Chetan Borse

Hotel Image Classification

1. Introduction

1.1 Problem Statement

This is a real world image classification problem. The objective is to classify images from hotels into one of 8 classes: 'Bathroom', 'Guestroom', 'Pool', 'Gym', 'Restaurant', 'Lobby', 'Aerial View', 'Business Center'.[1]

Note: Image database can have corrupted images as well.

1.2 Data Description

1. Nomenclature:

All training and test images are named as <image id>.jpg

2. Image type and Color space:

Image data consists of JPG images of varying size. Few images have CMYK color space as well.

3. Labels:

Image has correct class set with either 1 or 0; but test result can have posterior probabilities for each class.

4. Error Metric:

Evaluation metric will be Multi-class log loss.

i.e. Log Loss = -1/N ($\sum_{i=0}^{N} \sum_{j=0}^{M} y\square j \; X \; \log(pij)$)

5. Source:

http:://orwell.uncc.edu/6156/project.tar

1.3 Approach

This image classification algorithm is based on Convolutional Neural Network architecture.

First, images are preprocessed using different techniques such as color-space conversion, image resizing, image normalization, PCA transformation. Then, preprocessed images are fed to Convolutional Neural Network in order to learn complex, non-linear features and the end result will be probabilistic prediction for each class.

2. Image Preprocessing

2.1 Introduction

Image preprocessing is a set of operations with images at the lowest level of abstraction. The aim of preprocessing is an improvement of the image data that suppresses unwanted distortions and enhances some image features important for further processing such as classification, clustering, etc.

2.2 Preprocessing Techniques

2.2.1 Handle Corrupted Image

The image database has few corrupted images. So, first identified such corrupted images through Image Viewer software and noted down its ids.

Developed Python script has a feature named 'filter', which handles such noisy images. So, once a list of corrupted images is provided to 'filter' in a script, it filters out these images and do not consider it for further image preprocessing.

2.2.2 Color-space Conversion

Images read from disk are all JPG images with 'RGB' color-space. Thus, it has three different color channels.

For reducing dimension of image data, only one gray-scale color channel can be used instead using all three color channels. So, images can be converted into 'Grayscale' color-space. It reduces dimension of image data; but on the other hand, it loses the part of information as well.

2.2.3 Normalize Image

Normalization/standardization is one of the important stage in image preprocessing.

If images are not normalized, set of few features i.e. pixels can dominate all other features/pixels and we may get bad classification performance. Normalization rescales the data such that every pixel/feature will be given a same importance.

Principle Component Analysis (PCA) also requires image data to be normalized first. Otherwise, PCA transforms original data such that it always returns first principle component with variance close to 90-99%; thus almost discarding every other principle component.

2.2.4 Transform Pixel Type

An array of elements associated with every image is transformed into 'numpy.float32' type, so that transformed data can easily be used for further preprocessing like resizing image without any significant script error.

2.2.5 Resize Image

Initially, all the images are of different sizes. So, it is first resized to specified standard resolution i.e. 64x64. The motive of image resizing is to scale down all images to a same spatial size so that it can easily be processed further with fixed size array.

2.2.6 Flatten Image

After resizing an image to a specified standard resolution, its pixel array is flattened and all the images thereafter stacked together in the matrix.

2.3 Storage Techniques

2.3.1 Storage Issue

Preprocessed image data is too large (of the order of few GBs). So, loading entire image data into memory is a critical process and it can break the system if it is not compatible to handle such large data. It also slows down high performance computing systems.

2.3.2 Solution

2.3.2.1 Memory-mapped File

One of the good solution to handle this issue is to use Memory-mapped File.[2]
A memory-mapped file is a segment of virtual memory that has been assigned with some portion of file or file-like resource. This resource is typically a file that is physically present on disk, but can also be a device, shared memory object, or other resource that the operating system can refer through file descriptor.
Thus, it increases I/O performance, especially when used with large data.

2.3.2.2 Python equivalent

Python provides 'numpy.memmap' function, which is equivalent with Memory-mapped File.
It creates a mempry-map to an array stored in a binary file on disk. It is used for accessing small segments of large files on disk, without reading the entire file into memory. One of the advantage of using 'numpy.memmap' is that it is an array-like object making it simpler to use.[3]

3. Dimensionality Reduction

3.1 Issue

If we work on image patch having size 64x64 with 3 color channels, it will generate total 64x64x3 (=12288) features.
Running machine learning model on such high dimensionality data is a bad idea, because -

1. More time will be required for computations.
2. As few features can be co-related, using them without dimensionality reduction will have lower classification performance.
3. As image can be represented using a set of few features, no need to use all pixels in image data.

3.2 Solution

3.2.1 PCA

The best solution for dimensionality problem is to apply Principle Component Analysis (PCA) on preprocessed image data.
PCA is a statistical procedure that uses an orthogonal transformation to convert possibly correlated features into set of linearly uncorrelated variables called Principle Components. This transformation is defined in such way that first few principle component has the largest possible variance contributing around 90-95% of the total variance.
PCA transforms original image data into a new set of dimensions that are uncorrelated. Thus, it further improves classification task.

3.2.2 Implementation

In the implementation, we learn PCA model on preprocessed image data with 32x32x3 principle components. Thus, it reduces image data by almost 75% and information loss (i.e. variance loss) is less than around 5%.
Same learned PCA model is used to fit PCA transform to test image data.

4. Machine Learning Model

4.1 Introduction

The main issue with image classification is the difficulty in finding useful features. The manual approach for handcrafting features like shapes, edges, regions is not an easy and appropriate solution.
So, the solution is to use a special version of neural network, i.e. Convolutional Neural Network architecture (i.e. CNN).[4] CNN along with learning a model for classification automatically creates useful, complex and non-linear features. CNN learns complex representation of features which are also invariant to some types of transformations like rotations, transpositions, scaling.
Below are the few well known successful examples:

1. CIFAR-10
2. LeNet
3. AlexNet
4. mnist digit recognizer

4.2 Architecture

4.2.1 Layer Arrangement

Below is the layer arrangement in implemented CNN architecture:

Input Layer ->
Dropout Layer ->
(Convolutional Layer -> Max pooling Layer) * 2 ->
Convolutional Layer ->
Dropout Layer ->
Fully Connected Layer ->
Output Layer

4.2.1.1 Input Layer

Input to CNN model is the preprocessed image data and it is fed to machine learning model through 'Input layer'.
The shape of input layer is described as Width*Height*Channels,
Where,

- Width*Height: It is also called as spatial size of image (i.e. resolution of image).
- Channels: Number of color channels in preprocessed image. For getting benefit of feature learning using CNN and for preserving more image details, all three Red, Green, Blue color channels are used.

The shape of input layer is specified by user depending upon input. In the implementation, we are using PCA transformed images as an input to CNN and keeping all three Red, Green, Blue channels in images. Hence, input layer's shape is set to 32x32x3.

4.2.1.2 Convolutional Layer

The Convolutional Layer is the building block of CNN architecture which does the most of processing. During the forward pass, each filter is convolved across the spatial size of input volume generating more complex representation of features.

1. **Filters:**
   The convolutional layer consists of a set of learnable filters. Every filter in the implementation is small in size, i.e. 5x5, but it extends through the full depth of the input volume. Every filter (or neuron or activation unit) is connected to only a local region of the input volume (i.e. Receptive Field).
2. **Stride:**
   In the implementation, the stride with which we slide the filter across input volume is 1. It means that it moves the filters one pixel at a time.
3. **Zero Padding:**
   In order to preserve the spatial size of the input volume, we have set zero padding to 1 in CNN convolutional layer.
4. **Weight Sharing:**
   In the implementation, weight sharing is enabled at convolutional layer is enabled, so that number of parameters in CNN model can be reduced. Thus, reducing time required for learning further.
5. **Non-linearity Function:**
   In the CNN model, activation units in convolutional layer implements Rectified Activation function i.e. ReLU.
   It can be defined as,
   $$f(x) = \max(0, x)$$
   ReLU improves the performance of CNN and also resolves an issue of vanishing gradients.

4.2.1.3 Max Pooling Layer

This layer is inserted after Convolutional layer to reduce the spatial size of the representation. It ultimately reduces number of parameters to be learned in CNN and hence, overfitting can be implicitly avoided.

1. **Pool Size:**
   The implementation has pooling filter with size 2x2.
2. **Stride:**
   Stride size for max pooling layer is set to 2x2.
3. **Zero Padding:**
   We do not set zero padding in pooling layer. Hence, implementation has zero padding equal to 0.

4.2.1.4 Dropout Layer

Neural network models usually overfit to data because of too many parameters to be learned. The alternative solution to avoid overfitting in CNN is to insert Dropout Layer.
During training, dropout layer drops output of activation units with dropout probability equal to either 0.2 or 0.5. Thus, this randomness effectively reduces overfitting of model.

4.2.1.5 Fully Connected Layer

Activation units in a fully connected layer have full connections to all activations in the previous layer.

1. **Activation Units:**
   The implementation of fully connected layer has 512 activations.
2. **Non-linearity Function:**
   Similar to convolutional layer, fully connected layer implements activation units with Rectified Activation function i.e. ReLU.

4.2.1.6 Output Layer

Output layer has total 8 activations corresponding to every label in image classification. Output layer implements activations with Softmax function, so that probabilistic prediction can be output.

4.2.2 Weight Initialization

Weight initialization in CNN architecture is done with 'Xavier initialization' (also known as Glorot initialization), which samples set of weights from the uniform distribution.

4.2.3 Weight update

Weight updation in CNN architecture is done with 'Nesterov momentum', which first makes a step into velocity direction and then make a correction to a velocity vector based on a new location.

4.3 Hyper Parameters

4.3.1 Learning Rate and Momentum

Both Learning Rate and Momentum controls learning speed of model. If we set low learning rate, gradient descent converges slowly. On the other hand, if we set it to higher value, it descends quickly; but there is a risk of diverging instead converging.

Learning rate and momentum have tradeoff between them. So, we have set momentum to higher value i.e. 0.95 and learning rate to relatively lower value i.e. 0.001. Hence, it balances both parameters and models yields better performance.

5. Performance Improvement

To pick the best model learned and ultimately to improve the performance of CNN model, we have implemented 'Early Stopping' and 'Pick the best model' techniques.[5]

5.1 Early Stopping

Early stopping is the form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent.
So, if model starts to overfit after certain number of iterations, it stops training and returns the best model learned so far.

Algorithm:

1. For every iteration in training,
   a. If validation error < the best validation error,
      Then,
      Save weights of model learned with least validation error.
   b. Else if validation error is not improved for last 'K' iterations,
      I. Stop training.
      II. Return weights of the best learned model (saved with 'EarlyStopping' class).

5.2 Pick The Best Model

There can be the scenario, where validation error reduces gradually with small fraction and takes too many iterations to converge. This may take a lot of time for training the model.
Hence, to stop training after specified number of iterations and to return the best model learned so far, 'PickBestModel' class is implemented.

Algorithm:

1. For every iteration in training,
   a. If validation error < the best validation error,
      Then,
      Save weights of model learned with least validation error.
   b. Else if training reaches to specified number of maximum iterations,
      I. Stop training.
      II. Return weights of the best learned model (saved with 'PickBestModel' class).

6. Algorithm

@Input: XTrain => (nTrain * 1) 1-d vector of image ids (part of training images).
@Input: YTrain => (nTrain * 8) Target matrix having one column for each label.
@Input: TrainImages => Training images of different resolution.
@Input: XTest => (nTest * 1) 1-d vector of image ids (part of test images).
@Input: TestImages => Test images of different resolution.
@Output: => (nTest * 8) Target matrix having one column for each label
(Note: Here, it is a probabilistic prediction).

---

1. Preprocess image data (both training and test data).
   a. Identify and filter corrupt images.
   b. Read image from the disk.
   c. Transform color space of image if specified (Note: In the implementation, we have set it to RGB only in order to preserve as much as possible information).
   d. Normalize image data.
   e. Transform pixel type to 'float32'.
   f. Resize images to specified standard resolution (Note: In the implementation, we have set it to 64x64).
   g. Flatten pixel data and store it to memmap array.
2. Apply PCA for dimensionality reduction. (Note: In the implementation, we have set PCA transformation with 32x32x3 principle components).
3. Build a convolutional neural network architecture with specified hyper parameters and learning techniques such as Early Stopping and Pick the Best Model.
4. Train a CNN model with transformed training images.
5. Predict probabilistic labels for transformed test images using learned CNN model.
6. Summarize and save the predictions.
7. Stop the algorithm.

7. Experiments

7.1 Statistics

Multiclass Log Loss		
Training Error	**Validation Error**	**Test Error**
1.2326	1.2512	1.26251

7.2 Output

Note: This is the final output for CNN model and its hyper parameters described as above.

```
Chetans-MacBook-Pro: chetan$ ipython project.py
Reading and preprocessing training data from csv..
Reading and preprocessing training data from csv is finished!
Reading and preprocessing image data..
Reading and preprocessing image data is finished!
Building a model..
Building a model is finished!
Training a model..
# Neural Network with 2181096 learnable parameters
## Layer information
#       name           size
---  ----------------  --------
 0  input          3x32x32
 1  dropout_first    3x32x32
 2  conv2d_first     32x32x32
 3  maxpool2d_first   32x16x16
 4  conv2d_second    32x16x16
 5  maxpool2d_second  32x8x8
 6  conv2d_third     64x8x8
 7  dropout_third    64x8x8
 8  dense          512
 9  output         8

  epoch    train loss    valid loss    train/val    valid acc  dur
-------  ------------  ------------  -----------  -----------  -------
    1     1.90797       1.71052       1.11543      0.35183  510.22s
    2     1.67105       1.56333       1.06891      0.39323  513.48s
    3     1.56204       1.48668       1.05069      0.42071  505.89s
    4     1.49741       1.43067       1.04665      0.43999  505.60s
    5     1.46777       1.41068       1.04047      0.45301  479.80s
                  ……..
       10     1.39173       1.32614       1.04946      0.49340  478.74s
                  ……..
       20     1.32136       1.28748       1.02632      0.50968  477.96s
                  ……..
       40     1.24658       1.27102       0.98077      0.52637  478.08s
                  ……..
```

```
44    1.23269    1.25124    0.98517    0.52715  477.35s
      ..............
55    1.19820    1.26313    0.94860    0.52781  317.58s
      ..............
65    1.16877    1.26442    0.92436    0.52926  317.56s
      ..............
69    1.16245    1.26196    0.92114    0.53041  317.69s
Early stopping..
Best validation loss:  1.25124324731
At epoch:  44
Loaded parameters to layer 'conv2d_first' (shape 32x3x5x5).
Loaded parameters to layer 'conv2d_first' (shape 32).
Loaded parameters to layer 'conv2d_second' (shape 32x32x5x5).
Loaded parameters to layer 'conv2d_second' (shape 32).
Loaded parameters to layer 'conv2d_third' (shape 64x32x5x5).
Loaded parameters to layer 'conv2d_third' (shape 64).
Loaded parameters to layer 'dense' (shape 4096x512).
Loaded parameters to layer 'dense' (shape 512).
Loaded parameters to layer 'output' (shape 512x8).
Loaded parameters to layer 'output' (shape 8).
Training a model is finished!
Reading and preprocessing image data..
Reading and preprocessing image data is finished!
Predicting..
Predicting is finished!
```

7.3 Observations

7.3.1 Low vs High Learning Rate

Increasing or decreasing learning rate does not affect accuracy. It only speed up or slow down training phase.

e.g.

For learning rate = 0.001, we get the best validation error at 87th iteration.

For learning rate = 0.0001, we get the best validation error at 157th iteration.

7.3.2 Less vs More Complex CNN Model

Increasing complexity of CNN model increases number of parameters to be trained. Thus, there are high chances of overfitting of model.

But if the overfitting is handled appropriately, then the model outperforms and has slightly better generalization error.

e.g.

Scenario	Multiclass Log Loss		
	Training Error	Validation Error	Test Error
Learnable Parameters = 2181096	1.2326	1.2512	1.26251
Learnable Parameters = 1092872	1.21785	1.26552	1.28200

7.3.3 Grayscale vs RGB Color space

Grayscale image has only one color channel. On the other hand, RGB image has 3 color channels, i.e. Red, Green and Blue. RGB image holds more details than Grayscale image.
So, if RGB image is fed to CNN model, it learns better feature representation and ultimately outperforms.

7.3.4 Size of Pooling Filter

Size of pooling filter should be kept as small as possible, i.e. 2x2.
If filter size is increased, more information will be lost. Hence, performance of CNN model also reduces.

7.3.5 Regularization

Dropout layers are inserted with dropout probability 0.2/0.5, which controls overfitting of the model.
Without dropout layer, training error gradually reduces and validation error gradually increases after certain number of iterations causing overfitting of the model.

8. Libraries

1. OpenCV (http://opencv.org/downloads.html)
2. Theano (http://deeplearning.net/software/theano/)
3. Lasagne (http://lasagne.readthedocs.io/en/latest/)
4. nolearn (https://pythonhosted.org/nolearn/)

Note: Modify definition of predict(self, X) function in .../anaconda/lib/python3.5/site-packages/nolearn/lasagne/base.py for returning a vector of probabilistic prediction instead hard prediction.

9. References

1. https://inclass.kaggle.com/c/hotel-image-classification
2. https://en.wikipedia.org/wiki/Memory-mapped_file
3. http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.memmap.html
4. http://cs231n.github.io/convolutional-networks/
5. https://en.wikipedia.org/wiki/Early_stopping