Dog Breed Classifier

Meets Specifications

Congratulations on passing this project!👏 It was really a great experience reviewing your project.😄
This project is just intuitively teaching you how to implement a CNN architecture from scratch or by using transfer learning. If you want to engage in the research or work of Computer Vision, you should learn a lot. Besides image classification, image segmentation, object detection, and image generation are also included in Computer Vision tasks. Keep learning and don't stop!💪 Good luck with future submissions :)

Here are some resources for you to learn more about CNN and Transfer Learning:

- CS231n: Convolutional Neural Networks for Visual Recognition
- Using Convolutional Neural Networks to Classify Dog Breeds
- Building an Image Classifier
- Tips/Tricks in CNN
- Transfer Learning using Keras
- Transfer Learning in TensorFlow on the Kaggle Rainforest competition
- Transfer Learning and Fine-tuning
- Building powerful image classification models using very little data

---

References:

- [VGG16] VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION
- [Inception-v1] Going deeper with convolutions
- [Inception-v3] Rethinking the Inception Architecture for Computer Vision
- [Inception-v4] Inception-ResNet and the Impact of Residual Connections on Learning
- [ResNet] Deep Residual Learning for Image Recognition
- [Xception] Deep Learning with Depthwise Separable Convolutions

Files Submitted

	The submission includes all required, complete notebook files.
	All files are correctly submitted. Great!

Step 1: Detect Humans

	The submission returns the percentage of the first 100 images in the dog and human face datasets that include a detected, human face.

Well done!

> Percentage of human images with detected faces: 99.00%
> Percentage of dog images with detected faces: 0.00%

You can refactor your code in this way to make it more concise:

```
def detect(detector, files):
    # np.mean(list(map(detector, files))) # Or this method
    return np.mean([detector(f) for f in files])
print('human: {:.2%}'.format(detect(face_detector, human_files_short)))
print('dog: {:.2%}'.format(detect(face_detector, dog_files_short)))
```

Step 2: Detect Dogs

Use a pre-trained VGG16 Net to find the predicted class for a given image. Use this to complete a `dog_detector` function below that returns True if a dog is detected in an image (and False if not).

Good job!👏

You can refactor your code in this way to make it more concise:

```
def dog_detector(img_path):
    idx = VGG16_predict(img_path)
    return 151 <= idx <= 268
```

The submission returns the percentage of the first 100 images in the dog and human face datasets that include a detected dog.

> Percentage of human images with detected dogs: 1.00%
> Percentage of dog images with detected dogs: 95.00%

You can improve your code in this way to make it more concise:

```
print('human: {:.2%}'.format(detect(dog_detector, human_files_short)))
print('dog: {:.2%}'.format(detect(dog_detector, dog_files_short)))
```

Step 3: Create a CNN to Classify Dog Breeds (from Scratch)

Write three separate data loaders for the training, validation, and test datasets of dog images. These images should be pre-processed to be of the correct size.

Good job in implementing the procedure for preprocessing the data!

Answer describes how the images were pre-processed and/or augmented.

Great answer!

The submission specifies a CNN architecture.

Good job in implementing your CNN architecture!

Adding some dropout layers to reduce the risk of overfitting is a very sensible decision. Further, you can use batch_normalization[Ref] layers to avoid covariate shift and accelerate the training process, which is very popular in recent CNN architectures. Check this video to learn how Batch Normalization works.

```
class Net(nn.Module):
    ### TODO: choose an architecture, and complete the class
    def __init__(self):
        super(Net, self).__init__()
        ## Define layers of a CNN

        self.conv1 = nn.Conv2d(3, 64, 3, padding=1)
        self.conv2 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv3 = nn.Conv2d(128, 256, 3, padding=1)
        self.conv4 = nn.Conv2d(256, 512, 3, padding=1)
        self.conv5 = nn.Conv2d(512, 1024, 3, padding=1)
        self.conv6 = nn.Conv2d(1024, 2048, 3, padding=1)
        self.conv7 = nn.Conv2d(2048, 4096, 3, padding=1)

        self.bn1 = nn.BatchNorm2d(64)
        self.bn2 = nn.BatchNorm2d(128)
        self.bn3 = nn.BatchNorm2d(256)
        self.bn4 = nn.BatchNorm2d(512)
        self.bn5 = nn.BatchNorm2d(1024)
        self.bn6 = nn.BatchNorm2d(2048)
        self.bn7 = nn.BatchNorm2d(4096)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(4096, 2048)
        self.fc2 = nn.Linear(2048, 133)

        self.dropout = nn.Dropout(0.5)


    def forward(self, x):
        ## Define forward behavior
        x = self.bn1(self.pool(F.relu(self.conv1(x))))
        x = self.bn2(self.pool(F.relu(self.conv2(x))))
        x = self.bn3(self.pool(F.relu(self.conv3(x))))
        x = self.bn4(self.pool(F.relu(self.conv4(x))))
        x = self.bn5(self.pool(F.relu(self.conv5(x))))
        x = self.bn6(self.pool(F.relu(self.conv6(x))))
        x = self.bn7(self.pool(F.relu(self.conv7(x))))

        x = x.view(x.size(0), -1)
        x = F.relu(self.fc1(x))
        x = self.dropout(x)

        x = self.fc2(x)

        return x
```

Answer describes the reasoning behind the selection of layer types.

Good job in explaining your CNN architecture!

Choose appropriate loss and optimization functions for this classification task. Train the model for a number of epochs and save the "best" result.

The trained model attains at least 10% accuracy on the test set.

Perfect! Superb work on obtaining 24%. 👏

Step 4: Create a CNN Using Transfer Learning

The submission specifies a model architecture that uses part of a pre-trained model.

Well done for choosing VGG16!

Further, you can try different models, such as VGG19, Inception, RestNet, or Xception, then compare their performances, strengths, and weaknesses. This will not take lots of your time.

The pre-trained model itself has done most of the heavy work, so such a simple model structure is a good choice.

The submission details why the chosen architecture is suitable for this classification task.

Good job in explaining your reasoning for the chosen architecture. 👏

Further resources:

- ImageNet: VGGNet, ResNet, Inception, and Xception with Keras
- ResNet, AlexNet, VGGNet, Inception: Understanding various architectures of Convolutional Networks
- Systematic evaluation of CNN advances on the ImageNet

Train your model for a number of epochs and save the result wth the lowest validation loss.

Great work! Both loss function and optimizer are specified.😄

I recommend you to try Adam [Ref] or Adagrad[Ref] as the optimizer. For more details, here is An overview of gradient descent optimization algorithms to learn more about the strength and weaknesses of the most popular optimization algorithms.

Accuracy on the test set is 60% or greater.

Well done! Test accuracy is 82%. 👏

The submission includes a function that takes a file path to an image as input and returns the dog breed that is predicted by the CNN.

Step 5: Write Your Algorithm

The submission uses the CNN from the previous step to detect dog breed. The submission has different output for each detected image type (dog, human, other) and provides either predicted actual (or resembling) dog breed.

Great work on implementing your own Dog Breed Classification Application!👏

Step 6: Test Your Algorithm

The submission tests at least 6 images, including at least two human and two dog images.

Good job in the implementation and testing!👏

Submission provides at least three possible points of improvement for the classification algorithm.

Great analysis of the performance and improvements!

Here are my suggestions for improvement, hope useful:

1. **Model ensembling.**
   Use technologies of model ensembling, such as voting, blending, bagging and stacking, to improve the accuracy.
2. **More data.**
   Use data augmentation or GAN(Generative Adversarial Networks) to obtain more data. Also, data augmentation can reduce the risk of overfitting.
3. **Change algorithm of human detection.**
   Although OpenCV module is very powerful and efficient, it can also make many mistakes such as consider a cat as a human. So, maybe we should change an algorithm.
4. **Multi-object detection.**
   Use an algorithm like RCNN or Fast-RCNN to detect and predict the breed of multi-object in an image simultaneously.

DOWNLOAD PROJECT

RETURN TO PATH