STAT430: Machine Learning for Financial Data

LarryHua.com/teaching

Spring 2019

Convolutional neural networks

How to preprocess image data in Keras?

```
train_dir <- file.path(data_dir, "train") # should contain one sub-directory per class

train_datagen <- image_data_generator(rescale = 1/255)
```

- batches of 150 × 150 RGB images (shape `(20, 150, 150, 3)`) and binary labels
- the generator yields these batches indefinitely: it loops endlessly over the images in the target folder

```
train_generator <- flow_images_from_directory(
  directory = train_dir,
  generator = train_datagen,
  target_size = c(150, 150),
  batch_size = 20,
  class_mode = "binary"
)
```

- after `train_generator` is generated, treat it as the data

```
model %>% fit_generator(generator = train_generator, ... )
```

How to preprocess image data in Keras?

- Loads an image into PIL format
  - `image_load()`
- Generates batches of augmented/normalized data from images and labels, or a directory
  - `flow_images_from_data()`
  - `flow_images_from_directory()`
- Generate minibatches of image data with real-time data augmentation
  - `image_data_generator()`
- Convert image to 3D tensor
  - `image_to_array()`
  - `image_array_resize()`
  - `image_array_save()`
- Retrieve the next item
  - `generator_next()`

Example - image classification

- Use a stack of alternated `layer_conv_2d()` and `layer_max_pooling_2d()` before dense layers
- For bigger images and a more complex problem, make network larger
  - augment the capacity of the network
  - further reduce the size of the feature maps
  - usually, depth increases in the network, whereas the size of the feature maps decreases
- Try R

Data augmentation

- generating more training data from existing training samples, by "augmenting" the samples via a number of random transformations that yield believable-looking images
- used almost universally when processing images with deep learning models for reducing overfitting
- helps the model get exposed to more aspects of the data and generalize better
- In Keras, this can be done by configuring a number of random transformations through `image_data_generator()`

```
datagen <- image_data_generator(
  rescale = 1/255,
  rotation_range = 40, # a value in degrees (0-180), a range within which to randomly rotate pictures
  width_shift_range = 0.2, # ranges (as a fraction of total width or height)
  height_shift_range = 0.2,
  shear_range = 0.2, # randomly applying shearing transformations
  zoom_range = 0.2, # randomly zooming inside pictures
  horizontal_flip = TRUE, # randomly flipping half of the images horizontally
  fill_mode = "nearest" # strategy used for filling in newly created pixels
                        # which can appear after a rotation or a width/height shift
)
```

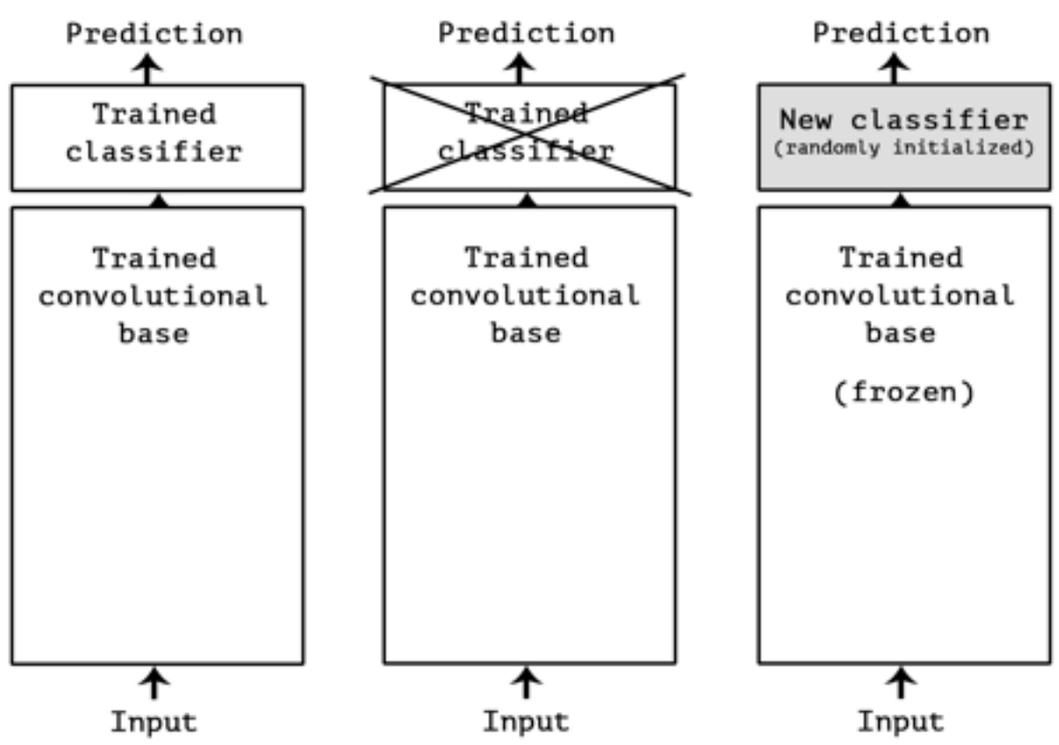
- Try R

Use pre-trained network

- A common and highly effective approach to deep learning on small image datasets
- A pre-trained network is simply a saved network previously trained on a large dataset
- If this original dataset is large enough and general enough, then the spatial feature hierarchy learned can be used even for different classes
    - eg, pre-trained network with ImageNet (where classes are mostly animals and everyday objects) is used for identifying furniture items
    - key advantage of deep learning
- two ways to leverage a pre-trained network
    - feature extraction
    - fine-tuning

Use pre-trained network - feature extraction

- using the representations learned by a previous network to extract interesting features from new samples, and then these features are then run through a new classifier
- convnet: convolutional base + densely-connected classifier
- feature extraction only takes the convolutional base of a previously-trained network, runs the new data through it, and trains a new classifier on top of the output



Use pre-trained network - feature extraction

- layers that come earlier in the model extract local, highly generic feature maps (such as visual edges, colors, and textures)
- layers higher-up extract more abstract concepts (such as "cat ear" or "dog eye")
- the convolutional base are likely to be more generic and reusable
- the representations learned by the densely-connected classifier are very specific to the set of classes
- if new dataset differs a lot, then use only the first few layers

Use pre-trained network - feature extraction

- Two approaches for feature extraction
  - Predict your new data based on `conv_base`, and then use these outputs as inputs to a new model
    - fast and cheap to run
    - only requires running the convolutional base once for every input image
    - does not allow data augmentation
  - Extend `conv_base` by adding dense layers on top, and running the whole thing end to end on the input data
    - far more expensive than the first
    - allows data augmentation

Use pre-trained network - feature extraction

- ImageNet: 1.4 million labeled images and 1000 different classes, contains many animal classes, including different species of cats and dogs
- Pretrained networks on the ImageNet dataset in Keras
  - Xception InceptionV3 ResNet50 VGG16 VGG19 MobileNet
- Example: VGG16 architecture: Karen Simonyan and Andrew Zisserman, 2014
  - Try R

Use pre-trained network - fine-tuning

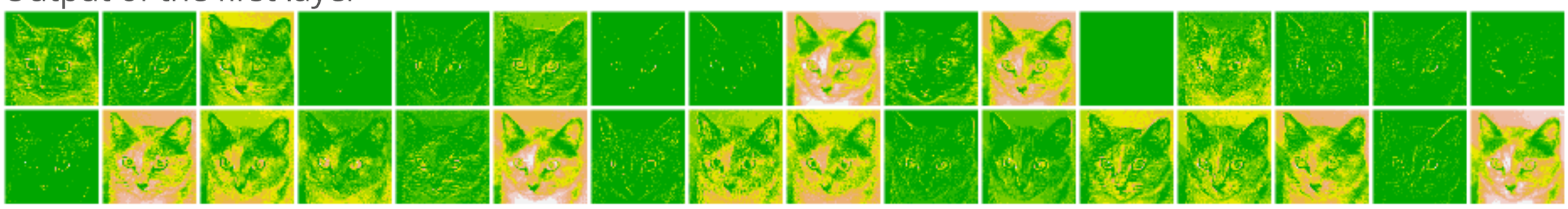
- unfreeze a few of the later layers of a frozen model base used for feature extraction, and jointly train both the newly added part of the model and the unfrozen layers
- steps:
  - add your custom network on top of an already trained base network
  - freeze the base network
  - train the part you added
  - unfreeze some layers in the base network
  - jointly train both these layers and the part you added.
- only fine tune later layers
  - earlier layers in the convolutional base encode more generic, reusable features, while layers higher up encode more specialized features
- Try R

Visualize learned convnet

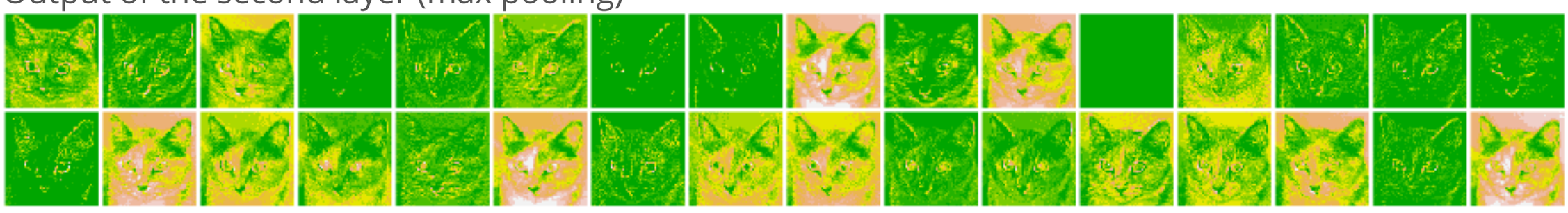
- useful for understanding the "black box" of convnet
- visualizing intermediate convnet outputs
  - display the feature maps that are output by various convolution and pooling layers, given a certain input
    - the output of a layer is often called its activation
  - understand how successive convnet layers transform their input
  - get a first idea of the meaning of individual convnet filters
- visualizing convnets filters
  - understand precisely what visual pattern or concept of each filter
- visualizing heatmaps of class activation in an image
  - understand which part of an image dominates the classification

Visualize learned convnet - intermediate outputs

- the first layer acts as a collection of various edge detectors
- the activations become increasingly abstract and less visually interpretable, but carry more information related to the class
- the sparsity of the activations is increasing with the depth of the layer: `blank` means that the pattern encoded by the filter isn't found
- **information distillation pipeline**: The features extracted by a layer get increasingly abstract with the depth of the layer
- Output of the first layer

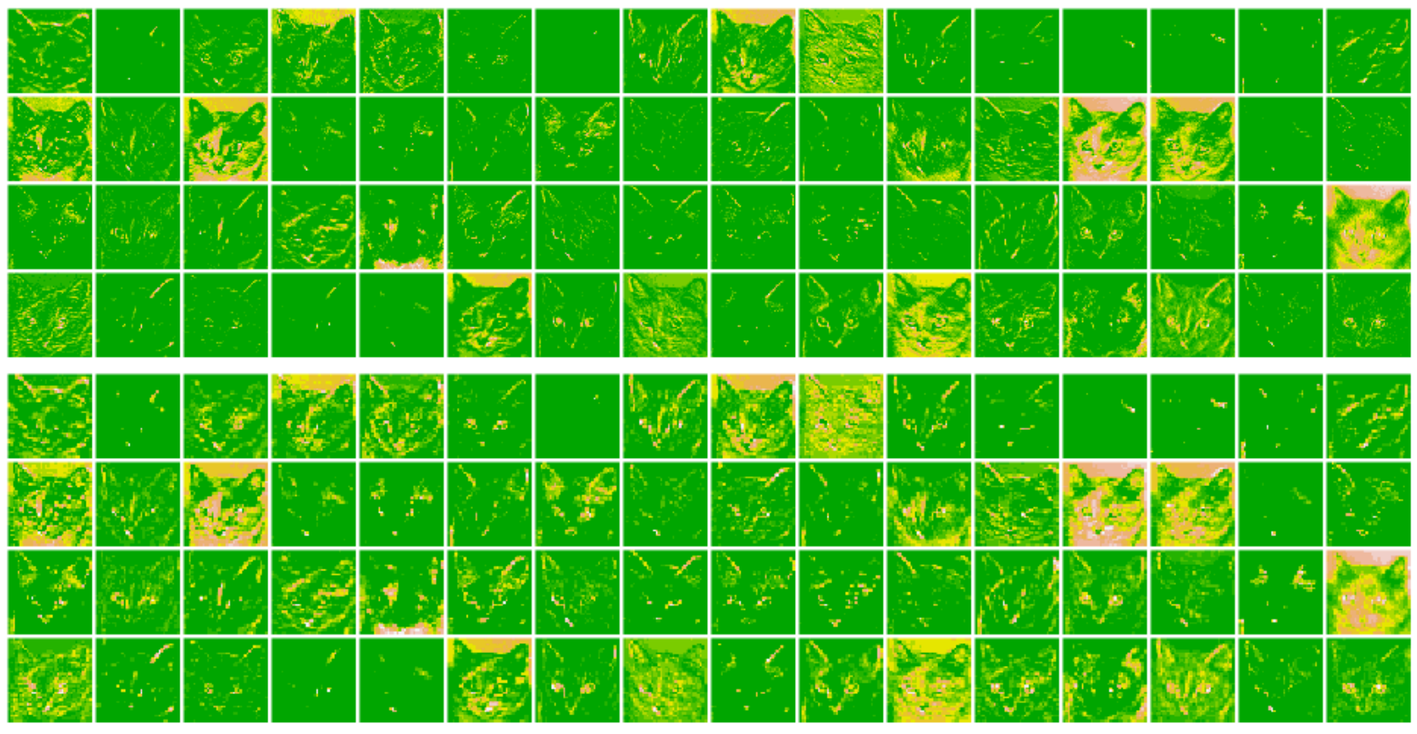


- Output of the second layer (max pooling)



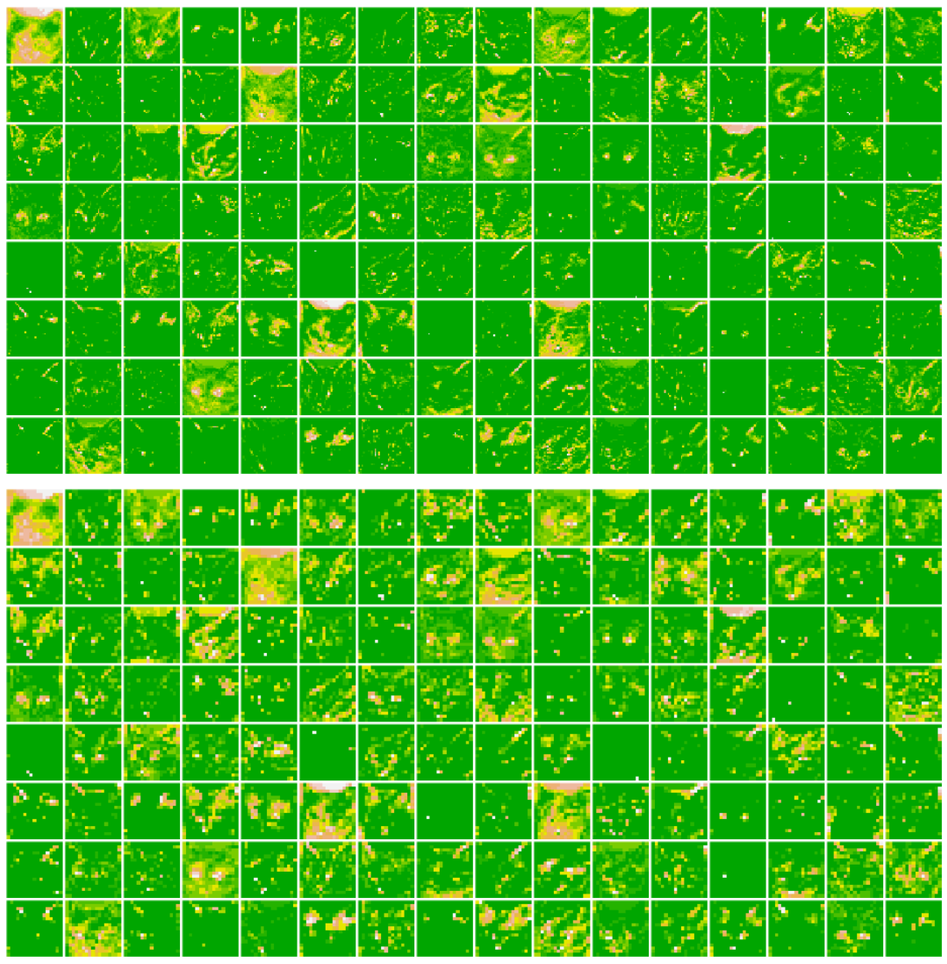
Visualize learned convnet - intermediate outputs

- Layers 3,4
- Try R



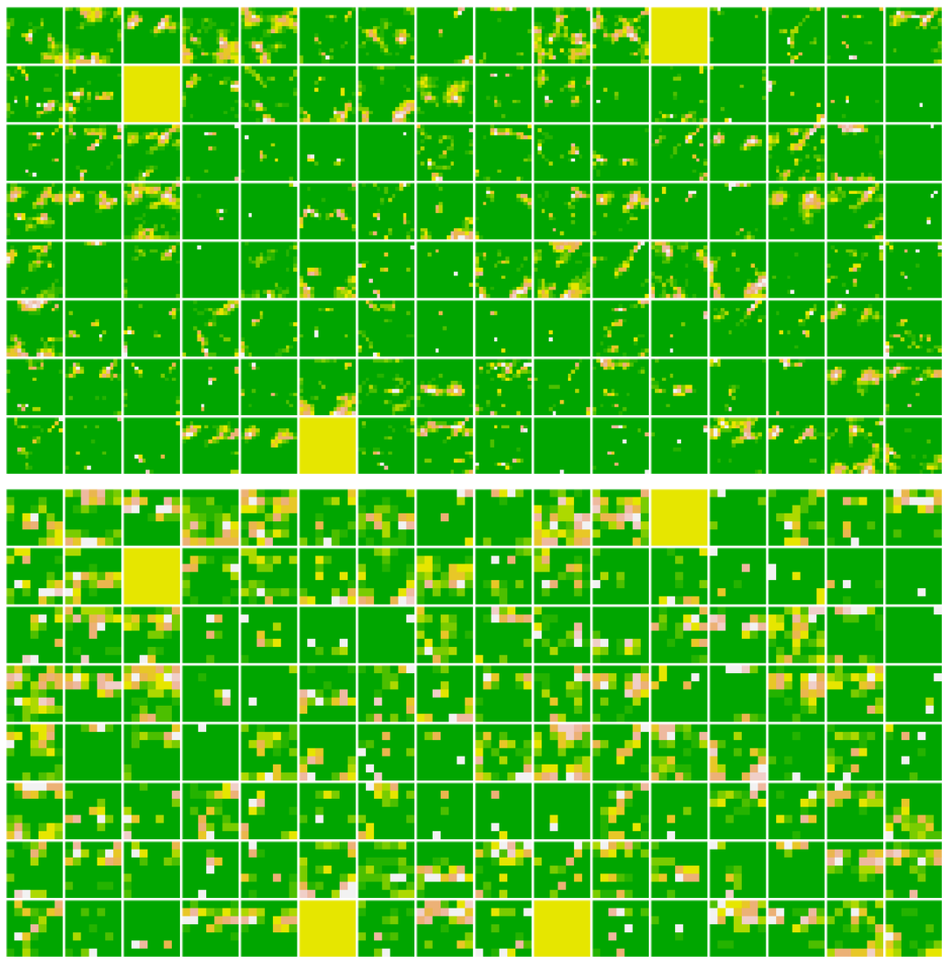
Visualize learned convnet - intermediate outputs

- Layers 5,6



Visualize learned convnet - intermediate outputs

- Layers 7,8

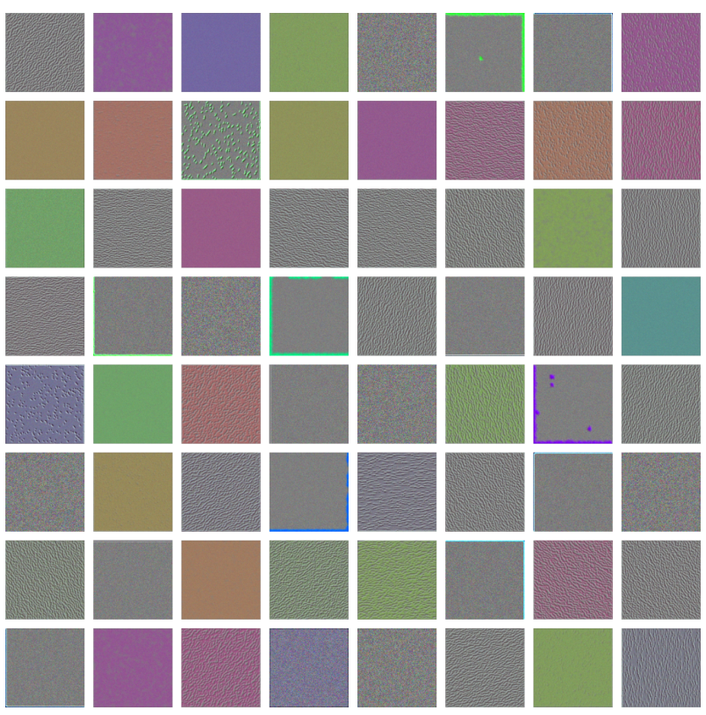


Visualize learned convnet - filters

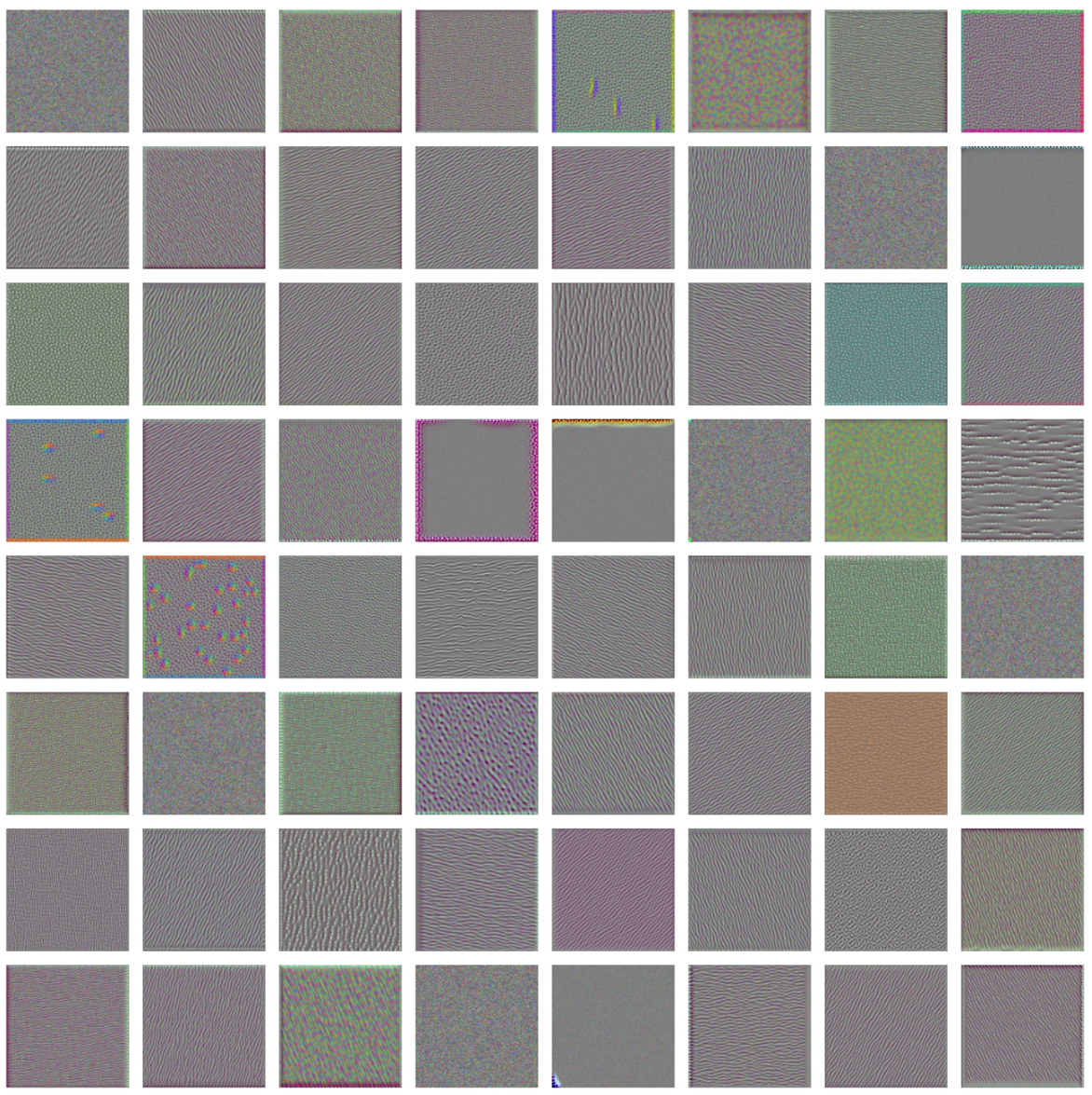
- display the visual pattern that each filter is meant to respond to
- start from a blank input image, then use stochastic gradient descent to adjust the values of the input image so as to maximize the activation value of a given filter in a given convolution layer
- Try R

Visualize learned convnet - filters

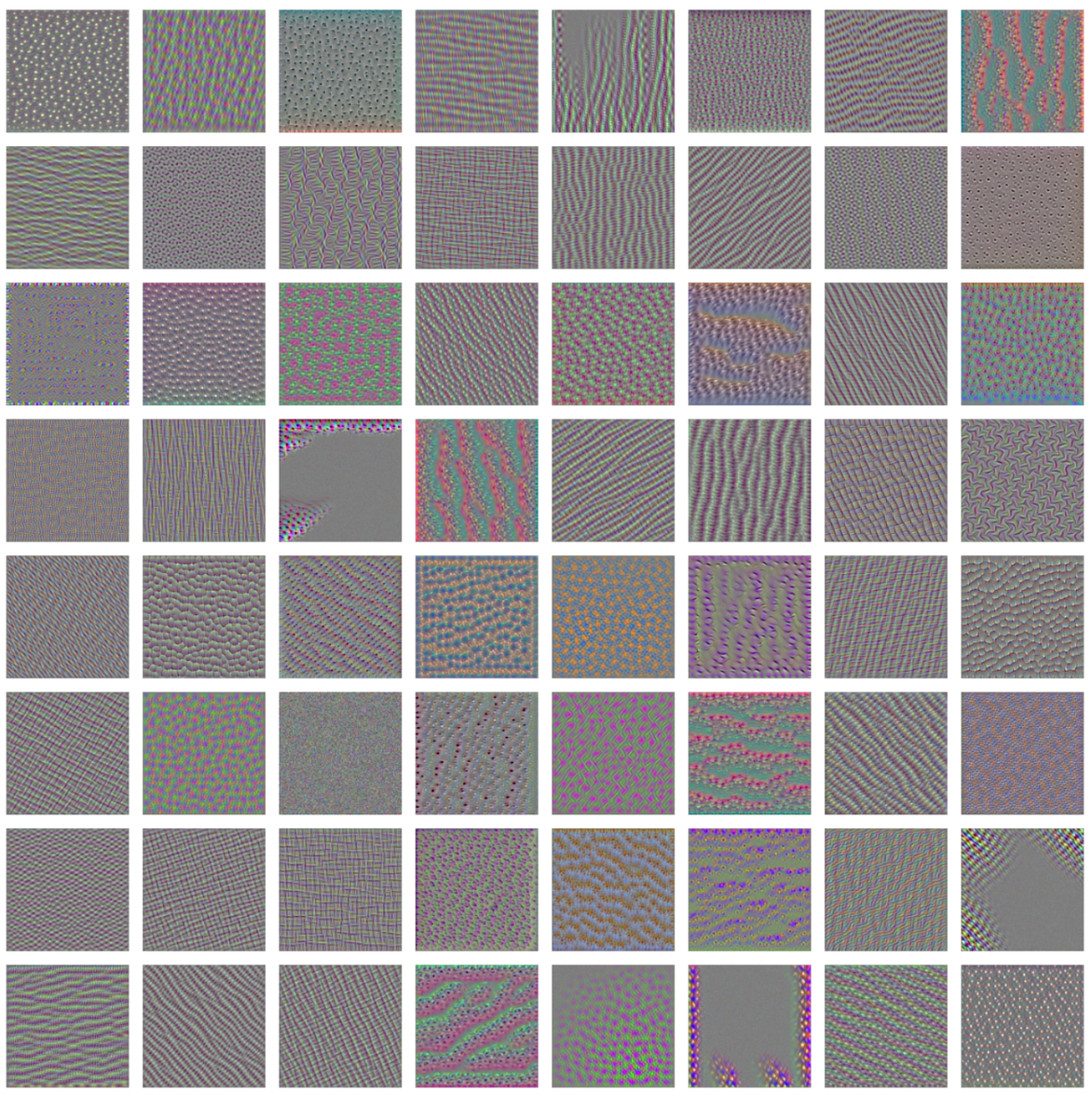
- the filters from the first layer in the model (`block1_conv1`) encode simple directional edges and colors (or colored edges in some cases)
- the filters from `block2_conv1` encode simple textures made from combinations of edges and colors
- the filters in higher layers begin to resemble textures found in natural images: feathers, eyes, leaves, and so on



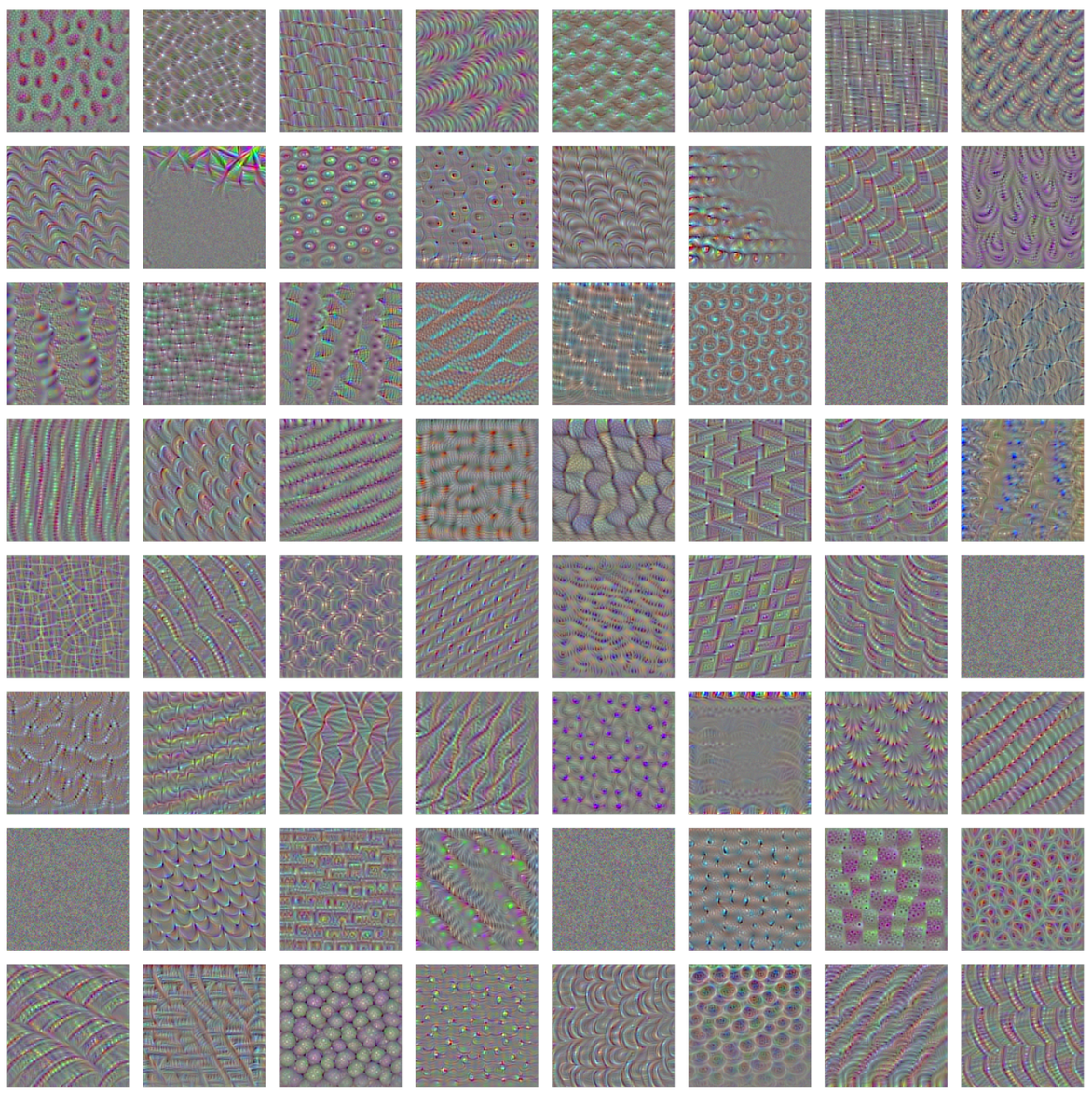
Visualize learned convnet - filters



Visualize learned convnet - filters



Visualize learned convnet - filters



Visualize learned convnet - heatmaps

- identify the parts of a given image that lead to the final classification decision
- Try R

CNN for limit order book

- CNN is useful for capturing presence of spatial-temporal related features
- CNN can account for spatial and temporal dependence of multivariate financial time series
- limit order book with Level II data can be looked at `spatial-temporal` images
    - directions of price movement can be classified by features of those `spatial-temporal`
    - but we need some special treatments for financial data
    - Try R
- Back to Course Scheduler