Not good modularity, not easy to tuning.

- **Advantages of TensorFlow:**
  - Built-in Visualization and Analytical tools
  - More flexibility

```python
def conv2d(x, weights, biases, strides=[1, 1, 1, 1]):
    x = tf.nn.conv2d(x, weights, strides=[1, strides, strides, 1], padding='SAME');
    x = tf.nn.bias_add(x, biases);
    return tf.nn.relu(x);

def maxpool2d(x, k=2):
    return tf.nn.max_pool(x, ksize=[1, k, k, 1], strides=[1, k, k, 1], padding='SAME');

def conv_net(x, weights, biases, dropout):
    # reshape input picture
    x = tf.reshape(x, shape=[-1, max_size[0], max_size[1], 3]);

    # convolution layer
    conv1 = conv2d(x, weights['wcl'], biases['bcl'], strides-strides['wcl']);
    print('conv1', conv1.get_shape(), get_size(conv1));

    conv1 = maxpool2d(conv1, k=weights['mp1']);
    print('pool1', conv1.get_shape(), get_size(conv1));

    conv1 = tf.nn.relu(conv1);
    print('relu1', conv1.get_shape(), get_size(conv1));
```
Final Neural Network Configuration

- 3 Layers
- 3x3 filters with 1 stride each
- 32 filters in first layer, 64 filters in second layer, 64 filters in third layer
- Implemented in TensorFlow
Transfer Learning Network

Pretrain a ConvNet on a very large dataset of sufficient size, and then use the ConvNet either as an initialization or a fixed feature extractor for the classification.

- ConveNet as fix feature extractor.
Preliminary Research *(Ravish, Qiqi, Conghui, Tianyu)*

- Researched in CNN Construction, Tensorflow, Keras Construction and Parameters Adjustment *(Ravish, Conghui, Qiqi, Tianyu)*

- Created initial Tensorflow-based Neural Network *(Ravish)*
  - Conducted Experiments on small dataset sample
  - Obtained around 65% test accuracy

- Created Keras-based Neural Network *(Ravish)*
  - Conducted Experiments on small dataset sample
  - Obtained around 85% test accuracy
Process - Second Phase

- Created Second Tensorflow-based Neural Network (Ravish)
  - Conducted experiments on full dataset
  - Obtained around 88% test accuracy
- Extracted Visualizations and Features from trained model for analytics (Ravish)
  - Extracted features from post-trained model to visualize most-activated parts of images
- Implemented Transfer Learning algorithm with Imagenet Convolutional Layers and customized Fully Connected Layers (Ravish)
  - Unable to run validation because of GPU restrictions
- Experimented with Classifiers on Extracted Features of Transfer Learning (Conghui, Qiqi)
  - Used Weka on Imagenet features to run SVM, KNN and combined with Cross Validation, finally received 95%.
- Applied algorithm on Kaggle Test Dataset and Submitted to Leaderboard (Ravish)
  - Leaderboard position 235 with final classifier
- Final Report (Ravish, Conghui, Qiqi)
Experiments on Keras Small dataset Model

- Tested different neural network configurations, including number of layers, size of filters in each layer, and stride length
  - Random Gaussian initializer
  - SIFT Features
- Tested different optimization algorithms, including
  - Stochastic Gradient Descent with decay rate
  - Adaptive Delta
  - ADAM Optimizer
- Tested different training dataset sizes from 2000 to 8000 images
- 50 iterations of the Neural Network with 2000 images run in 1 hour
Experiments on other models

- Tested Tensorflow Full dataset model on 18,000 training images and 2000 test.
  - Each experiment consisted of 10-fold Cross Validation and 100 iterations
  - Experiment takes 8 hours on a non-GPU configuration

- Tested Supervised Classifiers on Extracted Features from VGG16 Model using Weka
  - Experiments consisted of 10-fold Cross Validation on 20,000 images
  - Different supervised classifiers were tested, and adaboost was applied to improve accuracy.
System Architecture - Main Neural Network Classifier

convolution2d_input_1: InputLayer

convolution2d_1: Convolution2D

activation_1: Activation

maxpooling2d_1: MaxPooling2D

convolution2d_2: Convolution2D

activation_2: Activation

maxpooling2d_2: MaxPooling2D

convolution2d_3: Convolution2D

activation_3: Activation

maxpooling2d_3: MaxPooling2D

flatten_1: Flatten

dense_1: Dense

activation_4: Activation

dropout_1: Dropout

dense_2: Dense

activation_5: Activation
System Architecture - Main Neural Network Classifier

Convolution Layers

- 3 Layers
  - 32 Filters of 3x3 size on First Layer
  - 32 Filters of 3x3 size on Second Layer
  - 64 Filters of 3x3 size on Third Layer
- All Convolutional Layers have Pooling of 0.5

Fully Connected Layers

- 2 Layers
  - 512 Neurons with Relu activation on First Layer
  - 2 Neurons with Softmax activation on Second Layer
- Dropout layer with a Keep Probability of 0.5
Data Preprocessing and Augmentation

- Parameters used for Augmenting Training Data:
  - 1/255 Rescaling Ratio
  - 0.2 Shearing range
  - 0.2 Zoom range
  - Horizontal flipping
  - 25° rotation
System Architecture - Imagenet Feature Classifier

- VGG16 Model
  - 16 Layer Model from arXiv paper: "Very Deep Convolutional Networks for Large-Scale Image Recognition"
  - Pretrained Model from Caffee was obtained in Tensor Flow data structure format
  - Input im
# Experimental Results - All Classifiers

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Number of Layers</th>
<th>Configuration</th>
<th>Augmentation</th>
<th>Training Size</th>
<th>Final Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tensorflow + OpenCV</td>
<td>2</td>
<td>32 5x5 Filters 64 5x5 Filters</td>
<td>No</td>
<td>25,000</td>
<td>65% (Training) 48% (Testing)</td>
</tr>
<tr>
<td>Tensorflow + OpenCV</td>
<td>3</td>
<td>32 3x3 Filters 32 3x3 Filters 64 3x3 Filters</td>
<td>No</td>
<td>25,000</td>
<td>49% (Training) 51% (Testing)</td>
</tr>
<tr>
<td>Keras</td>
<td>3</td>
<td>32 3x3 Filters 32 3x3 Filters 64 3x3 Filters</td>
<td>Yes</td>
<td>2,000</td>
<td>82% (Training) 83% (Testing)</td>
</tr>
<tr>
<td>Tensorflow + OpenCV</td>
<td>3</td>
<td>32 3x3 Filters 32 3x3 Filters 64 3x3 Filters</td>
<td>Yes</td>
<td>20,000</td>
<td>97% (Training) 85% (Testing) (Cross-Validation)</td>
</tr>
<tr>
<td>Caffe/Imagenet + Weka</td>
<td>5</td>
<td>VGG16 Model</td>
<td>No</td>
<td>20,000</td>
<td>96% (Training) 95% (Testing) (Cross-Validation)</td>
</tr>
</tbody>
</table>
# Experimental Results - Tensorflow Model

<table>
<thead>
<tr>
<th>Test</th>
<th>Training Loss</th>
<th>Testing Loss</th>
<th>Training Accuracy</th>
<th>Testing Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0.1735</td>
<td>0.4395</td>
<td>0.9589</td>
<td>0.8470</td>
</tr>
<tr>
<td>2</td>
<td>0.13467</td>
<td>0.47313</td>
<td>0.9710</td>
<td>0.8385</td>
</tr>
<tr>
<td>3</td>
<td>0.1430</td>
<td>0.47458</td>
<td>0.9698</td>
<td>0.8423</td>
</tr>
<tr>
<td>4</td>
<td>0.07968</td>
<td>0.53282</td>
<td>0.9785</td>
<td>0.8535</td>
</tr>
</tbody>
</table>
Screenshots

Filters from First Convolution Layer on some sample images
Charts and Demo

- Accuracy

- Accuracy/     - Accuracy/ (raw)     - Accuracy/Validation

- Loss

- Loss/     - Loss/ (raw)     - Loss/Validation

Link to Demo (with Tensor Board).
Summary and Take-aways

- Unix environments are better suited for Deep Learning because of support from Tensorflow and other libraries. However, their lack of support for GPU on Macs makes model training very computationally expensive.
- Transfer Learning is the best approach for Image Neural Network learning. Because existing classifiers like VGG16 have already trained large networks, using the top layers with new Fully Connected Layers can reduce running time and computation, and help train a model by using features from a larger dataset. Features from these models can also be augmented by using Supervised Learning algorithms such as SVM.
Summary and Take-aways

- Data Augmentation is essential for obtaining higher prediction accuracy and learning good features. It reduces overfitting on the training dataset and provides more images to be trained from without requiring a large amount of data.
- Convolution Layer configuration is not as important as initially hypothesized. All of the results were obtained using the same number of neurons and layers, and adding additional layers did not affect performance significantly. Improvement in accuracy was primarily observed by tuning different parameters for data preprocessing, data augmentation, and size of training sample.
Next steps

- Fine tune network to obtain at least 90% accuracy
- Take advantage of the data that is not being used (> 10,000 images)
- Experiment with using features from a pre-trained network (ie. Imagenet)
- Run our model on the Kaggle Test dataset and submit to leaderboard
- Implemented Boosting
Code

- Code is available on GitHub Repository:  
  https://github.com/ravishchawla/DogsVsCats/

- Primary Classifier:  
References

Very Deep Convolutional Networks for Large-Scale Image Recognition
K. Simonyan, A. Zisserman
arXiv:1409.1556

Building powerful image classification models using very little data
https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html

A Practical Introduction to Deep Learning with Caffe and Python
http://adilmoujahid.com/posts/2016/06/introduction-deep-learning-python-caffe/

ImageNet Classification with Deep Convolutional Neural Networks

Cats and Dogs and Convolutional Neural Networks

Transfer Learning Algorithms for Image Classification