Thanks for all the reviewers, thank for your time and patient. We have some comments on reviewer 1:

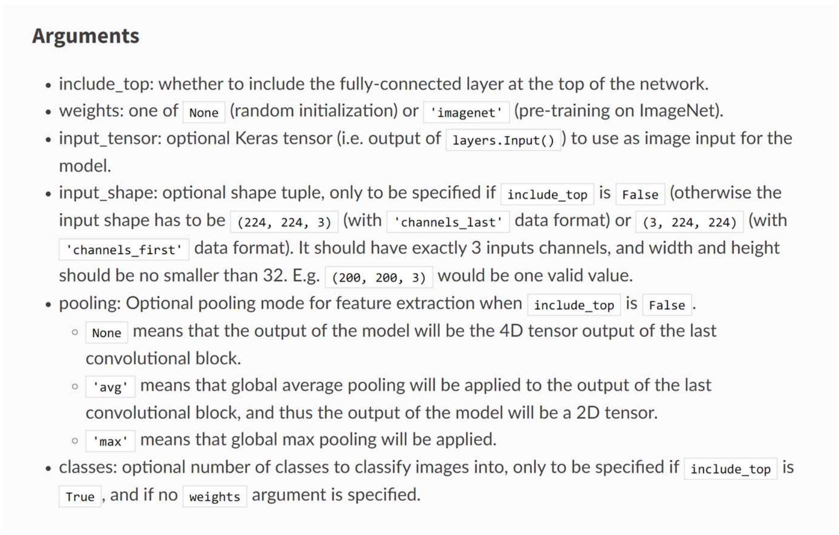
Thanks for your review opinion, I think this is the earnest review we have ever received. We appreciate your time and effort.
However, there are some technical points we want to clarify:

1.While loading MNIST dataset, it is not exactly clear if the input dataset is being correctly normalized and resized before feeding into pretrained networks. In the file load_data.py, in the function get_dataloader(), we see that the normalization used on the dataset is (0.5,), (0.25,). However, for pre-trained networks trained on ImageNet, there is a fixed set of summary statistics that should be used to normalize the input data. Also, the input images should be resized at 224 x 224 x 3 which is the size of the ImageNet dataset images.

First, the load_data.py is just for identifying the painting of Raphael using Scattering Net. It has nothing to do with using pretrained CNNs. For the MNIST dataset using CNNs, we use Keras and carefully followed the document of it. And in the official document, it says that if you remove the top classification layer (fully connected layer), any input with width and height no smaller than 32 is a valid input. Why we do not need to reshape the figure to the original "imagnet size" (224*224*3)? Because we have remove the top fully connected layers, and only convolutional layers left for feature extraction. And convolutional layer can receive not only "imagnet size". If we using smaller size, we just get a output feature map with smaller size. Considering that the Mnist dataset is too simple, we just choose to reshape it to a smaller size to make the extracted feature with a lower dimension, it is more computational effective.

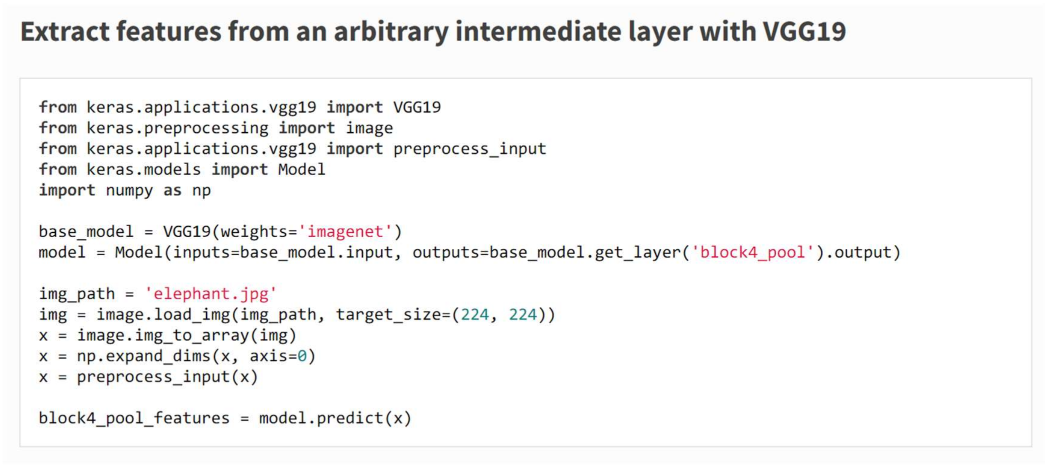
As for the normalization issue, we can show that the Keras official example also do not show the process explicitly. Note that Keras offers a high-level API, this process is hidden for user friendliness.

Screenshots of Official Documents:



Arguments

- include_top: whether to include the fully-connected layer at the top of the network.
- weights: one of `None` (random initialization) or `'imagenet'` (pre-training on ImageNet).
- input_tensor: optional Keras tensor (i.e. output of `layers.Input()`) to use as image input for the model.
- input_shape: optional shape tuple, only to be specified if `include_top` is `False` (otherwise the input shape has to be `(224, 224, 3)` (with `'channels_last'` data format) or `(3, 224, 224)` (with `'channels_first'` data format). It should have exactly 3 inputs channels, and width and height should be no smaller than 32. E.g. `(200, 200, 3)` would be one valid value.
- pooling: Optional pooling mode for feature extraction when `include_top` is `False`.
  - `None` means that the output of the model will be the 4D tensor output of the last convolutional block.
  - `'avg'` means that global average pooling will be applied to the output of the last convolutional block, and thus the output of the model will be a 2D tensor.
  - `'max'` means that global max pooling will be applied.
- classes: optional number of classes to classify images into, only to be specified if `include_top` is `True`, and if no `weights` argument is specified.



Extract features from an arbitrary intermediate layer with VGG19

```
from keras.applications.vgg19 import VGG19
from keras.preprocessing import image
from keras.applications.vgg19 import preprocess_input
from keras.models import Model
import numpy as np

base_model = VGG19(weights='imagenet')
model = Model(inputs=base_model.input, outputs=base_model.get_layer('block4_pool').output)

img_path = 'elephant.jpg'
img = image.load_img(img_path, target_size=(224, 224))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
x = preprocess_input(x)

block4_pool_features = model.predict(x)
```

2.In section 2.6, paragraph 1, the authors try to provide an intuitive explanation of why random forests do not perform well on the classification problem with extracted features. The authors, in my opinion, make a few inaccurate statements: – "Note that random forest randomly selects on pixel (variable) of the image as its filter". This should instead say that random forest selects a subset of features from an input feature vector/tensor. It is true that one can make it such that at each iteration the random forest algorithm will choose just 1 feature, but that is not apparent in the implementation in classification.py. – Also, it is not true that once an "important" feature is not chosen while growing a particular random tree, that would greatly affect the model as a 2 whole. While there would be some effect, the idea of a random forest is to grow a large number of random trees with a large amount of variance across those trees. Presumably, at least one random tree in the forest will be able to capture the pattern. – "But other classification methods would somehow take the weights of pixels, which measure the impact of the pixels, into consideration". It is an unsubstantiated claim that a random forest will not be able to do this. A random forest classifier would implicitly assign weights to certain features by the number of random trees that vote in the favor of the output (in a classification setting, which is what we are talking about here)

Our first statement is inaccurate, because the default setting of random forest is select sqrt(total feature). The reviewer tries to claim that the sparsity would not affect the performance of random forest. However, it is not that case. We further design a simple experiment to show that sparsity would affect the performance of default random forest which means select sqrt(total features). And random forest is more vulnerable comparing to other methods.

We choose IRIS dataset and do classification task. we first use the original data then add many noise features to the data, which would make the useful features sparse.

The results is the three fold cross validation accuracy. We can see that all the classifier could get good results on the original dataset because the task is relative simple. However, if we add many noise features, they all have a significant accuracy drop. And the default random forest model performs worst. That is because the model consider 101 features at most. It is hard to have a sub tree consider all the useful features. Also, most of the sub trees would use noise to overfit the train data. At the same time, logistic regression would consider all the features and find the useful ones. Especially, L1 penalty is designed for solving sparse problems, it performs well. The above results show that the random forest with default setting would fail when the feature is sparse. And that is the case we discuss in the report.

Then, we modify the setting of random forest to make it use more features. We set the max number feature to 0.1*(total features) and the result is much better. That means the reviewer's assumption is correct when there are more selected features. We also try this setting on Mnist with ResNet as feature extractor. It improves accuracy from 0.9337 to 0.9435.

Above analysis want to show that sparse feature is one of the reasons why default random forest performs worse than other algorithms, though, it might have other reasons like random forest is suitable for tabular data and axis-aligned data. It is not fair to say our statement is unsubstantiated for default random forest. And thanks for your opinion, we can improve our results.

Classification results on Iris data

Feature dimension	Logistic regression with l2 penalty	Logistic regression with l1 penalty	Default random forest	Random forest select more features (0.1 * total features)
4(origin dataset)	0.9468	0.9334	0.9738	0.9673
4+10000(noise)	0.7271	0.8717	0.5089	0.8986