On the similarity of Gabor filters and CNN first-layer filters

October 6, 2022

1 Instructions for reproducing the experiments

The following document describes the elements necessary to replicate the results presented in the paper. In order to create solutions that are easy to reproduce, in this work we tried to use only publicly available resources: data, pretrained CNN models and tutorials from reputable sources. This way, all of the resources are described in detail and easy to understand even for younger researchers starting their scientific career. We also create a short instruction and guidelines on how to reproduce the results easily. We hope that this will make the work easier for possible reproducers of our experiments.

First of all, we use Jupyterlab notebooks and Python for our experiments. We use the following Python libraries: TensorFlow, Keras, NumPy, opencv-python and SciKit-learn. We perform all of our experiments on a computer with AMD Ryzen 7 2700X Eight-Core, 3.70 GHz and 32 GB RAM.

1.1 Data

We used the following freely available datasets:

- **Imagenette** - it is a smaller subset of Imagenet (10 classes). It is thus easier to use and requires less computational resources. It can be found here: `https://github.com/fastai/imagenette`
- **MNIST** - it contains 10 handwritten digits. The dataset can be used directly in keras (tensorflow.keras.datasets.mnist.load_data() loads the dataset - see `https://keras.io/api/datasets/mnist`). Yann LeCun and Corinna Cortes hold the copyright of MNIST dataset, which is a derivative work from original NIST datasets. MNIST dataset is made available under the terms of the Creative Commons Attribution-Share Alike 3.0 license.
- **Fashion MNIST** - it contains 10 different groups of clothing (Zalando's article images). The dataset can be used directly in keras: (tensorflow.keras.datasets.fashion_mnist.load_data() loads the dataset - see `https:`

`//keras.io/api/datasets/fashion/mnist`). The copyright for Fashion-MNIST is held by Zalando SE. Fashion-MNIST is licensed under the MIT license.

- **NIPS17 Adversarial Attacks and Defenses Competition** - the dataset contains Imagenet-compatible images that can be used to evaluate the performance of adversarial attacks and defenses. The dataset is available at `https://github.com/cleverhans-lab/cleverhans/tree/master/cleverhans_v3.1.0/examples/nips17_adversarial_competition/dataset`

1.2 Data processing

Data processing procedure is dependent on a model used. State-of-art models available at `https://keras.io/api/applications` have their own preprocessing functions:

- **InceptionV3** - tf.keras.applications.inception_v3.preprocess_input (it scales input pixels between -1 and 1)
- **Xception** - tf.keras.applications.xception.preprocess_input (it scales input pixels between -1 and 1)
- **InceptionResnetV2** - tf.keras.applications.inception_resnet_v2.preprocess_input (it scales input pixels between -1 and 1)
- **MobileNet** - tf.keras.applications.mobilenet.preprocess_input (it scales input pixels between -1 and 1)
- **ResNet50V2** - tf.keras.applications.resnet_v2.preprocess_input (it scales input pixels between -1 and 1)
- **ResNet101** - tf.keras.applications.resnet.preprocess_input (it converts the input from RGB to BGR and zero-centers each channel with respect to the ImageNet dataset)

We scaled input pixels for our small models between 0 and 1. As raw data comes with pixels between 0, 255, it just has to be divided by 255.

1.3 Models

We use the following state-of-art deep CNN models. They can be used directly from Keras. On https://keras.io/api/applications/, one can find many other pretrained CNN models, and we encourage the potential reproducers to test some other models too. We decided to use the following diverse models:

- **InceptionV3**:

  `https://keras.io/api/applications/inceptionv3/`

- **Xception**:
  `https://keras.io/api/applications/xception/`
- **InceptionResnetV2** :
  `https://keras.io/api/applications/inceptionresnetv2/`
- **MobileNet**:
  `https://keras.io/api/applications/mobilenet/`
- **ResNet50v2**:
  `https://keras.io/api/applications/resnet/#resnet50v2-function`
- **ResNet101**:
  `https://keras.io/api/applications/resnet/#resnet101-function`

We also use small classifiers and autoencoders in our experiments. We trained them by ourselves. Networks and images are small, therefore it does not require high-end computational resources.

To build an autoencoder, we used a CNN autoencoder presented in the tutorial uploaded by Francois Chollet:
`https://blog.keras.io/building-autoencoders-in-keras.html`.
We test it on MNIST and Fashion MNIST. These both datasets are compatible in terms of processing, therefore to use Fashion MNIST instead of MNIST, one has to use another import - tensorflow.keras.datasets.fashion_mnist.load_data()). We use the same training parameters as in the tutorial. We train our autoencoders for 50 epochs.

We built our small classifiers with only two convolutional layers (padding - 'same') - as an example of a shallow CNN. In Table 1, we present the network structure and activation functions used for each layer.

Table 1: Structure of a shallow CNN used in our experiments. Summary of parameters: Total params: 19,466, Trainable params: 19,466.

Layer	Filters	Activation	Parameters
Conv2D	32 x (3, 3)	ReLU	320
Max Pooling 2D	-	-	0
Conv2D	64 x (3, 3)	ReLU	18496
Global Average Pooling 2D	-	-	0
Dense	-, size: 10	Sigmoid	584

We used the following parameters for training:

- Optimizer: Adam (default optimizer settings)
- Loss: Categorical Crossentropy
- Epochs: 30

- Batch size: 128
- Split: default split from keras.datasets.mnist.load_data() or keras.datasets.fashion_mnist.load_data()

1.4 Gabor filter bank generation

To generate our Gabor filter bank, we use NumPy and opencv-python. In file Gabor_replacement_CNNs.py in this repository, we provide a function that can be used to generate such a filter bank. We also provide a function that can be used to generate

1.5 Swapping CNN filters with Gabor filters

To replace the CNN filters with Gabor filters, we first need to find the most similar Gabor filters. To get the values of original filters in Keras, one can use the function get_weights() from Keras Layers API. We use three methods for finding the most similar Gabor filter: euclidean distance, manhattan distance and cosine similarity. We use them to measure similarity between pairs of filters (CNN filter and subsequent filters from a filter bank). The third one provided us with the best results. We treat filters as numerical arrays. All of the metrics used can be found in SciKit-learn library:

```
from sklearn.metrics.pairwise import cosine_similarity,
    euclidean_distances, manhattan_distances
```

They can compare arrays of vectors (multiple samples). Therefore, each filter has to be reshaped (we used a NumPy method reshape() and then expand_dims to obtain a 2-dimensional array with 1 vector as its first element - such a structure can be used with functions used to measure similarity).

Euclidean distance and manhattan distance are distance measures, therefore their smaller values suggest larger similarity. Cosine similarity, on the other hand, takes value 1 for the same arrays and decreases with decreasing similarity of filters. Therefore, the most similar filter from the bank is the one with the largest cosine similarity value. We use these values to compute cosine similarity statistics for each network. Filters chosen with distance measures can be replaced directly. Filters with high values of cosine similarity, on the other hand, can be related via a linear transformation. Therefore, before replacing CNN filters with Gabor filters, we use scaling in this case. We multiply the Gabor filter value by a constant value - obtained as a division of 2 Frobenius norms (of an original filter and a Gabor filter) - see Section 4 in the paper for an equation.

To replace original filters with Gabor filters, one can use the set_weights() method from Keras Layers API. Some convolutional layers include a bias. We do not change it and reupload an original array with bias values along with our new weights. An original bias array can be read with the function get_weights() from Keras Layers API - along with original values of filters. We provide an implementation of a swapping with cosine similarity - see function replace_CNN_filters_with_Gabor in file Gabor_replacement_CNNs.py. The function can operate also with the random filter bank (they have th same format).

1.6 Generation of adversarial samples

To generate adversarial samples, we based our code on the following tutorial: `https://www.tensorflow.org/tutorials/generative/adversarial_fgsm`. Its aim is to generate samples with a Fast Gradient Sign Method (FGSM), however it can easily be modified to generate normalized gradient values instead of gradient signs. Instead of using the function tf.sign() in the function create_adversarial_pattern, one has to scale values of gradient to range $\langle -1, 1\rangle$. The resulting perturbation is added to an original image. Resulting adversarial sample has to be scaled to a correct range for a particular network. Samples generated in such a manner can be fed into the pre-trained state-of-art models provided by Keras.