UDACITY

Return to Classroom

Dog Breed Classifier

REVIEW

CODE REVIEW

HISTORY

Meets Specifications

Overall, your understanding of the project was quite brilliant. Just a few points though,
1) Try going deeper and wider i.e try going deeper by using more layers and wider by using more filters. You have gone 512 filters wide. Try something like a Resnet 50 with 1024 filters. Generally a depth/width ratio also becomes a hyperparameter. But then there is a trade off between speed and accuracy. Like Resnet 152 is good but one thing you will learn is that how fast will such a network be during test time i.e when used in real time on an embedded system?

2) One thing that I recently learnt and I haven't seen many people working on it is that after having trained your network using SGD and achieving local minima, try using ADAM on the learnt weights and try reaching a global minima.

3) I think you should have used more epochs as the losses were constantly going down and you would have achieved better accuracy for the CNN you built from scratch. Moreover, for learning, you can try different type of convolutions and see what output you get. For example depth wise separable convs, spatial and cross channel convs. For a very basic idea, check this article https://ikhlestov.github.io/pages/machine-learning/convolutions-types/

For a deeper understanding https://www.analyticsvidhya.com/blog/2018/12/guide-convolutional-neural-network-cnn/

You should also try batch normalization. You can read about batch normalization here - https://arxiv.org/pdf/1502.03167.pdf. Though there have been many debates and research about batch norm but I still feel that this paper forms the foundation for understanding further debates about batch norm.

You can also try object detection on images. Read about the same here https://machinelearningmastery.com/object-recognition-with-deep-learning/

Rest all seems decent. Good work. Just answering the last question will let you come out with flying colors

I wish you good luck. Happy learning

Files Submitted

The submission includes all required, complete notebook files.

All files present as per requirement

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. You can read more about Haar cascades - http://www.willberger.org/cascade-haar-explained/
Also you can look at using MTCNN instead of haarcascades.
Also the following blog has very good insights at implementing face detection with multiple algorithms -> https://www.pyimagesearch.com/2018/09/24/opencv-face-recognition/

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).

While previous derivatives of AlexNet focused on smaller window sizes and strides in the first convolutional layer, VGG addresses another very important aspect of CNNs: depth. You can read more about VGG in this paper- https://arxiv.org/abs/1409.1556 . It is an easy paper to understand and will give you a good perspective into the architecture

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

Great work. Your coding skills are good

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.

Great work.
You can avoid shuffling validation and test set as every time we would train again, the validation and test accuracies would differ as the batches would differ so to benchmark things it is preferable to not shuffle.
Also pytorch's transform resize behaves differently when you pass only integer so keep that in mind
https://pytorch.org/docs/stable/torchvision/transforms.html#torchvision.transforms.Resize

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

Good intuition. Adding gaussian noise is something to try. Check -> https://discuss.pytorch.org/t/how-to-add-noise-to-mnist-dataset-when-using-pytorch/59745/7 . Also you can try adding shadow in the images and try training the network from there
This is also a worthy article to read about data augmentation - https://www.analyticsvidhya.com/blog/2019/12/image-

The submission specifies a CNN architecture.

Well done. This is some very nice work by you. Some links that will help in your journey to improve the overall way you look at CNN

- A Beginner's Guide To Understanding Convolutional Neural Networks - https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
- A Beginner's Guide To Understanding Convolutional Neural Networks Part 2 https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/
- Building powerful image classification models using very little data https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
- Cats and dogs and convolutional neural networks http://www.subsubroutine.com/sub-subroutine/2016/9/30/cats-and-dogs-and-convolutional-neural-network

Answer describes the reasoning behind the selection of layer types.

Nice work.

- Converting the incoming image into feature map of depth 64 and thereby using maxpooling to reduce the overall size of the feature map is a good step. Also it will only take the neurons that had the maximum activation within the window size you selected. Max pooling is done to in part to help over-fitting by providing an abstracted form of the representation. As well, it reduces the computational cost by reducing the number of parameters to learn and provides basic translation

invariance to the internal representation. Max pooling is done by applying a max filter to (usually) non-overlapping subregions of the initial representation. The other forms of pooling are: average, general. Overall a good starting to building a CNN. More about Maxpooling can be checked out here https://analyticsindiamag.com/max-pooling-in-convolutional-neural-network-and-its-features/

- Flattening the layers to 4096 is also a good step. Will given enough features to play with for proper classification
- ReLU activation is most logical step. Take a look at this article on kaggle which explains why ReLU works well and what changes can be made to the already existing function. https://www.kaggle.com/dansbecker/rectified-linear-units-relu-in-deep-learning

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

Good work. You can also look at the following article to know more about loss functions https://machinelearningmastery.com/loss-and-loss-functions-for-training-deep-learning-neural-networks/. Some common loss functions are

- Mean Squared Error (MSE) - MSE loss is used for regression tasks. As the name suggests, this loss is calculated by taking the mean of squared differences between actual(target) and predicted values.
- Binary Crossentropy (BCE) - BCE loss is used for the binary classification tasks. If you are using BCE loss function, you just need one output node to classify the data into two classes. The output value should be passed through a sigmoid activation function and the range of output is (0 – 1).
- Categorical Crossentropy (CC) - When we have a multi-class classification task, one of the loss function you can go ahead is this one. If you are using CCE loss function, there must be the same number of output nodes as the classes. And the final layer output should be passed through a softmax activation so that each node output a probability value between (0–1).
- Sparse Categorical Crossentropy (SCC) - When we are using SCCE loss function, you do not need to one hot encode the target vector. If the target image is of a cat, you simply pass 0, otherwise 1. Basically, whichever the class is you just pass the index of that class.
- For more loss functions, check https://www.analyticsvidhya.com/blog/2019/08/detailed-guide-7-loss-functions-machine-learning-python-code/

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

Well done. Try to incorporate more layers for better accuracy. Also , the following will help

- Tune Parameters - To improve CNN model performance, we can tune parameters like epochs, learning rate etc..
- Image Data Augmentation - Image augmentation parameters that are generally used to increase the data sample count are zoom, shear, rotation, preprocessing function and so on. Usage of these parameters results in generation of images having these attributes during training of Deep Learning model.
- Deeper Network Topology - A wide neural network is possible to train with every possible input value. Hence, these networks are very good at good at memorization, but not so good at generalization. There are, however, a few difficulties with using an extremely wide, shallow network. Though, wide neural network is able to accept every possible input value, in the practical application we won't have every possible value for training

we won't have every possible value for training.

- Deeper networks capture the natural "hierarchy" that is present everywhere in nature. See a convnet for example, it captures low level features in first layer, a little better but still low level features in the next layer and at higher layers object parts and simple structures are captured. The advantage of multiple layers is that they can learn features at various levels of abstraction.

- Handle Overfitting and Underfitting problem - Overfitting refers to a model that models the training data too well. What is the meaning of it. Lets simplify... In the overfitting your model gives very nice accuracy on trained data but very less accuracy on test data. The meaning of this is overfitting model is having good memorization ability but less generalization ability. Our model doesn't generalize well from our training data to unseen data.
  Underfitting refers to a model which works unwell on train as well as test data. Its very dangerous. Isn't it? Model is not fitted well on training data itself.

Step 4: Create a CNN Using Transfer Learning

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

Great work.
VGG Neural Networks. While previous derivatives of AlexNet focused on smaller window sizes and strides in the first convolutional layer, VGG addresses another very important aspect of CNNs: depth. Let's go over the architecture of VGG:

Input. VGG takes in a 224x224 pixel RGB image. For the ImageNet competition, the authors cropped out the center 224x224 patch in each image to keep the input image size consistent.
Convolutional Layers. The convolutional layers in VGG use a very small receptive field (3x3, the smallest possible size that still captures left/right and up/down). There are also 1x1 convolution filters which act as a linear transformation of the input, which is followed by a ReLU unit. The convolution stride is fixed to 1 pixel so that the spatial resolution is preserved after convolution.
Fully-Connected Layers. VGG has three fully-connected layers: the first two have 4096 channels each and the third has 1000 channels, 1 for each class.
Hidden Layers. All of VGG's hidden layers use ReLU (a huge innovation from AlexNet that cut training time). VGG does not generally use Local Response Normalization (LRN), as LRN increases memory consumption and training time with no particular increase in accuracy.
The Difference. VGG, while based off of AlexNet, has several differences that separates it from other competing models:
Instead of using large receptive fields like AlexNet (11x11 with a stride of 4), VGG uses very small receptive fields (3x3 with a stride of 1). Because there are now three ReLU units instead of just one, the decision function is more discriminative. There are also fewer parameters (27 times the number of channels instead of AlexNet's 49 times the number of channels).

VGG incorporates 1x1 convolutional layers to make the decision function more non-linear without changing the receptive fields.
The small-size convolution filters allows VGG to have a large number of weight layers; of course, more layers leads to improved performance. This isn't an uncommon feature, though. GoogLeNet, another model that uses deep CNNs and small convolution filters, was also showed up in the 2014 ImageNet competition.

Some transfer learning resources https://www.pyimagesearch.com/2017/03/20/imagenet-vggnet-resnet-inception-xception-keras/

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

Good work. This is a stepping stone for further indepth understanding on the basis of the explanation above

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

Accuracy on the test set is 60% or greater.

Great work. Try other networks like Densenet, Inception, Xception etc for comparison,

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.

Nice work here

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.

Step 6: Test Your Algorithm

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

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

Great intuition. Some more point will help

- You can think in terms of class imbalance and lack of data for some classes causing the model to be biased to some breeds. - Methods for addressing class imbalance can be divided into two main categories. The first category is data level methods that operate on training set. The other category covers classifier (algorithmic) level methods, which keeps the training dataset unchanged and adjust

training or inference algorithms

- Data level methods
  a. Oversampling
  b. Undersampling

- Classifier level methods
  a. Thresholding
  b. Cost sensitive learning
  c. One-class classification
  d. Hybrid of methods
- You can also discuss the possibility of detecting multiple faces and accuracy of face detector as we have cnn based face detectors performing well now.

- For CNN architecture, you can discuss how you can improve your accuracy by introducing weights initialization and more sophisticated network in your scratch model to improve its accuracy.

DOWNLOAD PROJECT

RETURN TO PATH