Applied Machine Learning

CNN Tricks

UNIVERSITY OF GOTHENBURG

CHALMERS

Richard Johansson

richard.johansson@cse.gu.se

today's menu

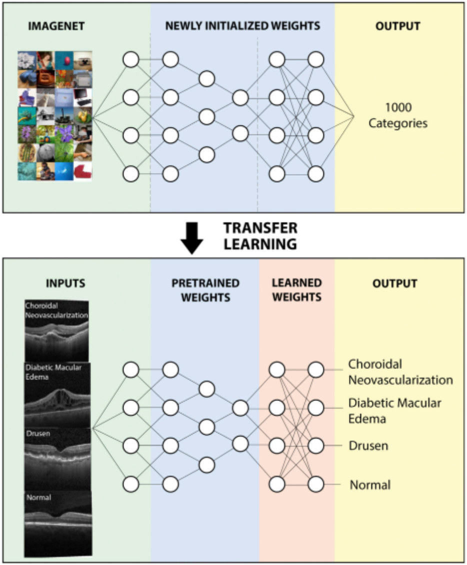
- using **pre-trained models** for **transfer learning**
- **data augmentation** for image-based ML
- **interpreting** CNNs

if we have a trained CNN, can we use it for new tasks?

transfer learning: idea and motivation

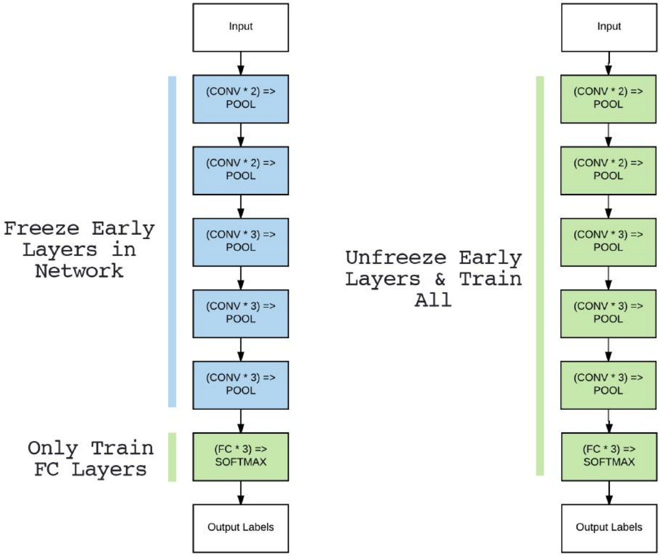
- in **transfer learning**, we try to exploit previously learned knowledge when solving new tasks
- in practice: after training, we **reuse some part** of the model
- why? because it can **reduce the need for training data** for the target task

transfer learning in vision



[source]

two high-level approaches to transfer learning in NNs



[source]

tradeoffs between freezing and fine-tuning

- if we **freeze** the pre-trained model, training is fast but there is a risk that the pre-trained part is not optimal for our task
- if we **fine-tune**, we are more flexible but risk forgetting what we learned previously: **catastrophic forgetting** (McCloskey and Cohen, 1989)
- we may explore intermediate solutions, e.g. by using lower learning rates for the pre-trained parts or adjust them in later epochs only

how can we deal with overfitting in CNNs?

using data augmentation to improve robustness

- CNNs are complex models and **overfit** easily
- it is good practice to apply one or more **regularization** techniques including L2 penalties (“weight decay”), early stopping, dropout
- for image-based ML tasks, it is very common to apply **data augmentation** to increase variation among images
  - random noising, shearing, rotating, darkening, flipping, ...
  - key assumption: output label is **unchanged** after transformation

ImageDataGenerator in Keras

tf.keras.preprocessing.image.ImageDataGenerator

See Stable | See Nightly

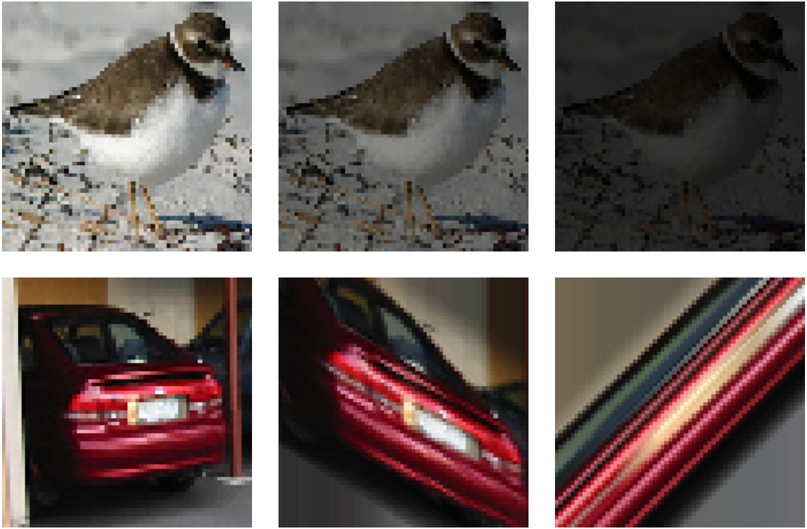
TensorFlow 1 version | View source on GitHub

Generate batches of tensor image data with real-time data augmentation.

View aliases

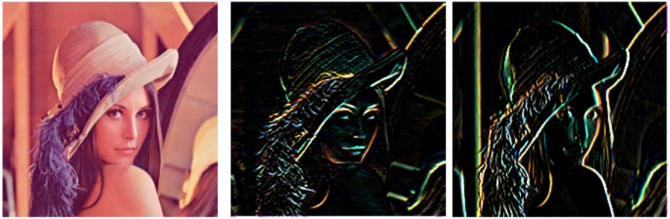
```
tf.keras.preprocessing.image.ImageDataGenerator(
    featurewise_center=False, samplewise_center=False,
    featurewise_std_normalization=False, samplewise_std_normalization=False,
    zca_whitening=False, zca_epsilon=1e-06, rotation_range=0, width_shift_range=0.0,
    height_shift_range=0.0, brightness_range=None, shear_range=0.0, zoom_range=0.0,
    channel_shift_range=0.0, fill_mode='nearest', cval=0.0,
    horizontal_flip=False, vertical_flip=False, rescale=None,
    preprocessing_function=None, data_format=None, validation_split=0.0, dtype=None
)
```

how much should we allow the examples to change?



how can we understand what a trained CNN does?

example: looking at feature maps



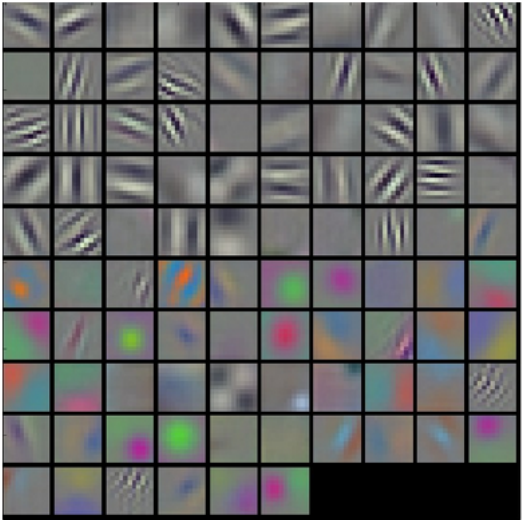
(a) Lenna (b) Horizontal edge (c) Vertical edge

Figure 4: The Lenna image and the effect of different convolution kernels.

[source]

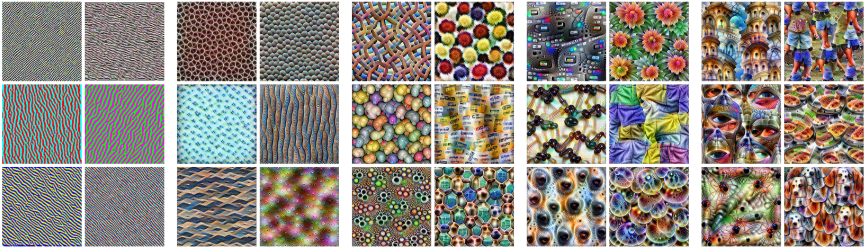
interpreting CNNs: drawing the filters

- see *Visualizing what ConvNets learn*



interpreting CNNs (2): generating images to optimize a part of the model

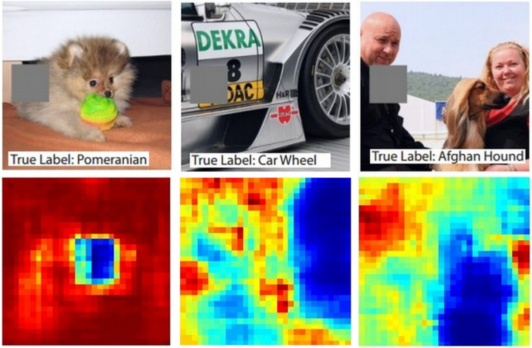
- see *Feature Visualization*



Edges (layer conv2d0) **Textures** (layer mixed3a) **Patterns** (layer mixed4a) **Parts** (layers mixed4b & mixed4c) **Objects** (layers mixed4d & mixed4e)

interpreting CNNs (3): occluding

- see *Visualizing what ConvNets learn*



summary

- **transfer learning** where we reuse **pre-trained models** may reduce the need for training data
- mitigating overfitting by applying **data augmentation** techniques: random image modifications
- there are several techniques to **interpret** CNNs by visualization

references

McCloskey, M. and Cohen, N. (1989). Catastrophic interference in connectionist networks: The sequential learning problem. *The Psychology of Learning and Motivation*, 24:109–165.