Interpretable Deep Learning

Deep Learning Methods Seminar

Balint Tamasi

06/05/2020

Interpretable deep learning

In general, deep learning models are regarded as black box models.

In the case of **convolutional neural networks**, the learned representations capture visual concepts thus they can be visualized and interpreted by humans.

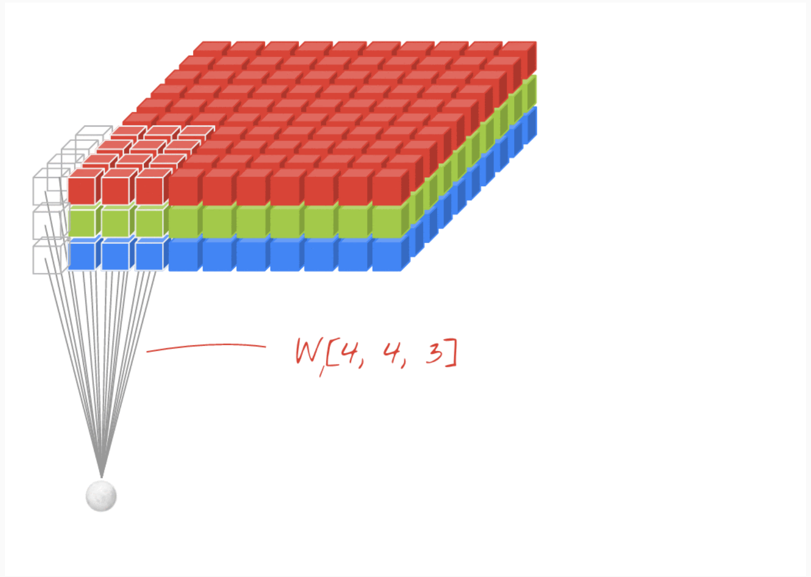
There are several ways of doing this, most of them require considerable amount of tinkering with keras/tensorflow.

This presentation focuses on three main methods in the context of convolutional neural networks and image classification, and demonstrates them through simple examples using R and Keras.

Some of the ideas can also be adapted to other deep learning models and tasks.

Recap: Convnets

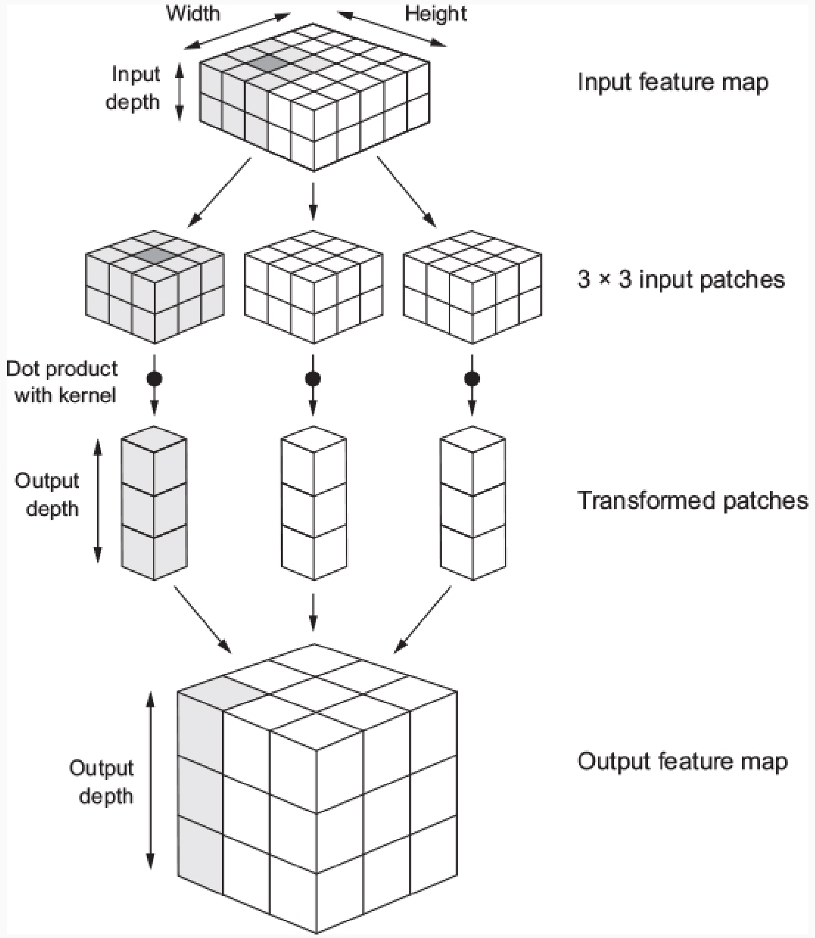
Filters



Sources:

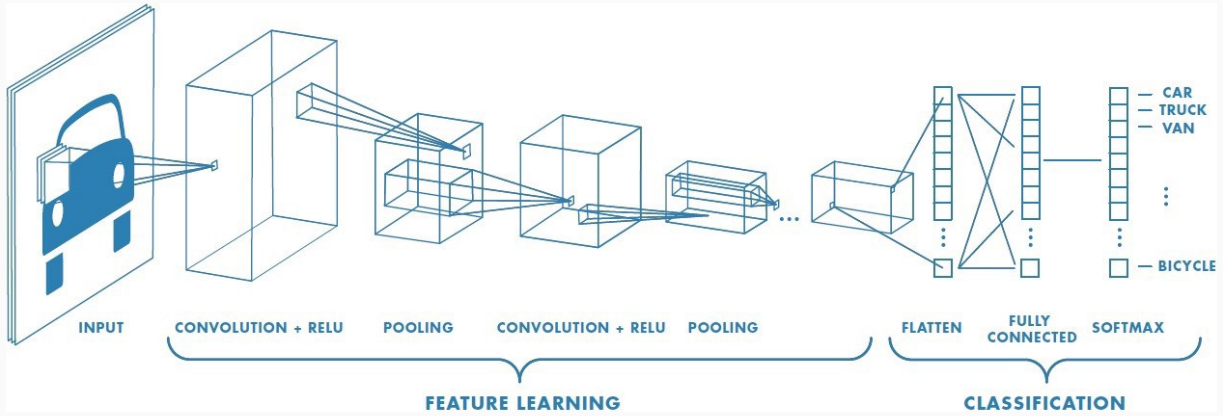
- https://codelabs.developers.google.com/codelabs/keras-flowers-tpu/#7
- https://livebook.manning.com/book/deep-learning-with-r/chapter-5/41

Convolutional layer



Recap: Convnets (cont'd)

Network architecture



Source: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Visualizing intermediate activations

We can feed an image into the model and visualize the activations at different layers (and channels).

This is a straightforward method and it can help to understand what patterns the neural net identifies in a given input.

- On lower layers filters detect edges and simple patterns. The activations retain most of the information present in the input.
- On higher layers, the encoded patterns are usually more complicated and abstract. The activations become sparser.

The disadvantage is that the output is very large (one activation image for each channel), and, especially on higher levels, it becomes harder to discern (and interpret) what the neural network detects.

Example: Visualizing intermediate activations

```
## Model: "sequential"
## ________________________________________________________________________________
## Layer (type)                        Output Shape                    Param #
## ================================================================================
## conv2d (Conv2D)                     (None, 148, 148, 32)            896
## ________________________________________________________________________________
## max_pooling2d (MaxPooling2D)        (None, 74, 74, 32)              0
## ________________________________________________________________________________
## conv2d_1 (Conv2D)                   (None, 72, 72, 64)              18496
## ________________________________________________________________________________
## max_pooling2d_1 (MaxPooling2D)      (None, 36, 36, 64)              0
## ________________________________________________________________________________
## conv2d_2 (Conv2D)                   (None, 34, 34, 128)             73856
## ________________________________________________________________________________
## max_pooling2d_2 (MaxPooling2D)      (None, 17, 17, 128)             0
## ________________________________________________________________________________
## conv2d_3 (Conv2D)                   (None, 15, 15, 128)             147584
## ________________________________________________________________________________
## max_pooling2d_3 (MaxPooling2D)      (None, 7, 7, 128)               0
## ________________________________________________________________________________
## flatten (Flatten)                   (None, 6272)                    0
## ________________________________________________________________________________
## dropout (Dropout)                   (None, 6272)                    0
## ________________________________________________________________________________
## dense (Dense)                       (None, 512)                     3211776
## ________________________________________________________________________________
## dense_1 (Dense)                     (None, 1)                       513
## ================================================================================
## Total params: 3,453,121
## Trainable params: 3,453,121
## Non-trainable params: 0
## ________________________________________________________________________________
```

Classification of images of cats and dogs using a moderate-size convolutional network.

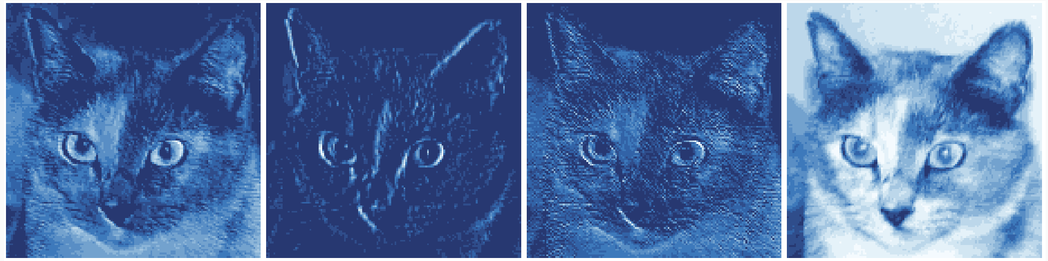
The network is trained on a few thousand images of cats and dogs using data-augmentation and dropout.

We are plotting the activations on the various convolutional layers (interesting channels selected).

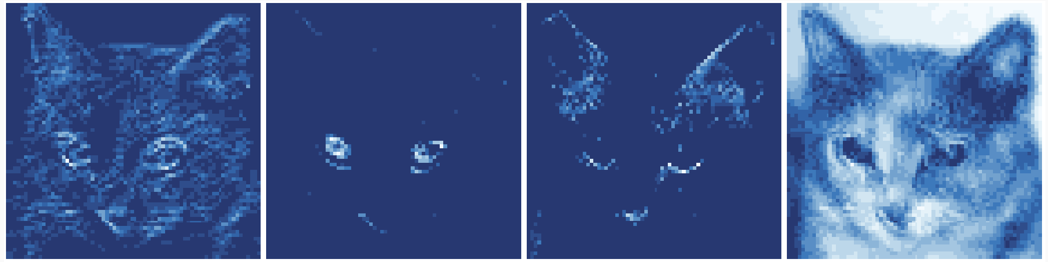
Example: Visualizing intermediate activations (cont'd)

Input

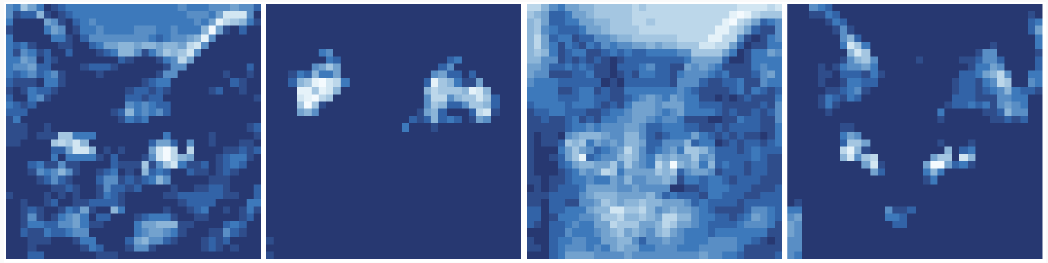
Convolutional layer 1 (examples)



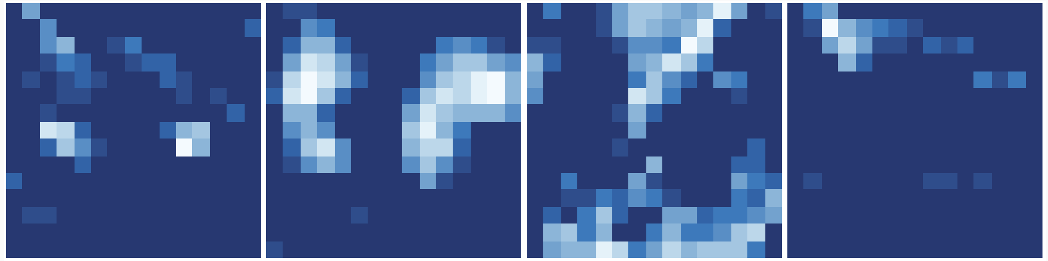
Convolutional layer 2 (examples)



Convolutional layer 3 (examples)



Convolutional layer 4 (examples)



Feature visualization

The goal of **feature visualization** is to make the features learned by the neural network explicit (without supplying a specific input). This can be done by generating an input that maximizes the activation of a specified **unit**.

A 'unit' can be a neuron, a channel, a whole convolution layer (e.g. in DeepDream), a class probability neuron, etc.

Optimization problem: starting with a 'blank' input image, find the image that maximizes the activation on a unit (e.g. a channel)

$$\text{img}^\star = \arg\max_{\text{img}} \sum_{x,y} h_{n,x,y,z}(\text{img})$$

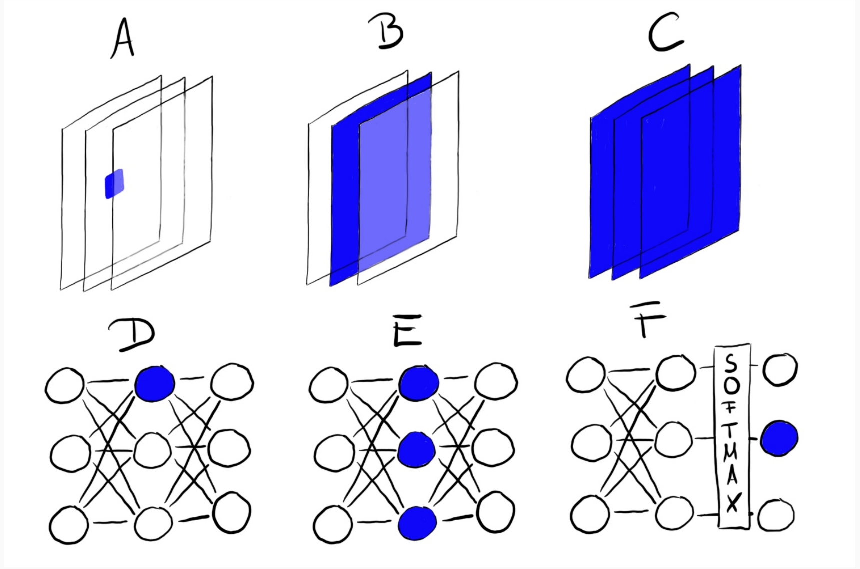


Image source: https://christophm.github.io/interpretable-ml-book/cnn-features.html#feature-visualization

Example: Visualizing convnet filters

We are using a deep neural network called VGG16, trained on the ImageNet database (millions of images, many categories).

```r
model <- application_vgg16(weights = "imagenet", include_top = FALSE)
summary(model)
```

Using only the convolutional layers of the network to generate images that maximize the activations on various (hand-picked) channels of different layers by solving the maximization problem.

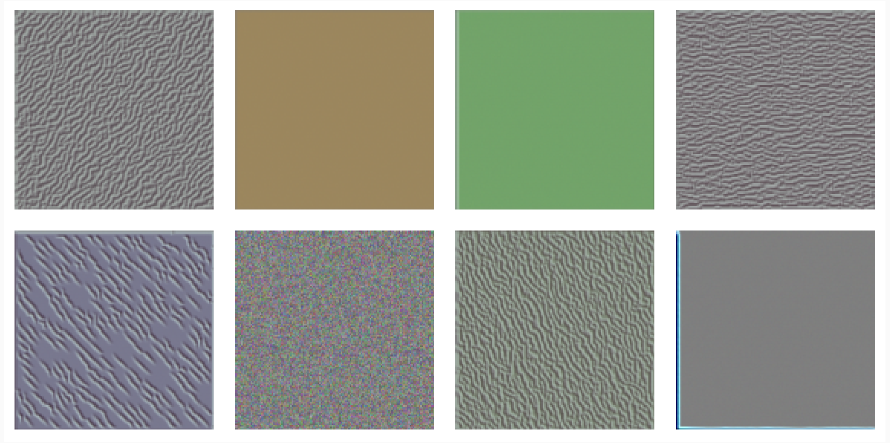
- Filters from the first layer encode colors and directional edges.
- On higher layers, simple textures and more complicated patterns emerge.

More information on VGG16 and ImageNet.

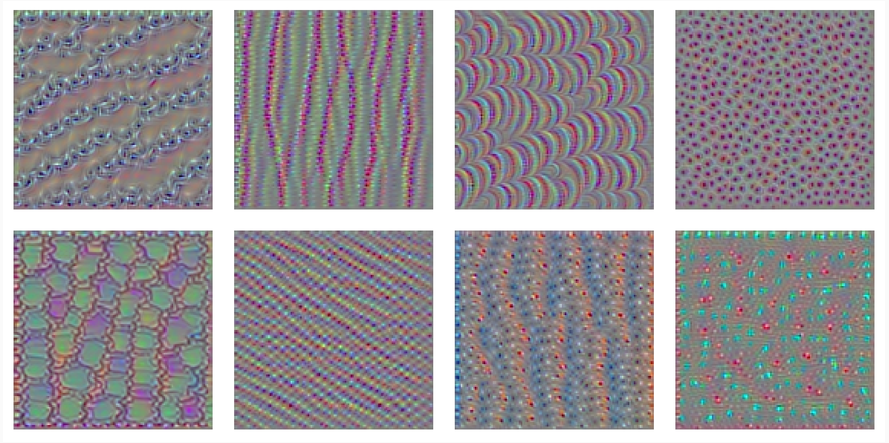
```
## Model: "vgg16"
## _________________________________________________________________________________
## Layer (type)                         Output Shape                    Param #
## =================================================================================
## input_1 (InputLayer)                 [(None, None, None, 3)]         0
## _________________________________________________________________________________
## block1_conv1 (Conv2D)                (None, None, None, 64)          1792
## _________________________________________________________________________________
## block1_conv2 (Conv2D)                (None, None, None, 64)          36928
## _________________________________________________________________________________
## block1_pool (MaxPooling2D)           (None, None, None, 64)          0
## _________________________________________________________________________________
## block2_conv1 (Conv2D)                (None, None, None, 128)         73856
## _________________________________________________________________________________
## block2_conv2 (Conv2D)                (None, None, None, 128)         147584
## _________________________________________________________________________________
## block2_pool (MaxPooling2D)           (None, None, None, 128)         0
## _________________________________________________________________________________
## block3_conv1 (Conv2D)                (None, None, None, 256)         295168
## _________________________________________________________________________________
## block3_conv2 (Conv2D)                (None, None, None, 256)         590080
## _________________________________________________________________________________
## block3_conv3 (Conv2D)                (None, None, None, 256)         590080
## _________________________________________________________________________________
## block3_pool (MaxPooling2D)           (None, None, None, 256)         0
## _________________________________________________________________________________
## block4_conv1 (Conv2D)                (None, None, None, 512)         1180160
## _________________________________________________________________________________
## block4_conv2 (Conv2D)                (None, None, None, 512)         2359808
## _________________________________________________________________________________
## block4_conv3 (Conv2D)                (None, None, None, 512)         2359808
## _________________________________________________________________________________
## block4_pool (MaxPooling2D)           (None, None, None, 512)         0
## _________________________________________________________________________________
## block5_conv1 (Conv2D)                (None, None, None, 512)         2359808
## _________________________________________________________________________________
## block5_conv2 (Conv2D)                (None, None, None, 512)         2359808
## _________________________________________________________________________________
## block5_conv3 (Conv2D)                (None, None, None, 512)         2359808
## _________________________________________________________________________________
## block5_pool (MaxPooling2D)           (None, None, None, 512)         0
## =================================================================================
## Total params: 14,714,688
## Trainable params: 14,714,688
## Non-trainable params: 0
## _________________________________________________________________________________
```

Example: Visualizing convnet filters (cont'd)

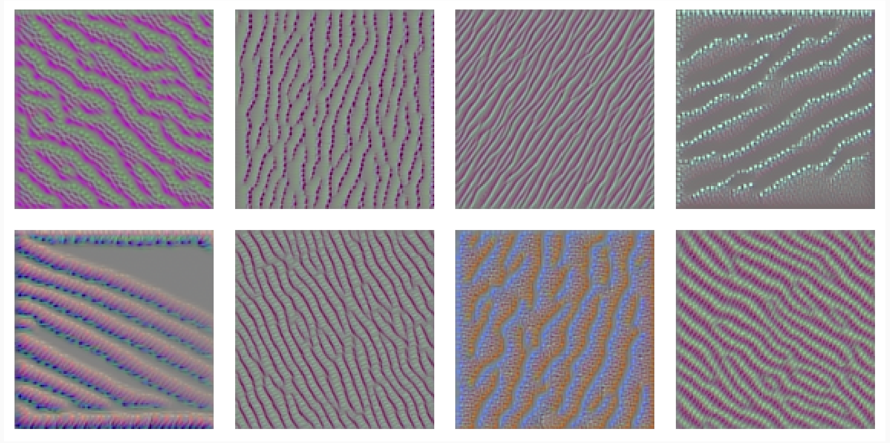
Layer `block1_conv1`



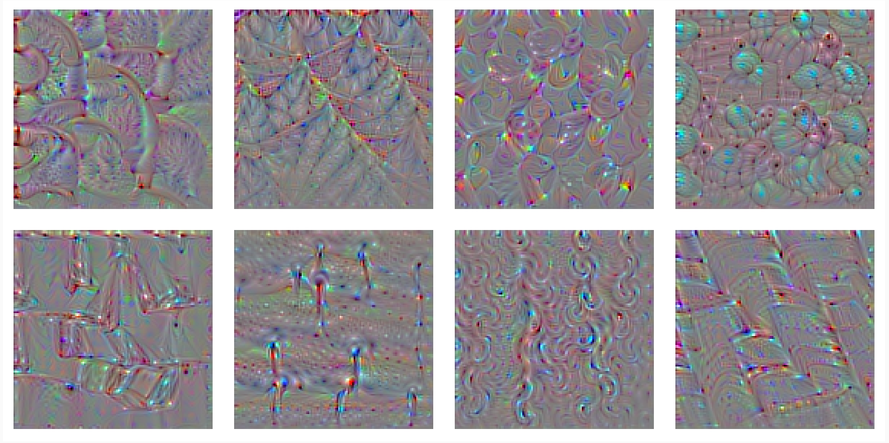
Layer `block3_conv3`



Layer `block2_conv2`



Layer `block5_conv1`



Attribution methods

A **class activation map** (CAM) of a particular category highlights the region in an input image that contributed to the identification of a category.

Based on properties of convnets,

- convolutional layers retain spatial information (which is lost in fully connected layers),
- deeper layers in a CNN capture higher-level representations.

Grad-CAM[1] uses gradient information (with backpropagation) on the last convolutional layer to weight the channels of its output activation map.

1. Calculate weights using the gradients of the score $\boldsymbol{y^c}$ for class $\boldsymbol{c}$ wrt the activation map $\boldsymbol{A^k}$.
2. Calculate weighted combination of the channels of the activation map.

$$\alpha_k^c = \frac{1}{Z}\sum_i \sum_j \frac{\partial y^c}{\partial A_{ij}^k}, \qquad L_{\text{Grad-CAM}}^c = \max\left(0, \sum_k \alpha_k^c A^k\right)$$

[1] Selvaraju et al: Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization

Example: Grad-CAM

Using VGG16 (whole network) trained on the ImageNet dataset.

```
model ← application_vgg16(weights = "imagenet")
img_path ← "catkeyboard.jpg"
img ← image_load(img_path, target_size = c(224, 224)) %>%
  image_to_array() %>%
  array_reshape(dim = c(1, 224, 224, 3)) %>%
  imagenet_preprocess_input()
preds ← predict(model, img)
imagenet_decode_predictions(preds, top = 5)[[1]]
```

```
##   class_name class_description      score
## 1  n02123394       Persian_cat 0.67351121
## 2  n02672831         accordion 0.04520087
## 3  n02342885           hamster 0.03817295
## 4  n02328150            Angora 0.03730543
## 5  n02086240          Shih-Tzu 0.01797394
```

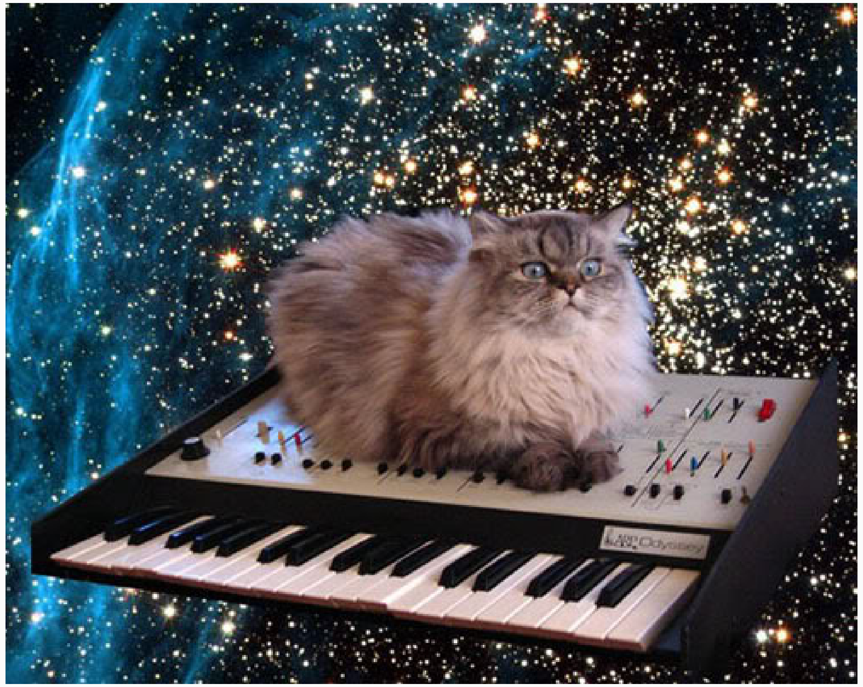
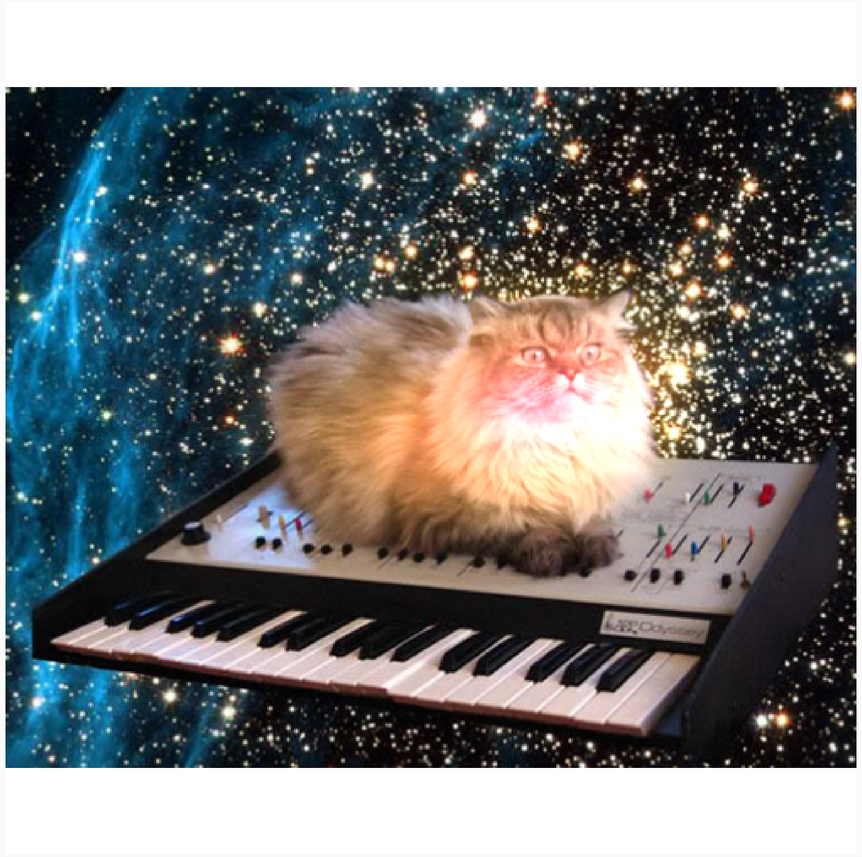


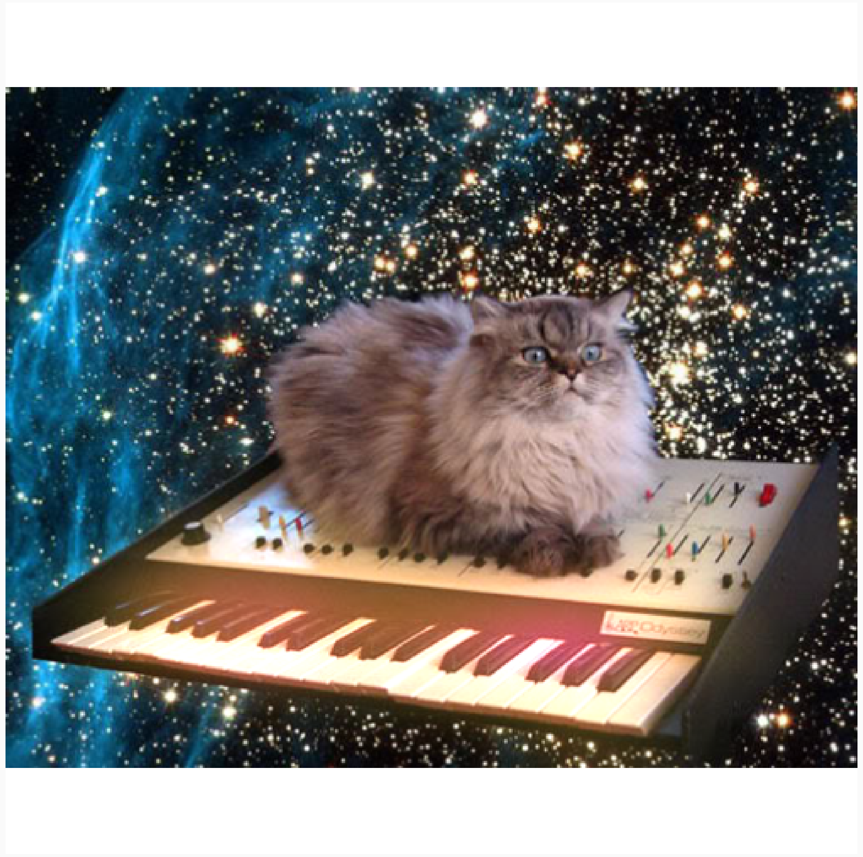
Image source: https://knowyourmeme.com/photos/279192-cat-on-a-keyboard-in-space

Example: Grad-CAM (cont'd)

Persian cat



Accordion



Sources and further readings

- The examples are adapted from J.J. Allaire's notebook (same as in Deep Learning with R book)
- Christoph Molnar: Interpretable Machine Learning, Chapter 7
- Olah, et al. (2017): Feature Visualization - How neural networks build up their understanding of images
- Olah, et al. (2018): The Building Blocks of Interpretability
- A curated list of machine learning interpretability resources
- Lucid: A collection of infrastructure and tools for research in neural network interpretability.