Pictures

Arthur Charpentier

UQAM

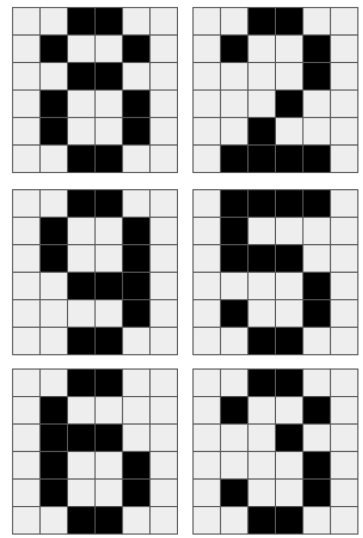
Actuarial Summer School 2019

Statistics & ML Toolbox : Neural Nets

Very popular for pictures...
Picture $\boldsymbol{x}_i$ is

- a $n \times n$ matrix in $\{0,1\}^{n^2}$ for black & white
- a $n \times n$ matrix in $[0,1]^{n^2}$ for grey-scale
- a $3 \times n \times n$ array in $([0,1]^3)^{n^2}$ for color
- a $T \times 3 \times n \times n$ tensor in $(([0,1]^3)^T)^{n^2}$ for video

y here is the label ("8", "9", "6", etc)
Suppose we want to recognize a "6" on a picture



$$m(\boldsymbol{x}) = \begin{cases} +1 \text{ if } \boldsymbol{x} \text{ is a "6"} \\ -1 \text{ otherwise} \end{cases}$$

Statistics & ML Toolbox : Neural Nets

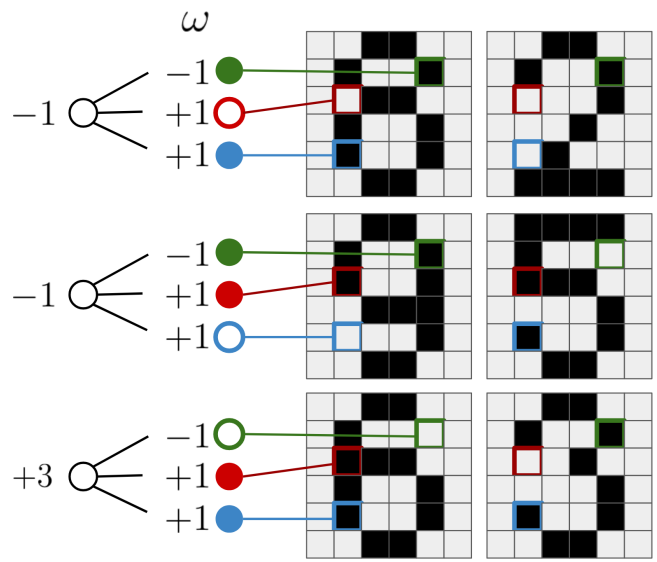
Consider some specifics pixels, and associate weights ω such that

$$\widehat{m}(\boldsymbol{x}) = \text{sign}\left(\sum_{i,j} \omega_{i,j} x_{i,j}\right)$$

where

$$x_{i,j} = \begin{cases} +1 \text{ if pixel } x_{i,j} \text{ is black} & \blacksquare \\ -1 \text{ if pixel } x_{i,j} \text{ is white} & \square \end{cases}$$

for some weights $\omega_{i,j}$ (that can be negative...)

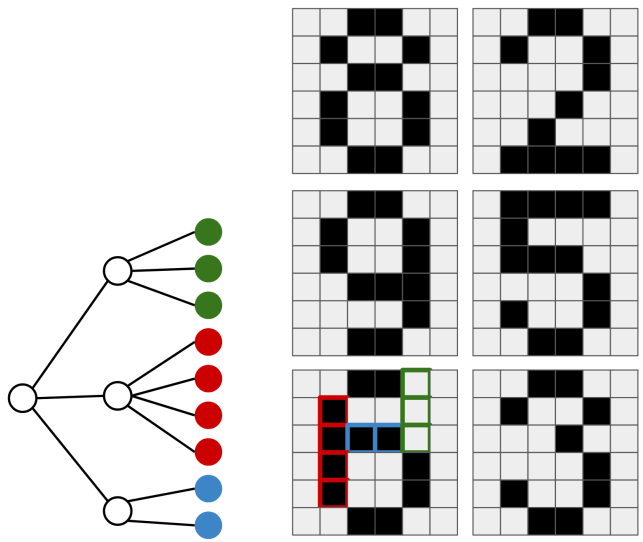


Statistics & ML Toolbox : Neural Nets

A deep network is a network with a lot of layers

$$\widehat{m}(\boldsymbol{x}) = \text{sign}\left(\sum_i \omega_i \widehat{m}_i(\boldsymbol{x})\right)$$

where $\widehat{m}_i$'s are outputs of previous neural nets. Those layers can capture shapes in some areas nonlinearities, cross-dependence, etc

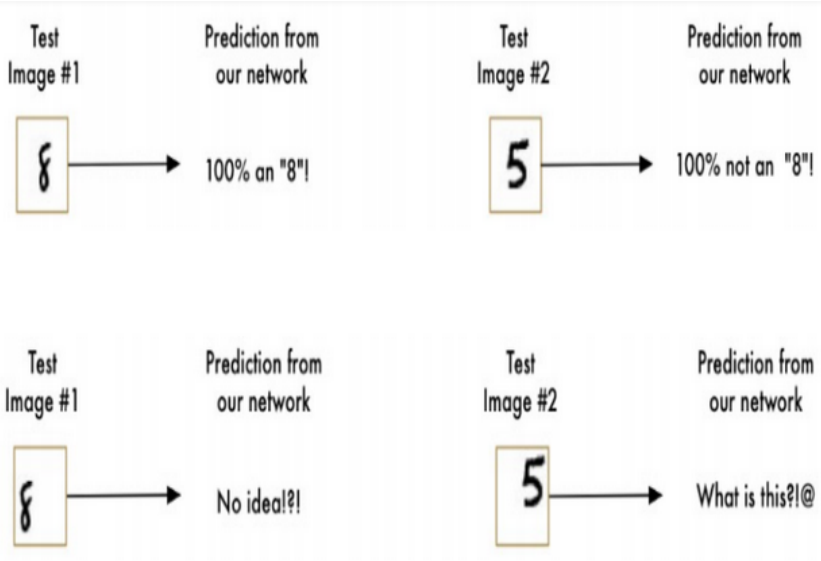


Detection & Recognition

can be complicated... see Marr (1982, Vision)

Convolutional Network

FIGURE 11.9. *Examples of training cases from ZIP code data. Each image is a 16 × 16 8-bit grayscale representation of a handwritten digit.*

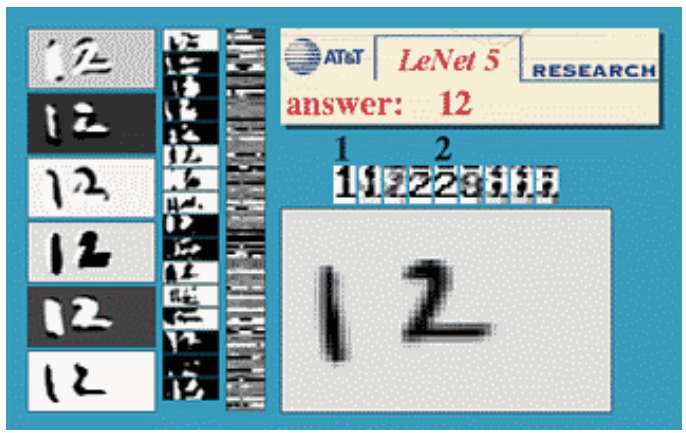


source: Hastie *et al.* (2009, The Elements of Statistical Learning)

Neural Nets fails to recognize '8' when the digit is not centered translation, scale and (small) rotation invariances are needed

Rotation, Scaling, Mixing, etc

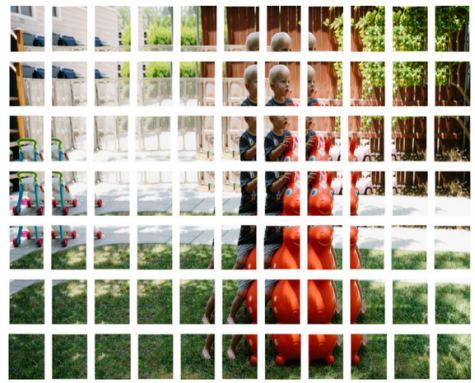
See http://yann.lecun.com/exdb/lenet/



See also http://scs.ryerson.ca/~aharley/vis/conv

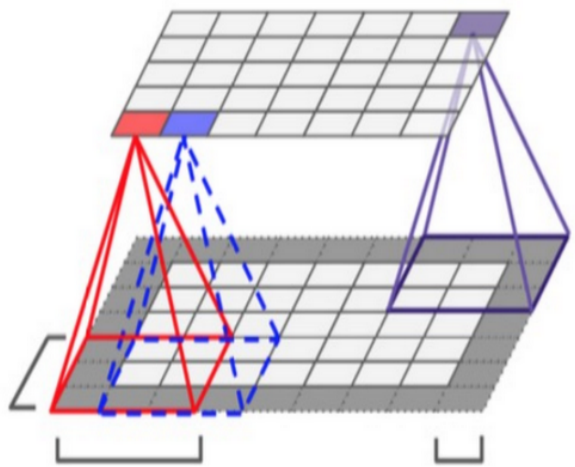
Convolutional Network

Break the image into overlapping image tiles and, feed each image tile into a small neural network with the same weights (and the same activation function)



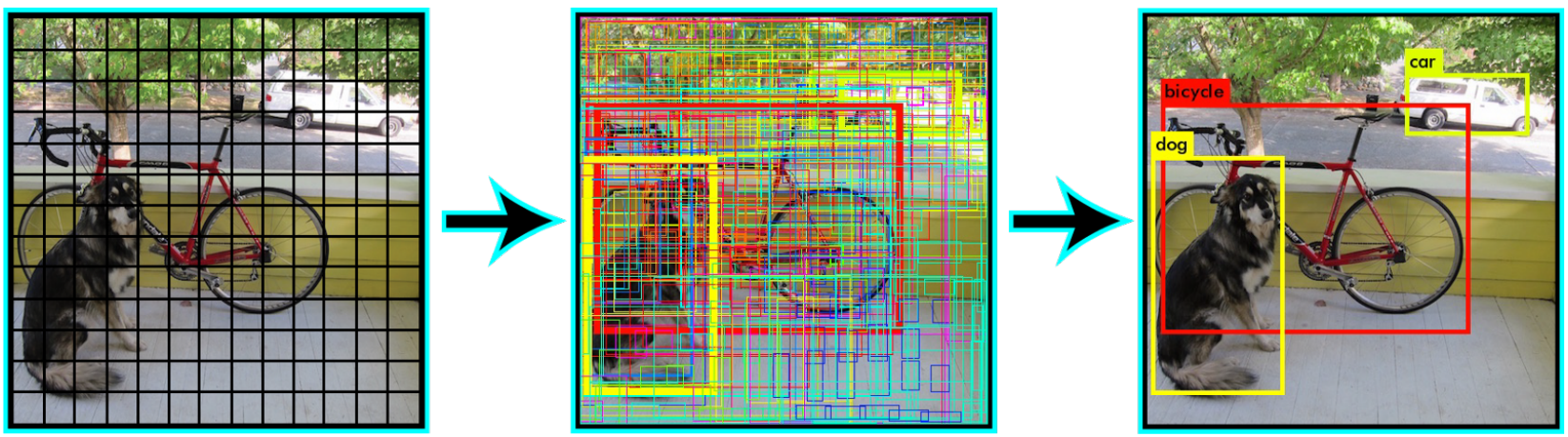
1. ConvNets exploit spatially local correlation: each neuron is locally- connected (to only a small region of the input volume)
2. Reduce the size of the array, using a pooling algorithm.

(pooling step reduces the dimensionality of each feature)



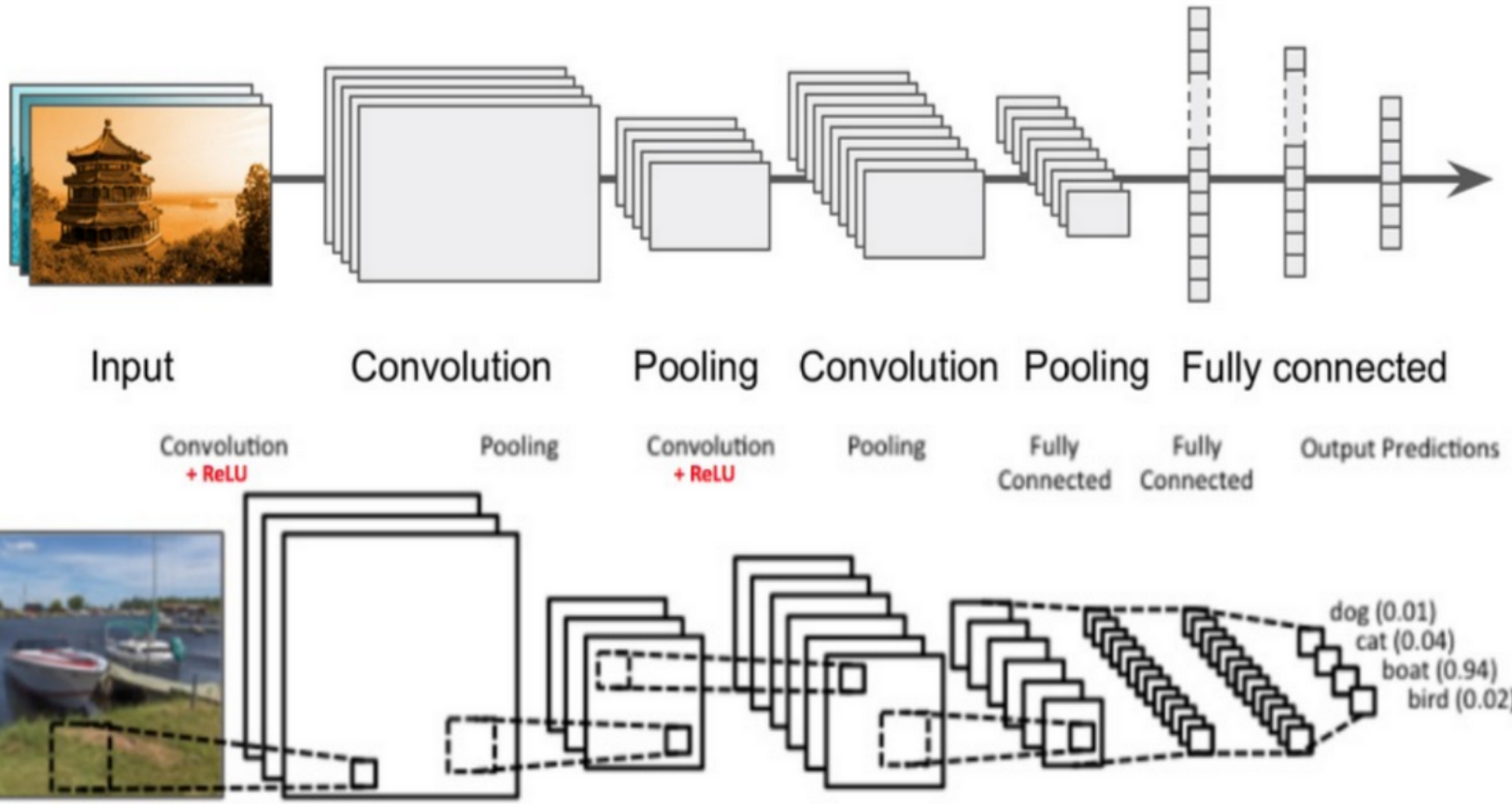
Convolutional Network

pooling:
- makes the input representations smaller and more manageable
- reduces the number of weights and links in the network, therefore, controlling overfitting
- makes the network invariant to small transformations, distortions and translations in the input image
- helps us arrive at an almost scale invariant representation of our image



(via yolo for objects detection)

Convolutional Network



Convolutional Network

Can be used to detect faces on a picture

via LeCun (2016, Leçon inaugurale)

Convolutional Network

and for object detection (even on a video)

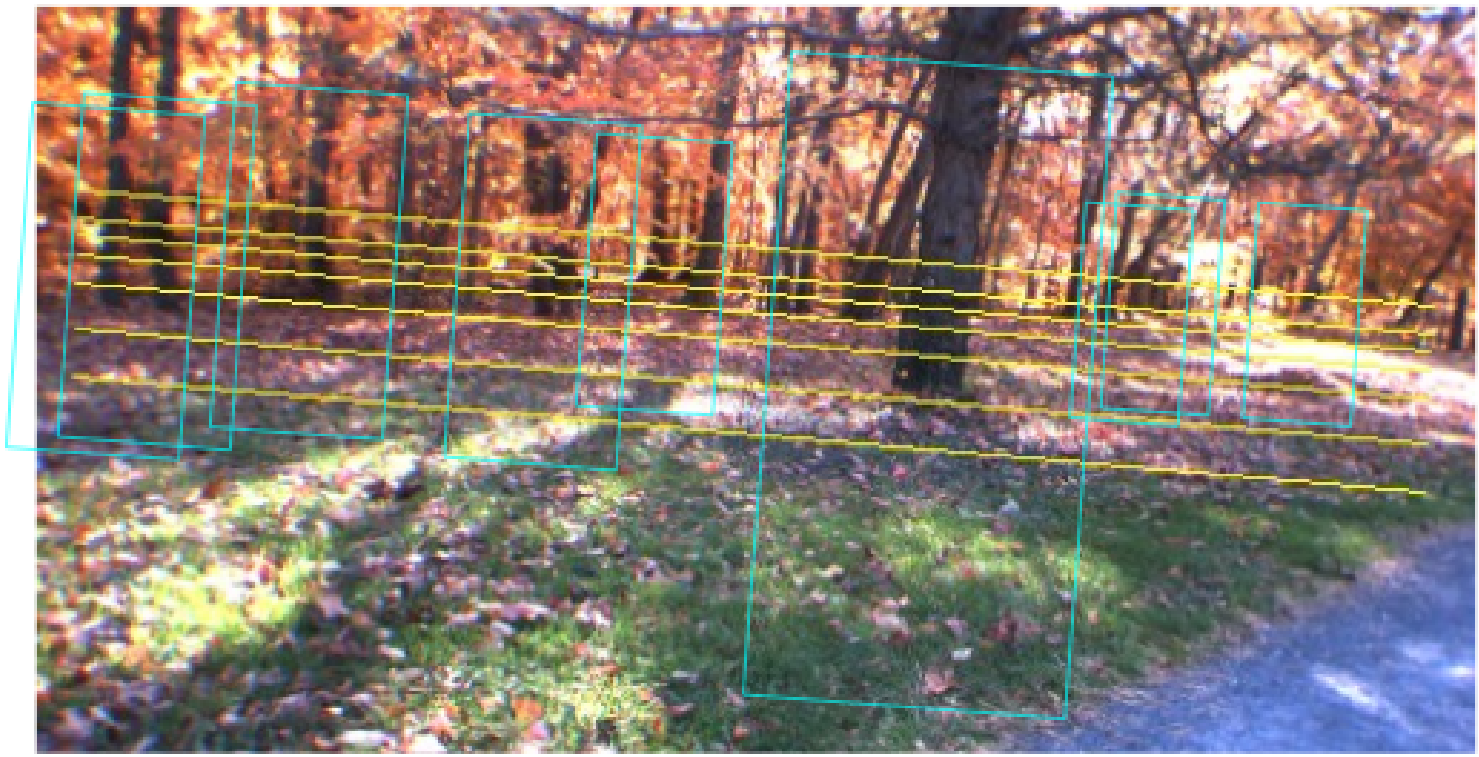
via LeCun (2016, Leçon inaugurale)

Convolutional Network

via LeCun (2016, Leçon inaugurale)

Convolutional Network

In some application, need to find scale invariant transformations

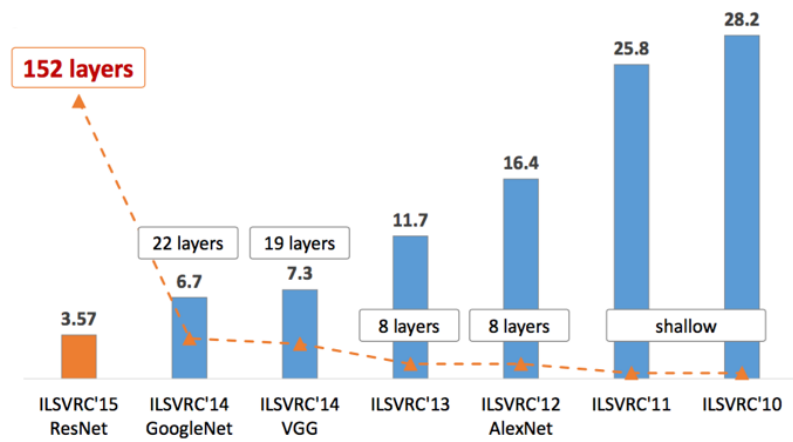


via LeCun (2016, Leçon inaugurale)

Recognition with Neural Nets

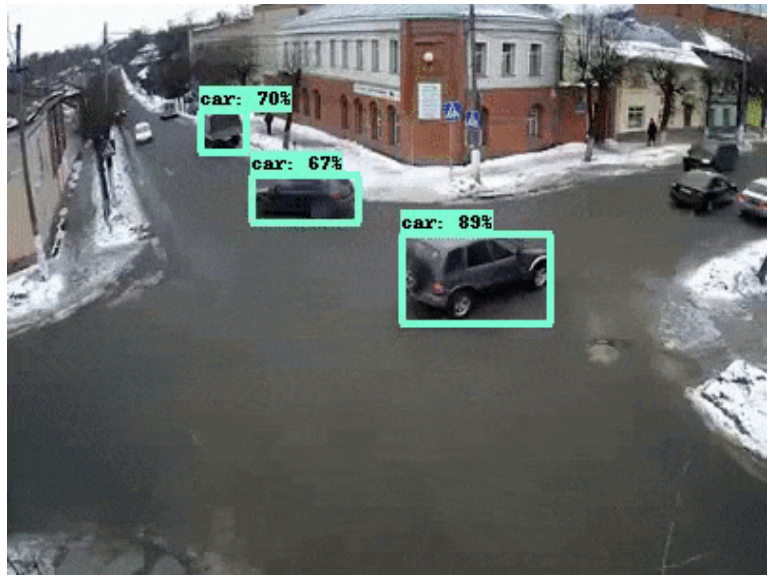
Mid-2000s caltech101 dataset, with 101 categories, 30 pictures per category (training sample)
2012 ImageNet dataset, 1.2 million (training) pictures, 20,000 categories see Li Fei-Fei's page

OverFeat 2012 13.8% error,
GoogLeNet 2014 6.6%
(22 layers),
ResNet 2015 3.6% (152 layers)
see @sidereal's post



From Pictures to Videos

A picture is a $w \times h \times 3$ tensor
A video is a $w \times h \times 3 \times t$ tensor
One can use RNN
recurrent neural nets (or LSTM)
Can also be use to identify objects
see https://towardsdatascience.com



Which Pictures? Satellite

From https://unequalscenes.com/, Hout Bay, about 15km south of Cape Town, South Africa and Dar es Salaam, Tanzania.

Which Pictures? Satellite

From https://nytimes.com/, Chennai (Madras) in India, 2018 and 2019

Which Pictures? Satellite

via https://geoweb.iau-idf.fr/ (1949 on the left, 2019 on the right)



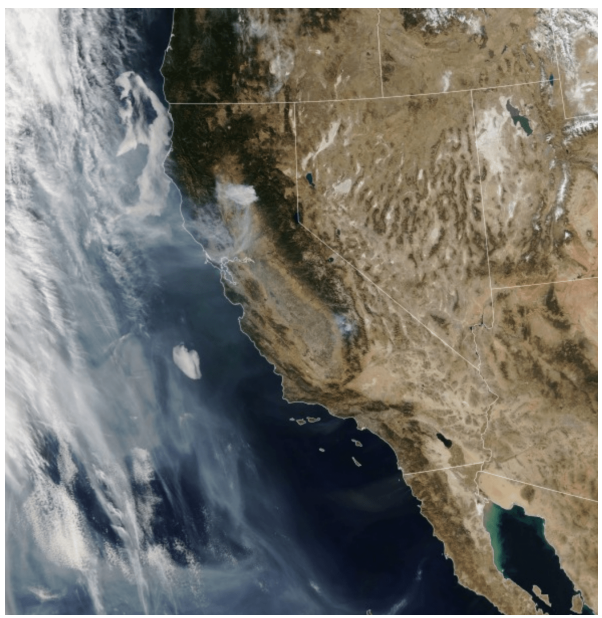

Which Pictures? Satellite

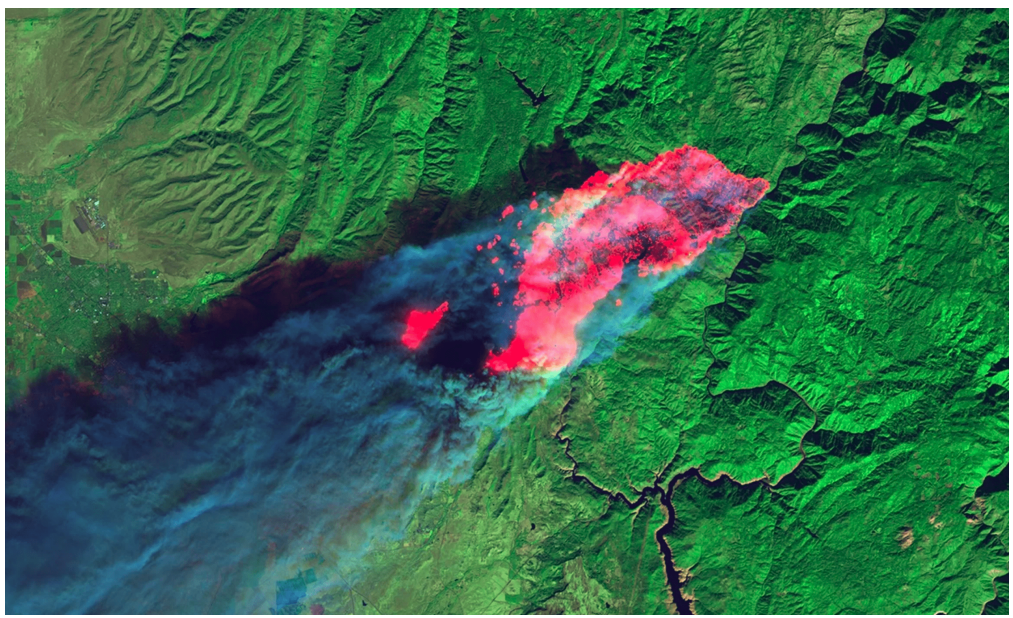
Possible use for crop insurance:
- To assess crop share (relative proportions) in a large area (no georeference available of the fields)
- To estimate yield of an specific crop/season in a large area (no georeference available of the fields)
- To detect and to georeference fields with specific crops
- To detect kind-of-crop info from specific fields

Various sources can be used :
- MODIS (Moderate Resolution Imaging Spectroradiometer) 250m $\times$ 250m, 2 images per day
- LANDSAT, 15m $\times$ 15m, 1 image every 16 days
(see also https://landsat.visibleearth.nasa.gov)

Which Pictures? Satellite

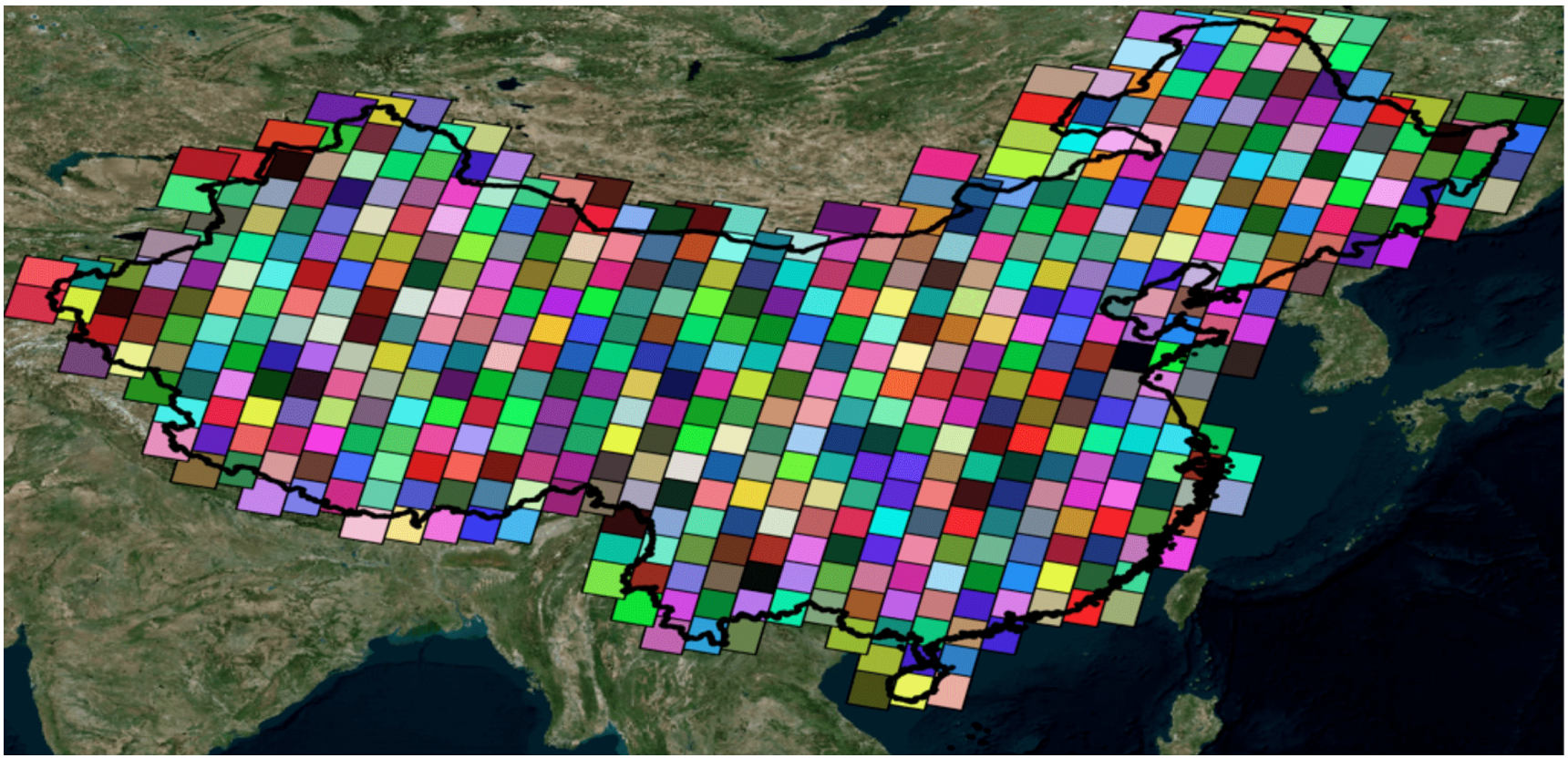
Moderate Resolution Imaging Spectroradiometer (MODIS) on NASA's Aqua satellite of the Camp Fire on November 12, 2018. [Photo: NASA Earth Observatory vs Nov. 8, 2018, by Landsat 8, shows short-wave infrared (red), which gives the full extent of the actively burning area of the Camp Fire





Which Pictures? Satellite

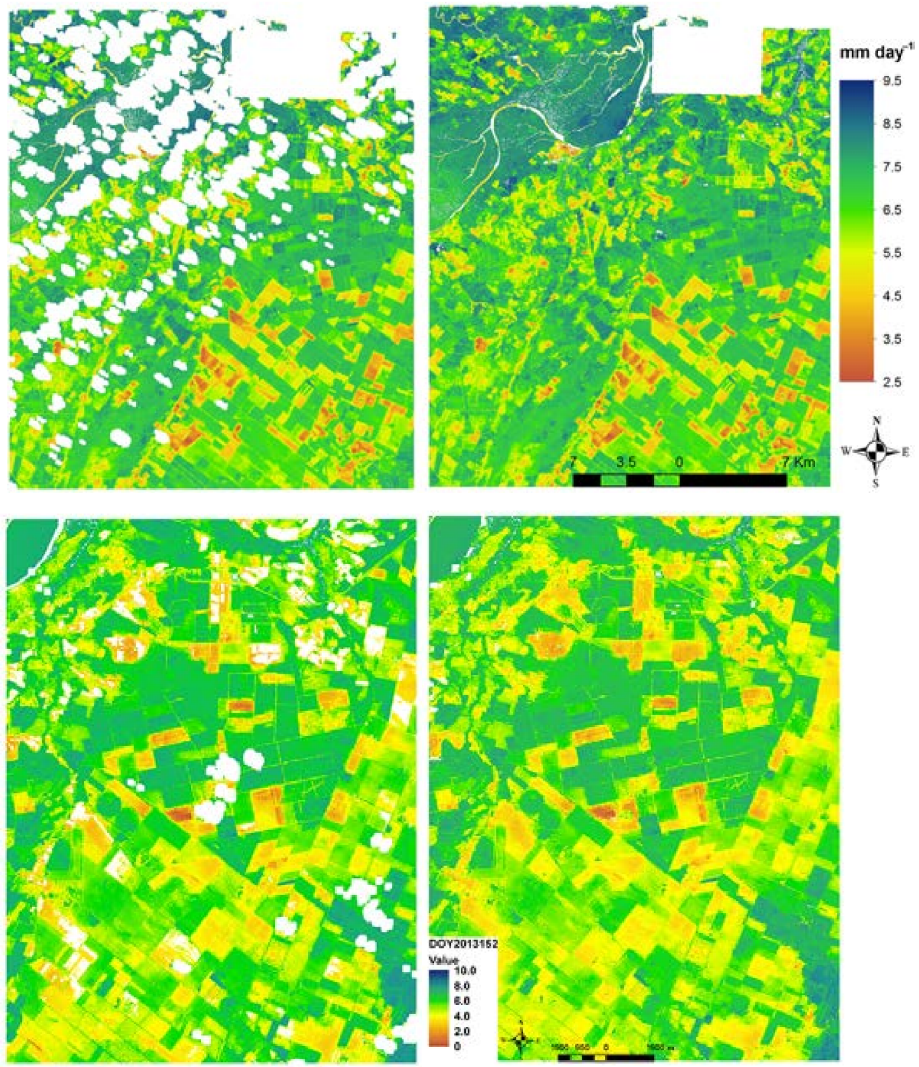
Landsat picture = 17,500px × 14,500, 950Mb
Landsat 8 to cover China : 530 tiles, 23 per year ∼ 11Tb per year



Which Pictures? Satellite

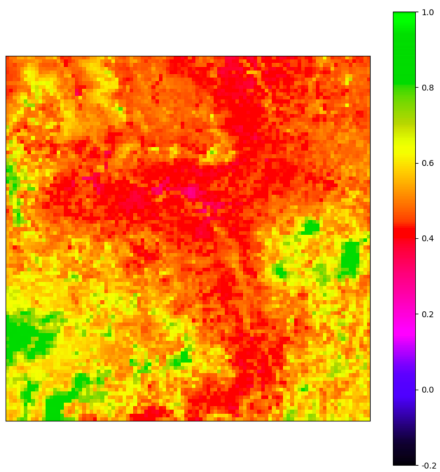
Satellite pictures are
more than visible colors
TIRS (Thermal Infrared Sensor)

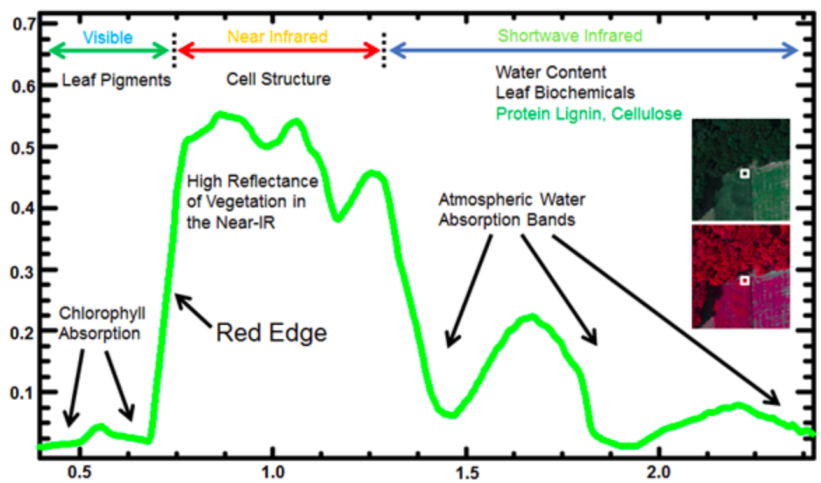
Practical problem : clouds
(see https://forestthreats.org/
or https://harrisgeospatial.com/)



Which Pictures? Satellite

Elowitz (2013, What is Imaging Spectroscopy (Hyperspectral Imaging)?)
Infrareds used to produce (e.g.) some Normalized Difference Vegetation Index (NDVI)
3d picture : lat, long, index (ndvi)





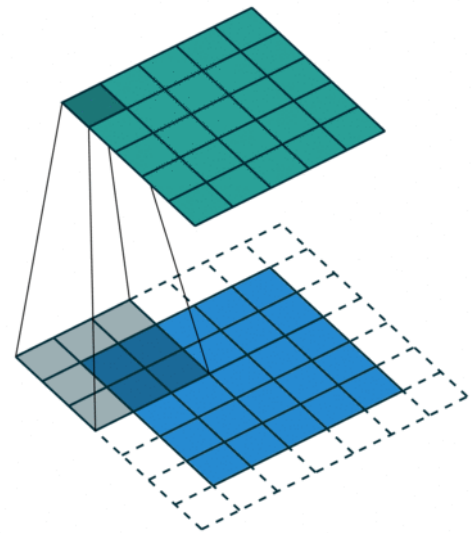
Practicals with Ewen Gallic on wildfires and natural disasters

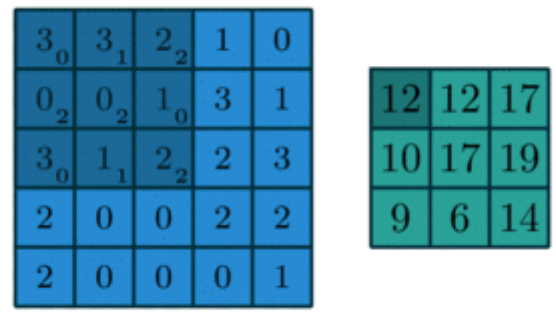
Convolutional Network

The light blue grid is called the input feature map. A kernel (shaded area) of value

$$\begin{pmatrix} 0 & 1 & 2 \\ 2 & 2 & 0 \\ 0 & 1 & 2 \end{pmatrix}$$

slides across the input feature map.
(see http://deeplearning.net/)



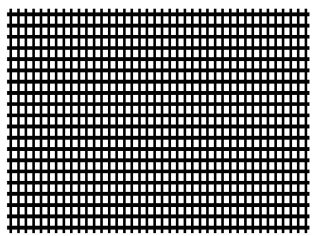
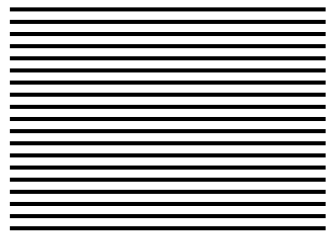
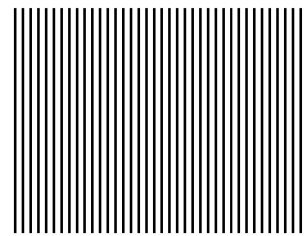


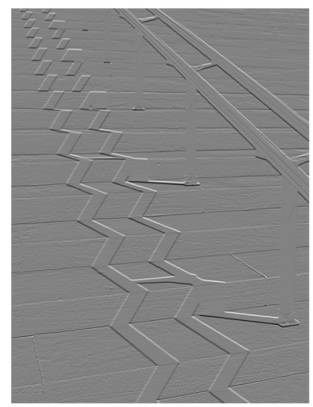
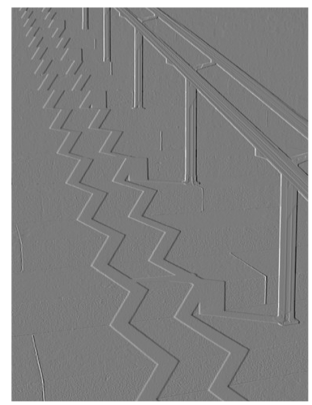
Extracting Shapes & Contours

See Sobel Operator,

$$\text{horizontal} \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}$$

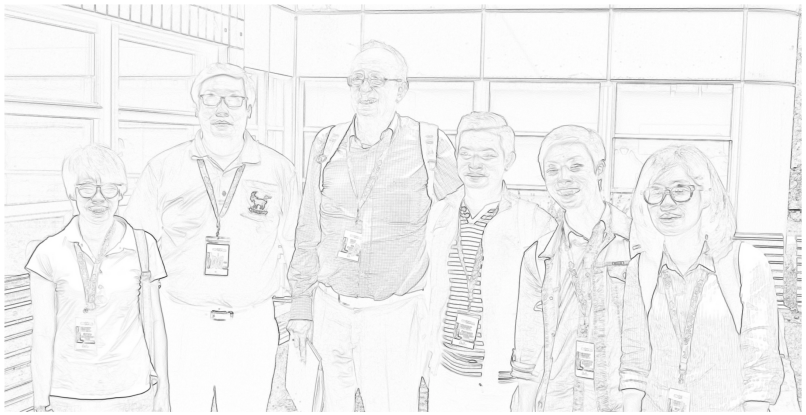
$$\text{vertical} \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$$

Extracting Shapes & Contours

One can also compute gradients to extract contours

Picture Classifier

use https://github.com/bnosac/image to extract contours

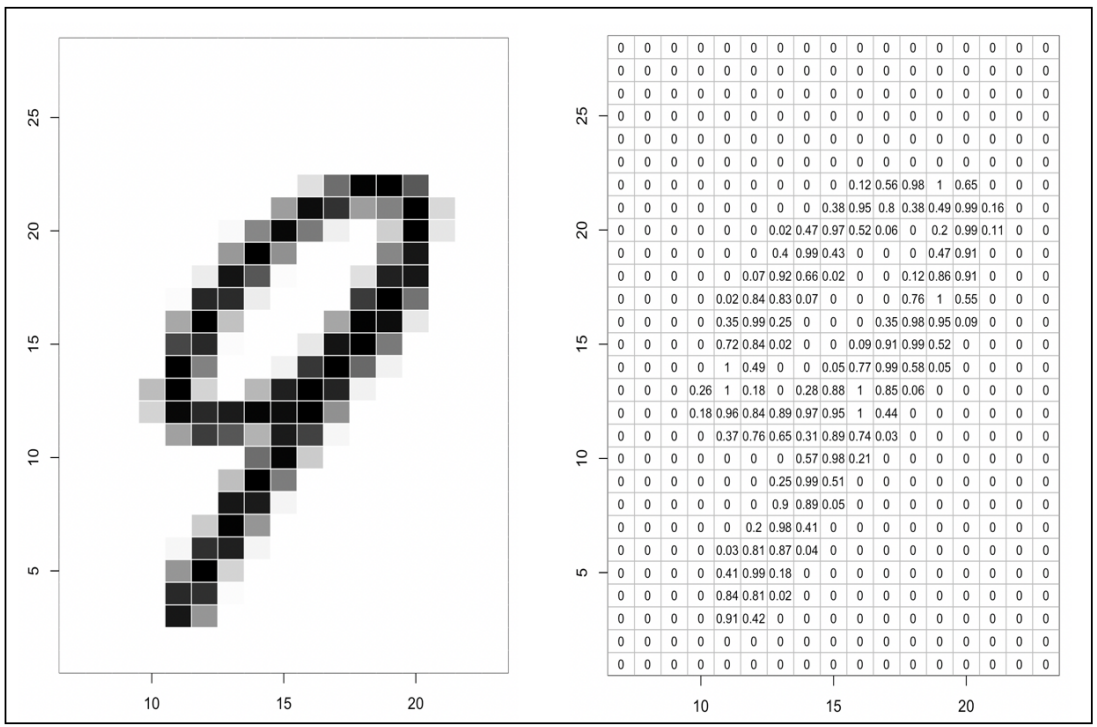
```
install.packages("image.LineSegmentDetector", repos =
    "https://bnosac.github.io/drat")
library(image.LineSegmentDetector)
library(imager)
img_path <- "../grey-lausanne.png"
im <- load.image(img_path)[,,1,1]
linesegments <-  image_line_segment_detector(im*255)
plot(linesegments)
```

or

```
library(image.ContourDetector)
contourlines <-  image_contour_detector(im*255)
plot(contourlines)
```

Picture Classifier

Classical $\{0, 1, \cdots, 9\}$ handwritten digit recognition



$$\boldsymbol{x}_i \in \mathcal{M}_{28,28} \qquad y_i \in \{0, 1, \cdots, 9\}$$

see tensorflow (Google) and keras.

Picture Classifier

See https://www.tensorflow.org/install

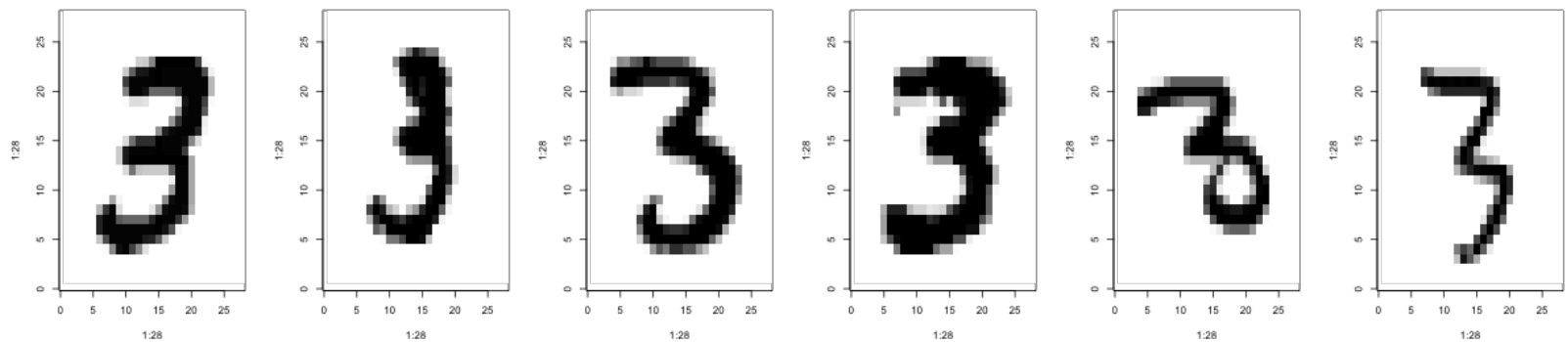
```
library(tensorflow)
install_tensorflow(method = "conda")
library(keras)
install_keras(method = "conda")
```

```
mnist <- dataset_mnist()
idx123 = which(mnist$train$y %in% c(1,2,3))
V <- mnist$train$x[idx123[1:800],,]
MV = NULL
for(i in 1:800) MV=cbind(MV,as.vector(V[i,,]))
MV=t(MV)
df=data.frame(y=mnist$train$y[idx123][1:800],x=MV)
reg=glm((y==1)~.,data=df,family=binomial)
```

Here $\boldsymbol{x}_i \in \mathbb{R}^{784}$ (for a small greyscale picture).

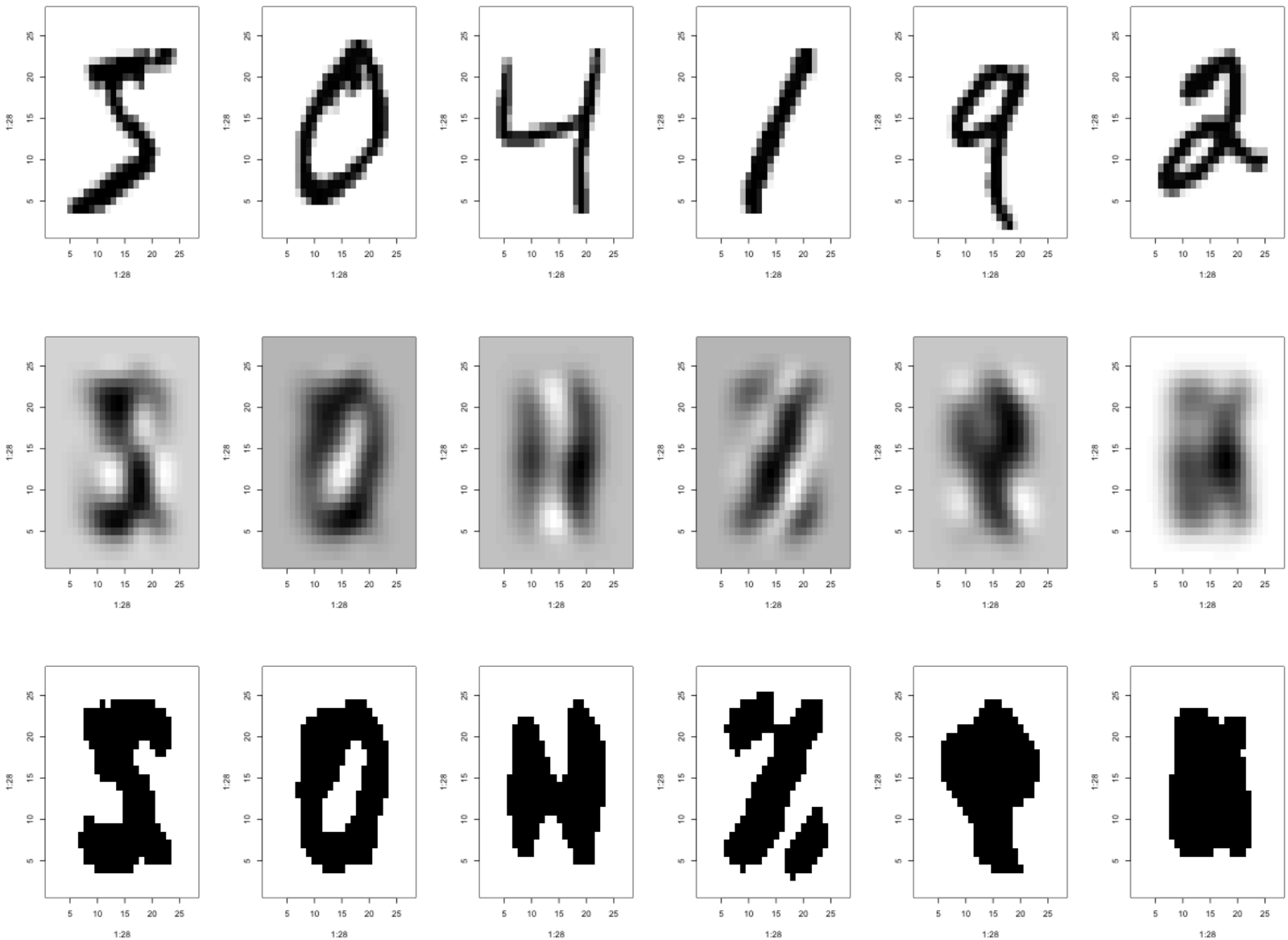
Picture Classifier

Still with those handwritten digit pictures, from the mnist dataset.
Here $\{(y_i, \boldsymbol{x}_i)\}$ with $y_i =$ "3" and $\boldsymbol{x}_i \in [0,1]^{28\times 28}$



One can define a multilinear principal component analysis
see also Tucker decomposition, from Hitchcock (1927, The expression of a tensor or a polyadic as a sum of products)
We can use those decompositions to derive a simple classifier.

Picture Classifier



Picture Classifier

To reduce dimension use (classical) PCA

```
V <- (mnist$train$x[1:1000,,])
MV <- NULL
for(i in 1:1000) MV <- cbind(MV,as.vector(V[i,,]))
pca <- prcomp(t(MV))
```

or Multivariate PCA

```
library(rTensor)
T <- as.tensor(mnist$train$x[1:1000,,])
tensor_svd <- hosvd(T)
tucker_decomp <- tucker(T, ranks = c(100, 3, 3))
T_approx <- ttl(tucker_decomp$Z, tucker_decomp$U, 1:3)
pca <- T_approx@data
```

or directly

```
pca <- mpca(T)
```

Picture Classifier

with the (classical) PCA

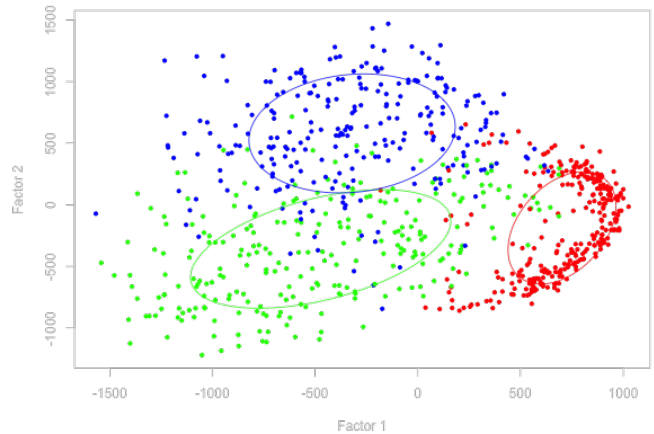
```
library(factoextra)
res.ind <- get_pca_ind(pca)
PTS <- res.ind$coord
value <- mnist$train$y[idx123][1:1000]
```

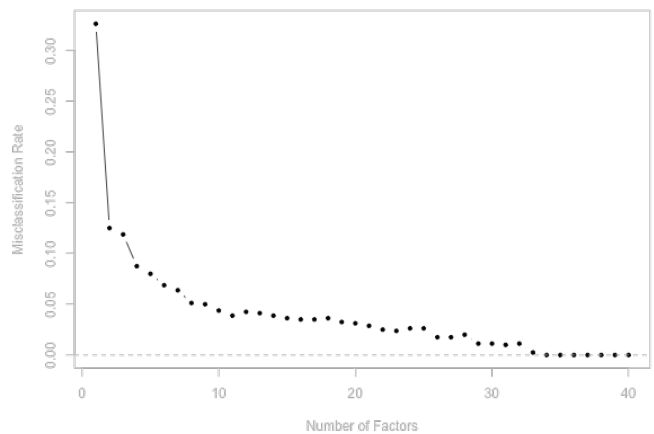
Consider a multinomial logistic regression, on the first k components

```
k <- 10
df <- data.frame(y=as.factor(value), x=PTS[,1:k])
library(nnet)
reg <- multinom(y~.,data=df,trace=FALSE)
df$pred <- predict(reg,type="probs")
```

Picture Classifier

Consider a dataset with only three labels, $y_i \in \{1, 2, 3\}$ and the k principal components obtained by PCA,
$\{x_1, \cdots, x_{784}\} \rightarrow \{\tilde{x}_1, \cdots, \tilde{x}_k\}$
E.g. scatterplot $\{\tilde{x}_{1,i}, \tilde{x}_{2,i}\}$
with • when $y_i = 1$, • $y_i = 2$ and • $y_i = 3$,
Let m_k denote the multinomial logistic regression
on $\tilde{x}_1, \cdots, \tilde{x}_k$ and visualize the misclassification rate





But one can also use some neural network model,
see Nielsen (2018, Neural Networks and Deep Learning)

Neural Nets

Consider optimization problem

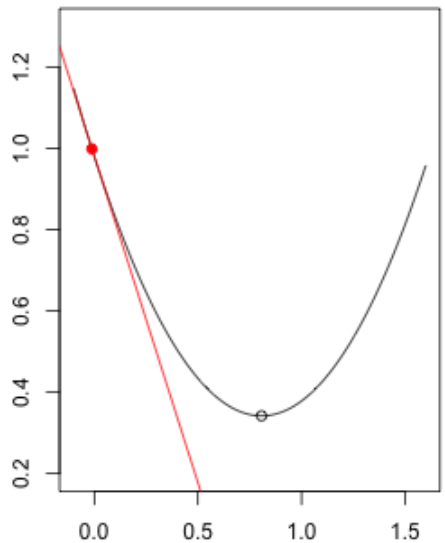
$$\min_{\boldsymbol{x}\in\mathcal{X}} \{f(\boldsymbol{x})\}$$

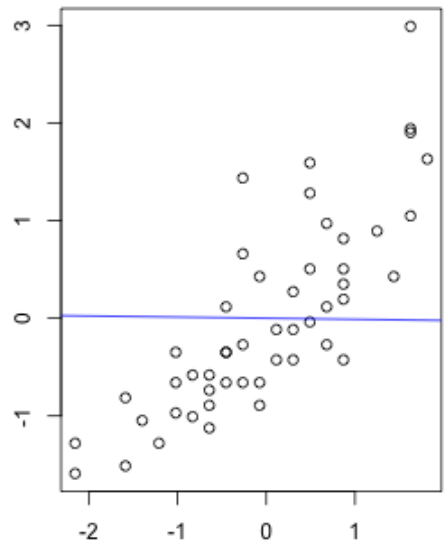
solved by gradient descent,

$$\boldsymbol{x}_{n+1} = \boldsymbol{x}_n - \gamma_n \nabla f(\boldsymbol{x}_n), \quad n \geq$$

with starting point $\boldsymbol{x}_0$

γ_n can be call learning rate





Sometime the dimension of the dataset can be really big: we can't pass all the data to the computer at once to compute $f(\boldsymbol{x})$

we need to divide the data into smaller sizes and give it to our computer one by one

Batches

The Batch size is a hyperparameter that defines the number of sub-samples we use to compute quantities (e.g. the gradient).

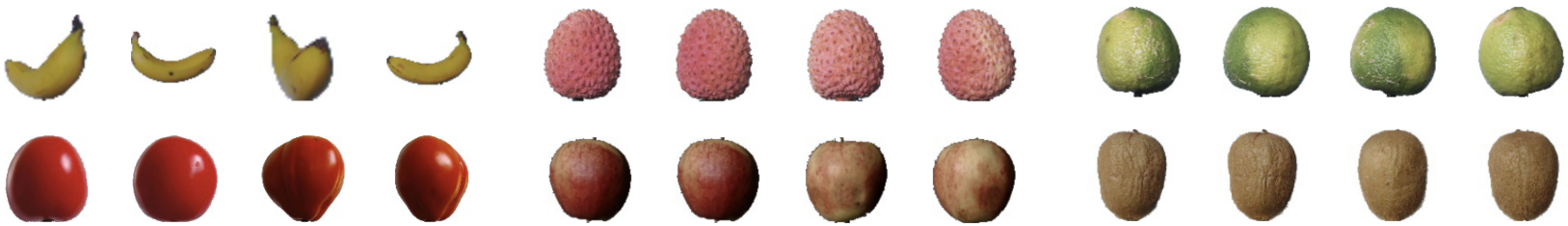
Epoch

One Epoch is when the dataset is passed forward and backward through the neural network (only once).

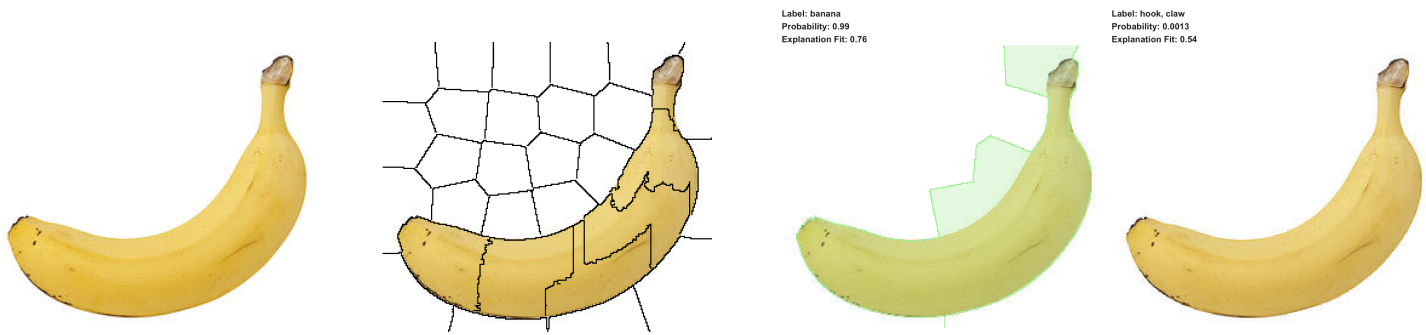
We can divide the dataset of $2,000$ examples into batches of 400 then it will take 5 iterations to complete 1 epoch.
The number of epochs is the hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset.

Picture Classifier

The algorithm is trained on tagged picture, i.e. $(\boldsymbol{x}_i, y_i)$ where here y_i is some class of fruits (banana, apple, kiwi, tomatoe, lime, cherry, pear, lychee, papaya, etc)



The challenge is to provide a new picture ($\boldsymbol{x}_{n+1}$ and to see the prediction $\widehat{y}_{n+1}$)



See fruits-360/Training, e.g. the banana (118)

Picture Classifier

```
library(keras)
library(lime)
library(magick)
model <- application_vgg16(weights = "imagenet",
    include_top = TRUE)
test_image_files_path <- "/fruits-360/Test"
img <- image_read('https://upload.wikimedia.org/
    wikipedia/commons/thumb/8/8a/Banana-Single.jpg/272
    px-Banana-Single.jpg')
img_path <- file.path(test_image_files_path, "Banana",
     'banana.jpg')
image_write(img, img_path)
plot(as.raster(img))
plot_superpixels(img_path, n_superpixels = 35, weight
    = 10)
```

Our testing picture is from ://upload.wikimedia.org/

Picture Classifier

We use here (pre-trained) VGG16 and VGG19 models for Keras, see Simonyan & Zisserman (2014, Very Deep Convolutional Networks for Large-Scale Image Recognition)

```
image_prep <- function(x) {
  arrays <- lapply(x, function(path) {
    img <- image_load(path, target_size = c(224,224))
    x <- image_to_array(img)
    x <- array_reshape(x, c(1, dim(x)))
    x <- imagenet_preprocess_input(x)
  })
  do.call(abind::abind, c(arrays, list(along = 1)))
}
res <- predict(model, image_prep(img_path))
```

To create our own model, see Chollet & Allaire (2018, Deep Learning with R) [github]

Picture Classifier

```
imagenet_decode_predictions(res)
[[1]]
  class_name class_description       score
1  n07753592            banana 0.9929747581
2  n03532672              hook 0.0013420789
3  n07747607            orange 0.0010816196
4  n07749582             lemon 0.0010625814
5  n07716906  spaghetti_squash 0.0009176208
```

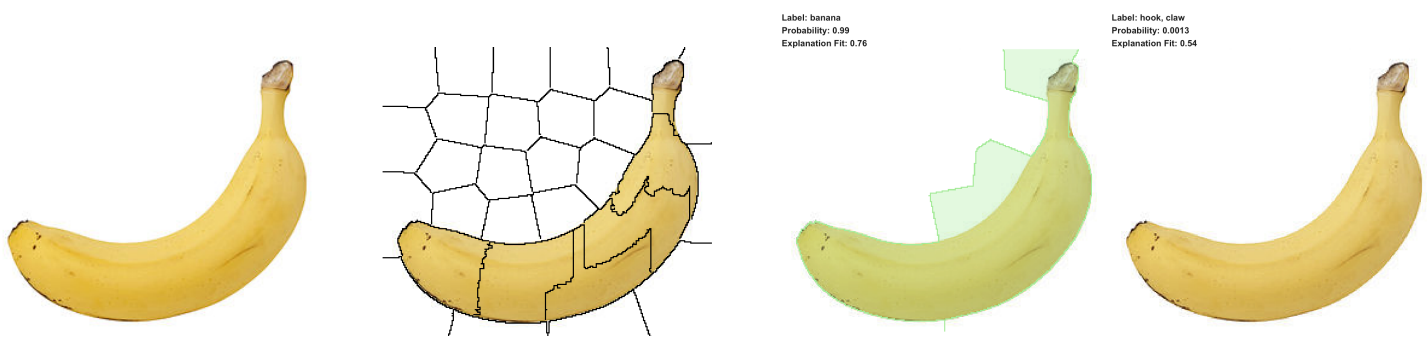
with 99.29% chance, the wikipedia picture of a banana is a *banana*

Picture Classifier

With the lime package - Local Interpretable Model-Agnostic Explanations we can also get some explaination of *why*

```
model_labels <- readRDS(system.file('extdata', '
    imagenet_labels.rds', package = 'lime'))
explainer <- lime(img_path, as_classifier(model,
    model_labels), image_prep)
explanation <- explain(img_path, explainer,
            n_labels = 2, n_features = 35,
            n_superpixels = 35, weight = 10,
            background = "white")
plot_image_explanation(explanation)
```

(see also the vignette or the kitten example)



Picture Classifier

We can build a class activation map, http://gradcam.cloudcv.org see Selvaraju *et al.* (2016, Visual Explanations from Deep Networks via Gradient-based Localization)

Compute the gradient of the score for class c, y_c (before softmax), with respect to feature maps A^k of convolutional layer, $\frac{\partial y_c}{\partial A^k}$

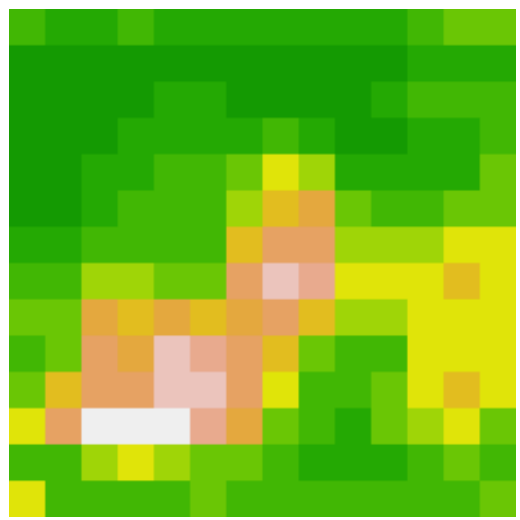
Define the importance weights as an average of gradients

$$\alpha_{c,k} = \frac{1}{n_x n_y} \sum_i \sum_j \frac{\partial y_c}{\partial A^k_{i,j}}$$

and define

$$\text{relu}\left(\sum_k \alpha_{c,k} A^k \right)$$

On a 14×14 grid, we obtain the picture on the right



Picture Classifier

```
img_path <- "car-accident.png"
img <- image_load(img_path,
 target_size = c(224,224)) %>%
 image_to_array() %>%
 array_reshape(dim =
       c(1,224,224,3)) %>%
 imagenet_preprocess_input()
```

Training pictures are $224 \times 224 \times 3$ tensors

Convert to $224 \times 224 \times 3$ size

Picture Classifier

```
model <- application_vgg16(weights = "imagenet")
preds <- model %>% predict(img)
```

Using the pre-trained VGG16 network
on imagenet dataset (55,3467,096 pictures)

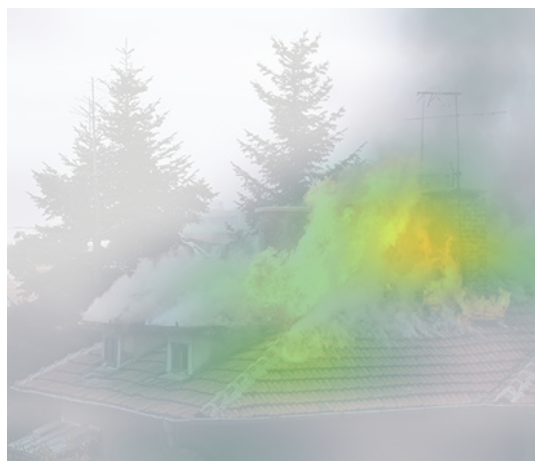
```
imagenet_decode_predictions(preds)
  class_name    class_desc    score
1  n03594945          jeep    0.336
2  n02814533   beach_wagon    0.197
3  n03100240   convertible    0.153
4  n03930630        pickup    0.062
5  n03770679       minivan    0.041
```

Picture Classifier

```
img_path <- "house-fire.png"
```

Using the pre-trained VGG16 network
on imagenet dataset (55,3467,096 pictures)

```
imagenet_decode_predictions(preds)
  class_name class_description score
1  n09472597           volcano 0.414
2  n09288635            geyser 0.320
3  n03773504           missile 0.104
4  n04310018 steam_locomotive 0.051
5  n04008634        projectile 0.045
```

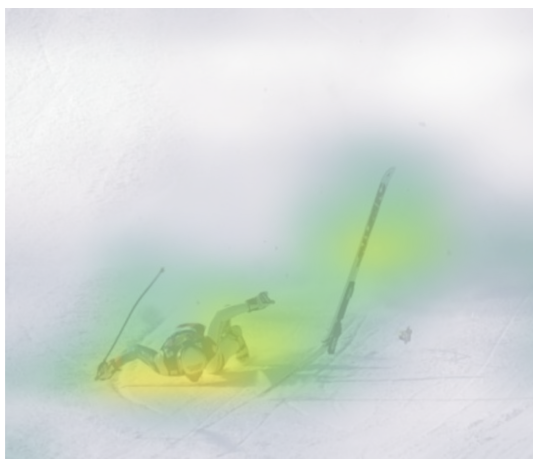


Picture Classifier

```
img_path <- "ski-fall.jpg"
```

Using the pre-trained VGG16 network
on imagenet dataset (55,3467,096 pictures)

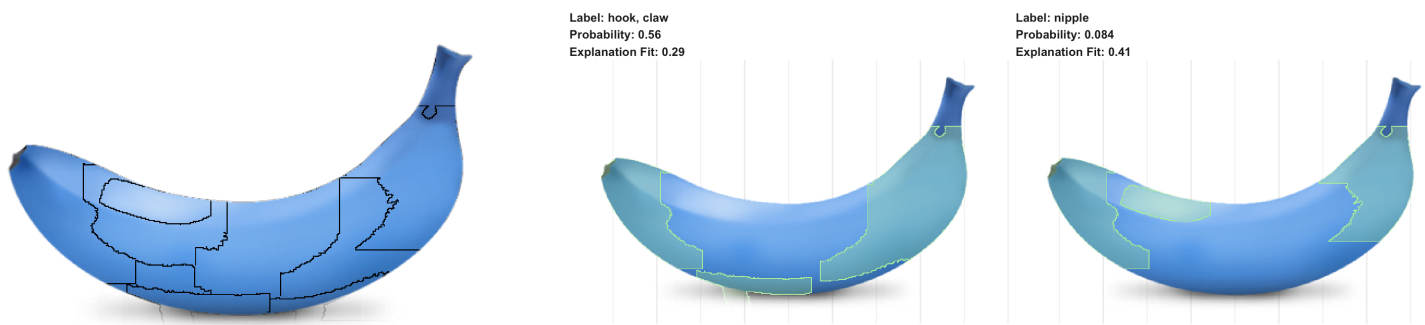
```
imagenet_decode_predictions(preds)
  class_name class_description score
1  n04228054              ski 9.971e-01
2  n09193705              alp 2.449e-03
3  n04252077        snowmobile 8.625e-05
4  n02860847           bobsled 5.331e-05
5  n03218198           dogsled 4.959e-05
```



Picture Classifier

Note that ($\boldsymbol{x}_{n+1}$ can be something that was not in the training dataset (here yellow zucchini)

or some invented one (here a blue banana)



see also pyimagesearch for food/no food classifier

Picture Classifier

To train our own neural networks, use Chollet & Allaire (2018, Deep Learning with R) [github]'s code (with keras)

```
channels <- 3
img_width <- 20
img_height <- 20
target_size <- c(img_width, img_height)
train_image_files_path <- "../fruits-360/Training/"
valid_image_files_path <- "../fruits-360/Validation/"
library(keras)
library(lime)
library(magick)
library(ggplot2)
train_samples <- train_image_array_gen$n
valid_samples <- valid_image_array_gen$n
```

Picture Classifier

```
train_data_gen = image_data_generator(
  rescale = 1/255
  # rotation_range = 40,
  # width_shift_range = 0.2,
  # height_shift_range = 0.2,
  # shear_range = 0.2,
  # zoom_range = 0.2,
  # horizontal_flip = TRUE
)
batch_size <- 32
epochs <- 10
```

Picture Classifier

```
train_image_array_gen <- flow_images_from_directory(
    train_image_files_path,
       train_data_gen,
       target_size = target_size,
       class_mode = "categorical",
       classes = fruit_list)
valid_data_gen <- image_data_generator(rescale =
    1/255)
valid_image_array_gen <- flow_images_from_directory(
    valid_image_files_path,
       valid_data_gen,
       target_size = target_size,
       class_mode = "categorical",
       classes = fruit_list)
```

Picture Classifier

```
model <- keras_model_sequential()
# add layers
model %>%
  layer_conv_2d(filter = 32, kernel_size = c(3,3),
    padding = "same", input_shape = c(img_width,
    img_height, channels)) %>%
  layer_activation("relu") %>%

  # Second hidden layer
  layer_conv_2d(filter = 16, kernel_size = c(3,3),
    padding = "same") %>%
  layer_activation_leaky_relu(0.5) %>%
  layer_batch_normalization() %>%

  # Use max pooling
  layer_max_pooling_2d(pool_size = c(2,2)) %>%
  layer_dropout(0.25) %>%
```

Picture Classifier

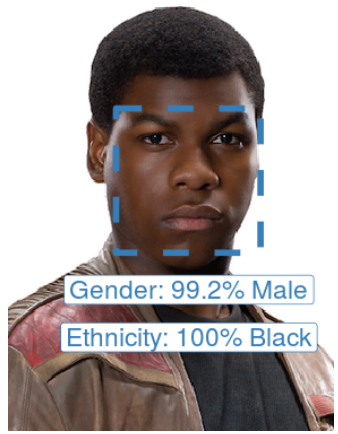
```
15    # Flatten max filtered output into feature vector
16    # and feed into dense layer
17    layer_flatten() %>%
18    layer_dense(100) %>%
19    layer_activation("relu") %>%
20    layer_dropout(0.5) %>%
21
22    # Outputs from dense layer are projected onto output
          layer
23    layer_dense(output_n) %>%
24    layer_activation("softmax")
25
26  # compile
27  model %>% compile(
28    loss = "categorical_crossentropy",
29    optimizer = optimizer_rmsprop(lr = 0.0001, decay = 1
        e-6),
30    metrics = "accuracy"
31  )
```

Picture Classifier

```
hist <- model %>% fit_generator(
  # training data
  train_image_array_gen,
  # epochs
  steps_per_epoch = as.integer(train_samples /
    batch_size),
  epochs = epochs,
  # validation data
  validation_data = valid_image_array_gen,
  validation_steps = as.integer(valid_samples /
    batch_size),
  # print progress
  verbose = 2,
  callbacks = list(
    # save best model after every epoch
    callback_model_checkpoint("fruits_checkpoints.h5",
     save_best_only = TRUE),
    # only needed for visualising with TensorBoard
    callback_tensorboard(log_dir = "logs")))
```

Facial Recognition

For facial recognition, see Schwemmer (2018, facerec: An interface for face recognition in R), based on kairos



or https://github.com/bnosac/image for various R functions (Corner Dor Edges etection, etc).

Picture Classifier

```
install.packages("image.darknet", repos = "https://
    bnosac.github.io/drat")
library(image.darknet)
yolo_tiny_voc <-  image_darknet_model(
  type = 'detect',
  model = "tiny-yolo-voc.cfg",
  weights = system.file(
    package = "image.darknet","models","tiny-yolo-voc.
    weights"),
  labels = c("aeroplane", "bicycle", "bird", "boat", "
    bottle", "bus", "car", "cat",  "chair", "cow", "
    diningtable", "dog", "horse", "motorbike", "person
    ", "pottedplant", "sheep", "sofa", "train", "
    tvmonitor"))
image_darknet_detect(file = "lausanne-d-1.png",
  object =    yolo_tiny_voc)
```

Picture Classifier

```
Loading weights from /Library/Frameworks/R.framework/
    Versions/3.5/Resources/library/image.darknet/
    models/tiny-yolo-voc.weights...Done!
lausanne-d-1.png: Predicted in 0.904149 seconds.
Boxes: 845 of which 6 above the threshold.
person: 82%
person: 47%
person: 61%
person: 60%
person: 54%
person: 40%
```