Machine Learning Engineer Nanodegree

Capstone Proposal

Yuwei Chen
September 8, 2019

Proposal

Domain Background

The dog_app project is a visual object detection based on deep learning. Visual object detection is an important topic in computer vision, and has great practical merits in applications such as visual surveillance and autonomous driving. In recent years, significant breakthroughs of deep learning methods in image recognition research have arisen much attention of researchers and accordingly led to the rapid development of visual object detection.

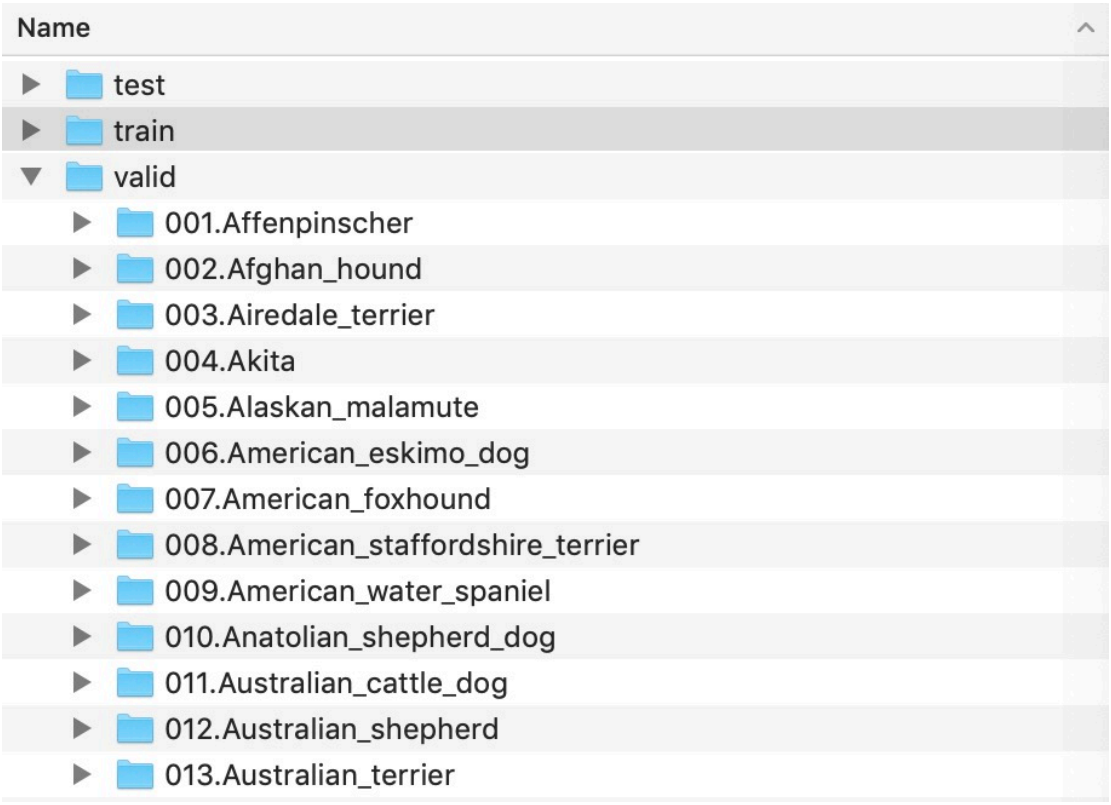
The paper <<Machine Learning Methods for Visual Object Detection>> gives me a simple introduction about how machine learning was applied to visual object detection, proposing some useful methods like Histogram of Oriented Gradient (HOG), Local Binary Pattern (LBP) and Local Ternary Pattern (LTP) .
(link http://www.theses.fr/2011GRENM070.pdf)

Problem Statement

The purpose of this application is to achieve the following:
1. If detecting an image of a dog, return the breed.
2. If detecting an image of a person, return the resembling dog breed.
It can be divided into three steps:
1. Apply a pretrained network VGG16 to detect dog images.
2. Build a network by myself to classify the dogs;
3. Use transfer learning to predict dog breeds for higher accuracy.

Datasets and Inputs

The dataset for this project has a dog dataset named ‘dog_images’ and human dataset named ‘lfw’. There is no annotations or bounding boxes included for each image.
The ‘dog_images’ folder is shown below, which is divided into three parts: train, valid and test, including 133 kinds of dog breed, amounted to totally 8351 images:



While the lfw file contains totally 13233 images with different categories.

Each image have 3 color layers (RGB). It is better to check whether the dataset is balanced, by counting the frequency of each class. If not balanced, we can enlarge the dataset. The proportion of training, validation and testing set are usually 70%, 15% and 15%.

Solution Statement

The solution I use is mostly learned from VGG_16 and Resnet_50. I would like to create the convolution layers and pooling layers with the same kernel size and orders, followed by fully connected layer leading to a 133 elements vector representing 133 categories of dogs. With regard to transfer learning, I also utilize the pretrained Resnet_50, only change the last fully connected layer to get exactly 133 output features.

Benchmark Model

The model I would like to talk about is VGG_19, based on basic VGG network and Resnet, which are both typical tool for extracting image features. The critical block of Resnet_50 and VGG are listed below:

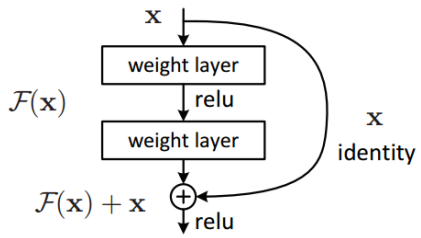
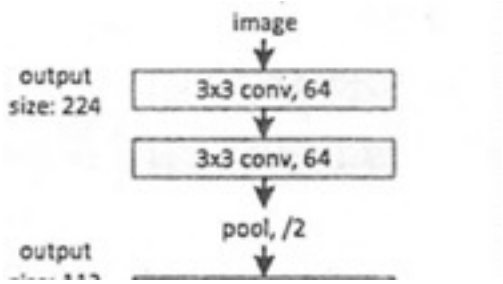
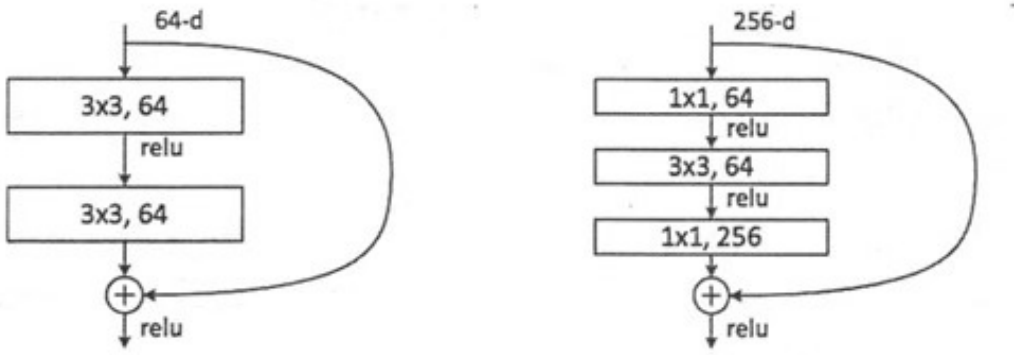


Figure 2. Residual learning: a building block.



Combine these two structures, VGG_19 has a network structure presented below:



The VGG_19 has a total of 34 layers, using conv layers to extract features and short cut to implement information loss.

Evaluation Metrics

The evaluation metrics are accuracy and test loss.
Test loss is updated by the formula below:
test_loss = test_loss + ((1 / (batch_idx + 1)) * (loss.data - test_loss))
while accuracy is defined as following:
Accuracy = (TP+TN)/(TP+TN+FP+FN) (only when the classes are balanced)

	Actual = Yes	Actual = No
Predicted = Yes	True Positive(TP)	False Positive(FP)
Predicted = No	False Negative(FN)	True Negative(TN)

If the dataset is unfortunately unbalanced, which means the training set has different classes distribution from testing set, other metrics should be taken, like Recall, Precision and F1 score.

Recall = TP/(TP+FN)
Precision = TP/(TP+FP)
F1 = 2*((precision*recall)/(precision + recall))

Project Design

The overall review of my solution is presented in the “solution statement” part. Here I will talk about some details and show the critical code.

Before building the network, we should load the data from the files(train, valid and test). It is necessary to transform the pictures into the standard format suitable for the pretrained VGG model. Also, set the batch size to be 100. The transformation is achieved by the following code:

```
normalize = transforms.Normalize(mean=[0.485, 0.456, 0.406],
                                  std=[0.229, 0.224, 0.225])
data_trans=transforms.Compose([transforms.Resize(256),
                               transforms.CenterCrop(224),
                               transforms.ToTensor(),
                               normalize,
                              ])
```

First is the self-built CNN network, each block is organized by a convolution layer (kernel size 3*3), a activation function(relu), a max pooling layer (kernel size 2*2) and a dropout layer to reduce overfitting. The fully connected layer is activated by sigmoid function. It is worthy to mention that, between conv layers and fc layers, there is a reshape step to make the features suit the shape of fc layer input. The code is below:

```
x = self.conv1(x)
x = F.relu(x)
x = self.pool(x)
x = self.drop(x)


x = self.conv2(x)
x = F.relu(x)
x = self.pool(x)
x = self.drop(x)

x = self.conv3(x)
x = F.relu(x)
x = self.pool(x)
x = self.drop(x)


x = x.reshape(x.size(0), -1)

x = self.fc1(x)
x = F.sigmoid(x)

x = self.fc2(x)
x = F.sigmoid(x)
```

The optim function is Adam and the loss is measure by cross entropy.

Transfer learning is carried out as following:

```
## TODO: Specify model architecture
model_transfer = models.resnet18(pretrained=True)
for param in model_transfer.parameters():
    param.requires_grad = False

# Parameters of newly constructed modules have requ
num_ftrs = model_transfer.fc.in_features
model_transfer.fc = nn.Linear(num_ftrs, 133)
```

With the same loss function and a optim method SGD.