Dog Breed Classification Project Report

Defination

Background

The project focus on using deep learning for image classification problem. Image classification is a type of supervised learning, where the input is an image (can be represented as a 2D/3D feature array) and the output is the probability of the image beloning to certain class. Convolutional neural network (CNN) has been proven as an effective method for this problem. One popular image classification challenge in the ImageNet contest: ImageNet Large Scale Visual Recognition Challenge (LSVRC)[1]. The challenge is to correctly classify 1000 classes. Winners of the ILSVRC include several break through in CNN architecture: AlexNet2011[2] , VGG16201[3], ResNet502015[4]. These architectures are often used as pretrained model in transfer learning.

Project statement

In this project, we will build a neural network to identify dog breed from a supplied dog image. In specific, when given an arbitrary image, the algorithm should first identify if it is a dog or a human. If dog is detected, the code will return the estimate of breed. If human is detected, the code will return the resembling dog breed.

I choose this project because I find deep learning is very fascinating. And I want to gain experience in building neural network to solve a practical problem.

Evaluation Metrics

For dog/human detection, we can evaluate the accuracy. We also would like to focus on false positive and false negative rate. For example, given a set of human images, the human detector can failed to detect how many images. Given a set of dog images, how many images the human detector may mistakenly detect dog as human.

False positive rate = FP / (FP + TN)

False negative rate = FN / (FN + TP)

For dog breed classification, as this is a multiclass classification, we can use cross-entropy as optimization and evaluation metric. We can also use accuracy as evaluation metric.

Cross-entropy Loss: $L = -\frac{1}{N}\sum_{i=1}^{N_{samples}}\sum_{c=1}^{N_{class}} t_{ic}log(\hat{y}_{ic})$

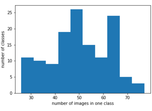
Accuracy: $\frac{\text{number of correctly classified samples}}{\text{number of samples}}$

Exploratory Analysis

Data Set

Two data set are provide by Udacity. (1) Dog data set (https://s3.us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip):

```
* 8351 dog images of 133 dog breed categories
* split into train, validation and test set, with 6680, 835, 836 images respectively.
* In training set, each class has 39-70 images.
```



(2) Human data set (http://vis-www.cs.umass.edu/lfw/lfw.tgz):

```
* 13233 human face images
```

Images in both dog and human dataset are colored, so the input data will be *nW* x*nH* x*nC* where *nC*=3. If we are going to detecting faces, gray scale image might be enough, but for identifying dog breed, the color may be necessary. A sample image from dog and human dataset is shown below:

Benchmark Model

For dog breed classification, I tried to use AWS Sagemaker's build in image classification. I choose to use pretrained 18 layer network and fine tune the last layer. However, I can only archieve 1% accuracy on both training and validation, and early stopping was triggered before 30 epochs. The algorithm is kind of black-box to me and I am not sure how to improve it.

Below are my code to start a image classification training job on Sagemaker:

In [ ]:
```
ic = sagemaker.estimator.Estimator(
                training_image,
                role,
                train_instance_count=1,
                train_instance_type='ml.p2.xlarge',
                train_volume_size = 50,
                train_max_run = 360000,
                input_mode= 'File',
                output_path=s3_output_location,
                sagemaker_session=sess,
                base_job_name='project-dog',
                 )
ic.set_hyperparameters(num_layers=18,
                       use_pretrained_model=1,
                       image_shape = '3,224,224',
                       num_classes=133,
                       num_training_samples=6678,
                       mini_batch_size=32,
                       epochs=30,
                       learning_rate=0.001,
                       precision_dtype='float32',
                       early_stopping=True,
                       optimizer='adam',
                       checkpoint_frequency=5,
                      )
```

Implementation and Results

Human Detection

Detecting human faces is a classic problem in computer vision, and there are already quite a few existing implementations we can use directly. We experimented with two options:

1. OpenCV Cascade Classifier

Cascade Classifier[5] extracts a large number of simple features (linear feature, edge feature etc) from an image and uses Adaboost to select a small set of effective features. It combining weak classifiers based on these features in a "cascade" method, which allows non-relevant region of the image to be quickly discarded and focus the computation on promising regions.

1. AWS Rekognition Service

AWS Rekognition is an image/video analysis service to detect and recognize objects, scenes, and faces in images. It requires no machine learning knowledge and provide an API we can access from app. For a sample image, the service returns a set of labels and corresponding confidence such as:

```
    'Person': 98.46886444091797,
    'Human': 98.46886444091797,
    'Teacher': 97.05602264404297,
    'Indoors': 94.71190643310547,
    'Room': 94.71190643310547,
    'Judge': 86.63158416748047,
    'Court': 84.27397155761719,
    'Senior Citizen': 61.337562561035156,
    'Head': 61.05041122436523}
```

We will use the `Human` label for human detection (we can also use it for dog detection with the `Dog` label).

For both of these two methods, we will test with 100 human files and 100 dog files, the formal will inform as the false negative rate, and the latter will indicate false positive rate. Both OpenCV and AWS successfully detected human in all 100 human images, indicating 0% false negative rate. OpenCV detects human in 10 out of 100 dog images, where as AWS Rekognition detects human in 11 dog images, suggesting false positive rate is around 10%. However, if we looked at the images closely, we find a few of them does have human in it:

Dog Detection

For Dog detection, we will use pretrained models provided in Pytorch. The Pytorch pretrained models are trained on imagemet images and output the probability of the image being to 1000 classes. If the class index of largest probability corresponds to dog classes (in the range of [151, 268]), then dog is detected. We tested two pretrained models, Vgg16 and ResNet50, on 100 human images and 100 dog images. The results is as follows:

model	100 human images	100 dog images
Vgg16	0% dog detected, FP rate=0	99% dog detected, FN rate=1%
ResNet50	1% dog detected, FP rate=1%	99% dog detected, FN rate=1%

As the results are similar, it suggested smaller network Vgg16 is sufficient for this problem.

Dog Breed Classification

For dog breed classification, we can either train a neural network from scratch or use transfer learning. The architecture neural network of image classification typically consists of two part: feature extraction part and classifier part. The feature extraction part usually consists of a series of convolutional layers. The classifier usually consists of a few fully-connected/linear layers and the final layer would out put N_class numbers that can be used to compute loss function. For transfer learning, we can use a pretrained model and change the last layer to match the number of classes in our problem. We will fix the weight of feature extraction part, and fine tune the classifier part.

The workflow is as follows:

- Read images
- Preprocess images. This step is includes resizing all training images to the same size, rescale the pixels in the range that is expected for pretrained model.
- Data Augmentation. We can use random flip or rotation to augment the dataset. Pytorch does the augmentation on the fly conveniently: random augmentation is performed each time when a batch of image is loaded.
- Training: We use Adam optimizer with default parameters (lr=0.001). Model weights are adjusted according to CrossEntropyLoss. The validation dataset is used to monitor the loss function and accuracy during training process. Best model is saved according to the performance on validation dataset.
- Testing: after loading the best model, we can evaluate metrics on the test dataset.

The network trained from scratch uses a custom architecture:

```
Feature part: 3 Conv2D layer with 64, 128, 256 4*4 filters, atride=2, padding=1, each followed by a ReLU activat
ion.
Average pooling layer with output size=7*7. the average pooling layer is especially useful when we test on our
own images later, adapt any input size to to a fixed size to feed to the following fully connected layers.
Classifier part: we have two fully connected layers:
1. input 12544, output 1024, followed by a dropout layer with dropout rate=0.2, and ReLU activation.
2. input 1024, output 133 (which is the number of dog breed classes). There is no softmax followed because Cross
Entropyloss will take care of it.
```

For transfer layer, we work on top of Vgg16 pretrained model. The classifier of Vgg16 consists of 5 groups of 13 convolutional layers, each group uses a maxpool2D layer to reduce the height and width to 1/2. Vgg16 then has three fully connected layers as classifier which we will replace and fine tune. Our new classifier architecture has three fully connected layer with output feature = 4096, 1024 and 133, ReLU is applied the two hidden layers.

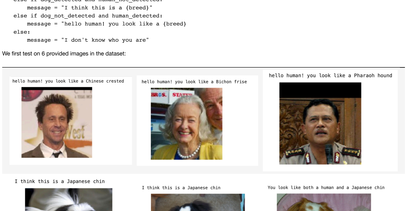
For both training from scratch and transfer learning, we trained 10 epoches, and achieved 11% and 73% accuracy on test set, respectively.

Combined algorithm

We combine OpenCV human detector, Vgg16 dog detector and Fine tuned Vgg16 dog breed classifier to design a deliverable function that achieve the project goal: predict dog breed given a dog image or identify a resembling dog breed given a human image. The function follows the following logic:

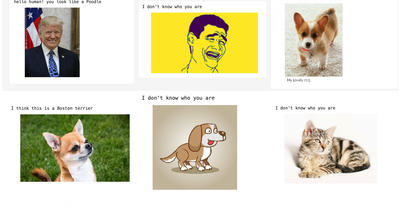
```
detect human
detect dog
predict dog breed

if dog_detected and human_detected:
    message = "You look like both a human and a (breed)"
else if dog_detected and human_not_detected:
    message = "I think this is a (breed)"
else if dog_not_detected and human_detected:
    message = "hello human! you look like a (breed)
else:
    message = "I don't know who you are"
```

We first test on 6 provided images in the dataset:



The human and dogs are correctly detected. However, this test is optimistic because the images were used for training.

We then test some of my own images:



The results are pretty interesting,

- It correctly identified Cogi
- It correctly identified Trump as human
- It mistakenly identify Chiwawa as Boston terrier
- It didn't identify cat as dog.
- It cannot identify catton dog or human.

References

[1] Russakovsky, Olga, et al. "Imagenet large scale visual recognition challenge." International journal of computer vision 115.3 (2015): 211-252.

[2] Krizhevsky, Alex; Sutskever, Ilya; Hinton, Geoffrey E. (2017-05-24). "ImageNet classification with deep convolutional neural networks" (PDF). Communications of the ACM. 60 (6): 84–90. doi:10.1145/3065386. ISSN 0001-0782.

[3] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[4] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[5] Viola, Paul, and Michael J. Jones. "Robust real-time face detection." International journal of computer vision 57.2 (2004): 137-154.

In [ ]:
```

```