Udacity capstone report - Dog Breed Classifier

Overview

There are situations where people are curious about the species of an animal or plant and want to identify it. Image classification is the perfect solution to this problem. In this project, we will classify dog breed from a given image and measure the model's accuracy. The model should be capable of generalizing to new images from one of the recognized breeds. Similar methods can be applied to identify other objects. It can also suggest the dog breed most similar to a person.

To achieve this, we will use a deep convolutional neural network, a common technique in computer vision. Techniques such as image classification, image recognition, and object detection have found wide applications from enhancing accessibility to security screening to auto-driving. There are certain unique attributes of these neural networks - they need to deal with high-dimensional inputs as images are a large array of pixels, and also robust to variations as the objects in the images are subject to translation, rotation, scaling, and different illuminations.

Analysis

Dataset

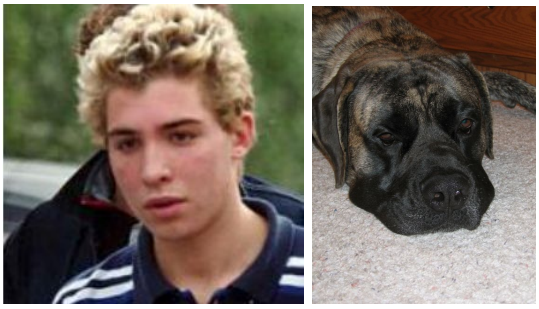
Since we want to run the model on both dog and human images, we need both canine and human datasets.

The provided data sets consist of colored images.

- 13223 (61%) are humans including different skin tones
- 8351 (39%) are dogs including different fur colors

The dimensions of the images are of varying size and include both square and non-square ones.

Additionally, there is a data dictionary containing the class index and dog breed types here.



Sample images

Algorithms

Object identification

Before classifying dog breeds, we can use CV2's HAAS Cascade Filter to detect human faces to tell apart whether the given image contains a human or a dog.

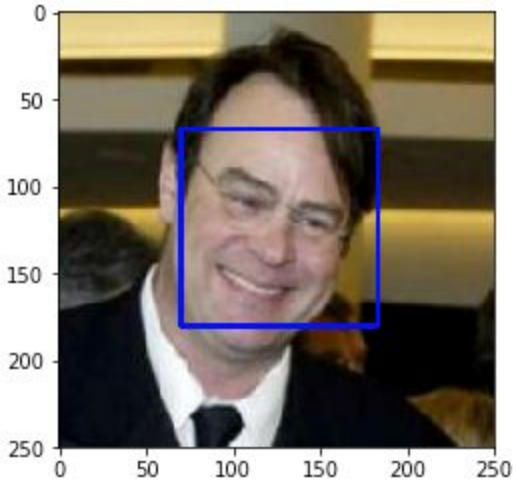


Image classification

Various pre-trained models of varying sizes and accuracies have seen promising applications in image classification.

Model	Size	Top-1 Accuracy	Top-5 Accuracy	Parameters	Depth
Xception	88 MB	0.790	0.945	22,910,480	126
VGG16	528 MB	0.713	0.901	138,357,544	23
VGG19	549 MB	0.713	0.900	143,667,240	26
ResNet50	98 MB	0.749	0.921	25,636,712	-
ResNet101	171 MB	0.764	0.928	44,707,176	-
ResNet152	232 MB	0.766	0.931	60,419,944	-
ResNet50V2	98 MB	0.760	0.930	25,613,800	-
ResNet101V2	171 MB	0.772	0.938	44,675,560	-
ResNet152V2	232 MB	0.780	0.942	60,380,648	-
InceptionV3	92 MB	0.779	0.937	23,851,784	159
InceptionResNetV2	215 MB	0.803	0.953	55,873,736	572
MobileNet	16 MB	0.704	0.895	4,253,864	88
MobileNetV2	14 MB	0.713	0.901	3,538,984	88
DenseNet121	33 MB	0.750	0.923	8,062,504	121
DenseNet169	57 MB	0.762	0.932	14,307,880	169
DenseNet201	80 MB	0.773	0.936	20,242,984	201
NASNetMobile	23 MB	0.744	0.919	5,326,716	-
NASNetLarge	343 MB	0.825	0.960	88,949,818	-

The top-1 and top-5 accuracy refers to the model's performance on the ImageNet validation dataset.

These high-performing models can be further generalized when combined with transfer learning. We can leave lower layers of the convolution neural network frozen and replace the last layers of classifier to customize for dog breed classification. Different approaches include creating a few fully-connected linear layers with drop-out, or just one linear layer with outputs equalling the number of classes. Here for simplicity and speed I've chosen the latter.

Then we make predictions based on the highest class probability returned from the softmax. We can measure the accuracy and loss to monitor the model performance. Afterward we save the model and weights to reproduce the result on new dog images.

Benchmark

To establish the baseline, I used a CNN from scratch to predict the dog breed and measured its accuracy.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.cnn_layers = nn.Sequential(
```

```
            # Defining a 2D convolution layer
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.Conv2d(32, 32, kernel_size=3, stride=1, padding=1),
            nn.BatchNorm2d(32),
            nn.ReLU(inplace=True),
            nn.MaxPool2d(kernel_size=2, stride=2),
        )

        self.linear_layers = nn.Sequential(
            nn.Dropout(p = 0.5),
            nn.Linear(401408, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(inplace=True),
            nn.Dropout(p = 0.5),
            nn.Linear(512, 133),
        )

    def forward(self, x):
        x = self.cnn_layers(x)
        #print('shape after cnn',x.shape)
        x = x.view(x.size(0), -1)
        #print('after reshape',x.shape)
        x = self.linear_layers(x)
        return F.log_softmax(x)
        return x
```

This model turned out to be time-consuming with low accuracy of 0.12, with a lot of potential for improvements.

Methodology

Data Preprocessing

1. Resizing: the images are resized to (224,224) as per ImageNet standard
2. Given the ~8000 dog images and 133 classes, there are only 60 images per breed and the images are taken in various environmental backgrounds. For more robust results. Image augmentation can be an effective way to enrich the data. Common techniques include rotation, flipping, shifting, blurring. In this project I mainly did horizontal flipping on training images.
3. Normalization is applied to all tensorized images

4. Prepare the data for modeling by splitting into training, testing, and validation sets. The validation set will be used to monitor the loss during training rounds.

Implementation

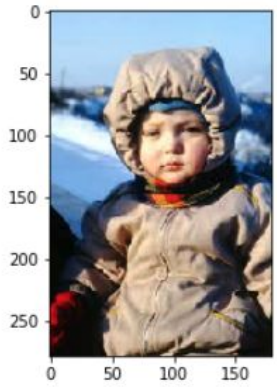
For the actual model, I used VGG16 with the last linear layer replaced to train for dog breed recognition.

```
model_transfer = models.vgg16(pretrained=True)

for param in model_transfer.features.parameters():
    param.requires_grad = False


n_classes = 133
n_inputs = model_transfer.classifier[6].in_features
last_layer = nn.Linear(n_inputs, n_classes)

model_transfer.classifier[6] = last_layer


if use_cuda:
    model_transfer = model_transfer.cuda()
```

Parameter choices

Since it is a multi-class problem, I used CrossEntropyLoss as the loss function. For the optimizer I used SGD with momentum.

This dog is probably a Pomeranian



You look like a American foxhound

Refinement

At first I choose a larger learning rate and lower number of epochs, after observing the accuracy rate, I tuned down the learning rate to 0.05 and increased the number of epochs to obtain better accuracy

Results

The transfer learning model obtained an accuracy of 0.76 on the validation set, significantly higher than the 0.12 accuracy by CNN model from scratch.

Next steps

For future improvements, I will look into these measures as the next steps:

- Cross-validation to improve the robustness of the solution
- Tune hyperparameters such as learning rate
- Increase model depth with add another linear layer
- More image augmentation
- Consider adding other regularizations such as l2
- Model interpretation with SHAP deepExplainer

Reference

https://www.analyticsvidhya.com/blog/2019/12/image-augmentation-deep-learning-pytorch/
https://keras.io/applications/