MATH 4750 / MSSC 5750

Instructor: Mehdi Maadooliat

Deep Learning in R

Fully Connected Neural Network

Convolutional neural network tutorial

Department of Mathematical and Statistical Sciences

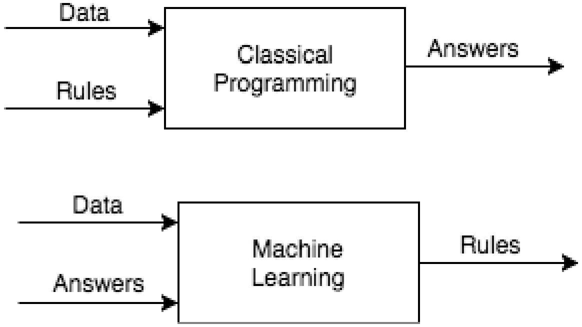
Deep learning: Myths and Truths

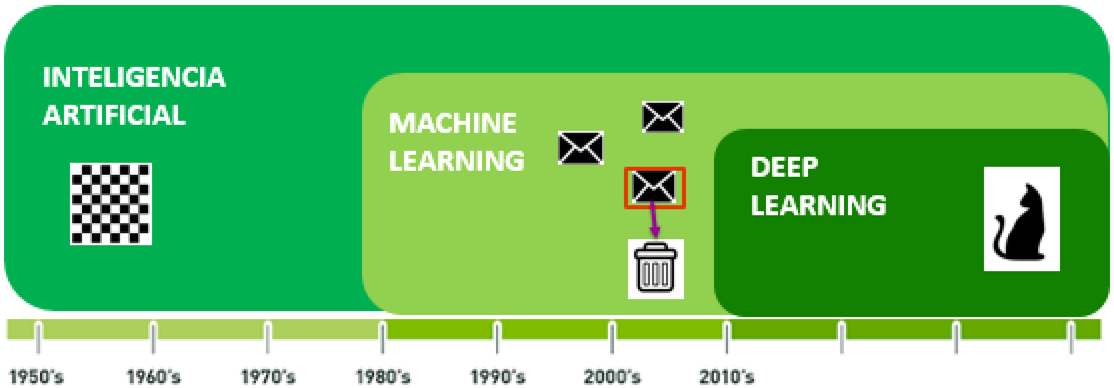
Classical Programming vs Machine Learning

- Deep learning is often presented as algorithms that “work like the brain”, that “think” or “understand”.

  Reality is however quite far from this dream

- AI: *the effort to automate intellectual tasks normally performed by humans*.



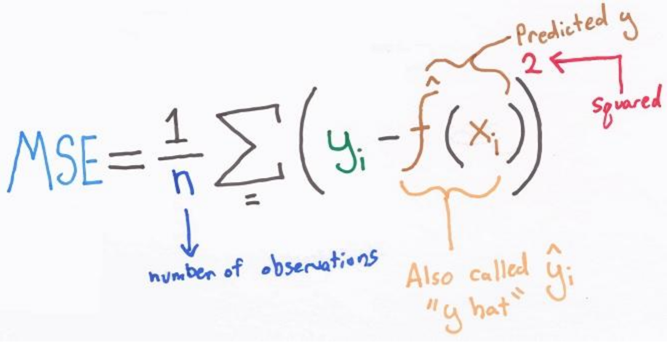


- ML: Could a computer surprise us? Rather than programmers crafting data-processing rules by hand, could a computer automatically learn these rules by looking at data?

MARQUETTE UNIVERSITY
Be The Difference.

RECIPES OF A MACHINE LEARNING ALGORITHM

- *Input data points, e.g.*
  - if the task is speech recognition, these data points could be sound files of people speaking
  - If the task is image tagging, they could be picture files
- *Examples of the expected output*
  - In a speech-recognition task, these could be human-generated transcripts of sound files
  - In an image task, expected outputs could tags such as "dog", "cat", and so on

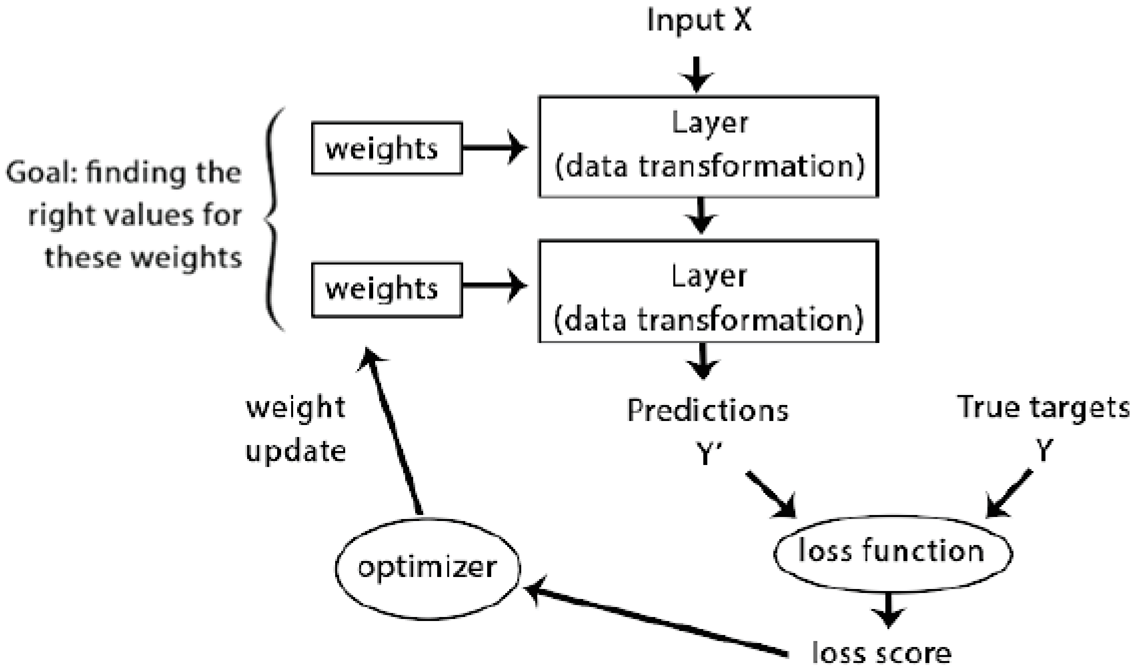


- *A way to measure whether the algorithm is doing a good job*
  - This is necessary in order to determine the distance between the algorithm's current output and its expected output.
  - The measurement is used as a feedback signal to adjust the way the algorithm works. This adjustment step is what we call *learning*.

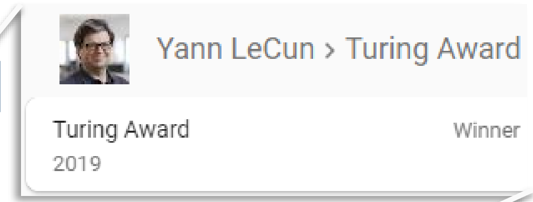
MARQUETTE UNIVERSITY
Be The Difference.

Anatomy of a Neural Network

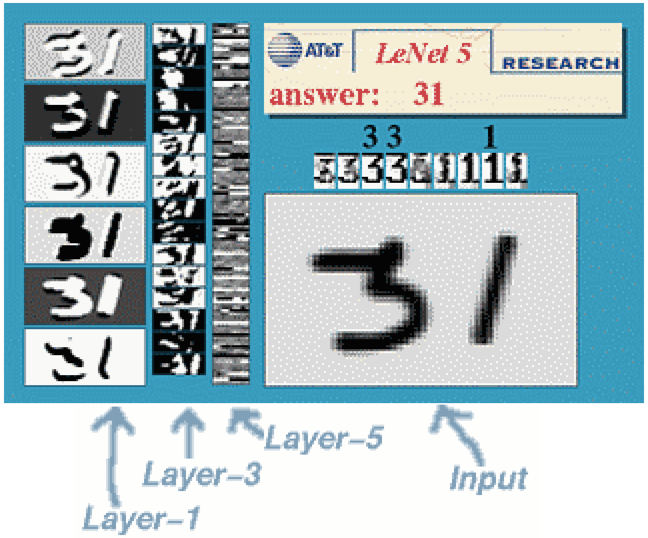
- The *input data* and corresponding *targets*
- *Layers*, which are combined into a *network* (or *model*)
- The *loss function*, which defines the feedback signal used for learning
- The *optimizer*, which determines how learning proceeds



LENET-5: A PIONEERING 7-LEVEL CNN



- The first successful practical application of neural nets came in 1989 from Bell Labs, when Yann LeCun combined the earlier ideas of convolutional neural networks and backpropagation, and applied them to the problem of classifying handwritten digits.
- The resulting network, dubbed *LeNet*, was used by the USPS in the 1990s to automate the reading of ZIP codes on mail envelopes.
- LeNet-5 was applied by several banks to recognize hand-written numbers on checks digitized in 32x32 pixel images.

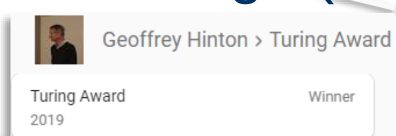


MARQUETTE UNIVERSITY
Be The Difference.

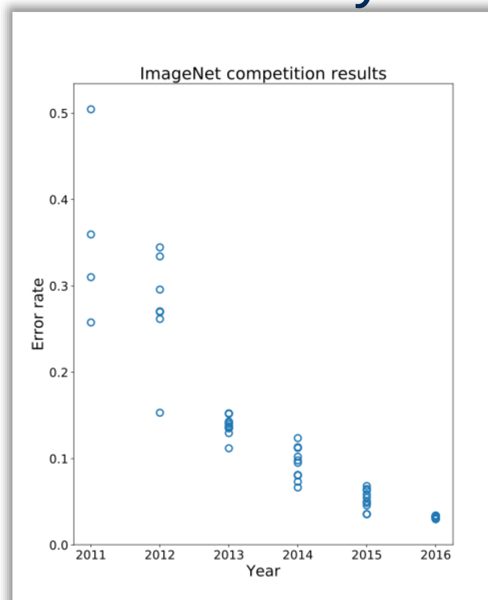
Why 30+ Years Gap?

- In 2011, Dan Ciresan from IDSIA (Switzerland) began to win academic image-classification competitions with GPU-trained deep neural networks

- in 2012, a team led by Alex Krizhevsky and advised by Geoffrey Hinton was able to achieve a top-five accuracy of 83.6%--a significant breakthrough (in 2011 it was only 74.3%).

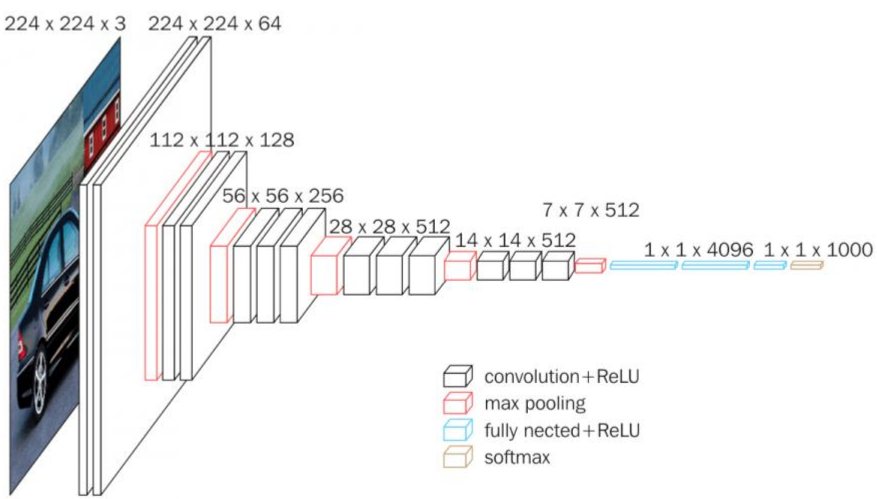


- Three forces are driving advances in ML:
  - Hardware
  - Datasets and benchmarks
  - Algorithmic advances



VGG16 – CNN for Classification and Detection

- VGG16 is a convolutional neural network model proposed by K. Simonyan and A. Zisserman from the University of Oxford
- The model achieves 92.7% top-5 test accuracy in ImageNet. It was one of the famous model submitted to ILSVRC-2014.
- It makes the improvement over AlexNet by replacing large kernel-sized filters (11 and 5 in the first and second convolutional layer, respectively) with multiple 3×3 kernel-sized filters one after another.
- VGG16 was trained for weeks using NVIDIA Titan Black GPU’s.

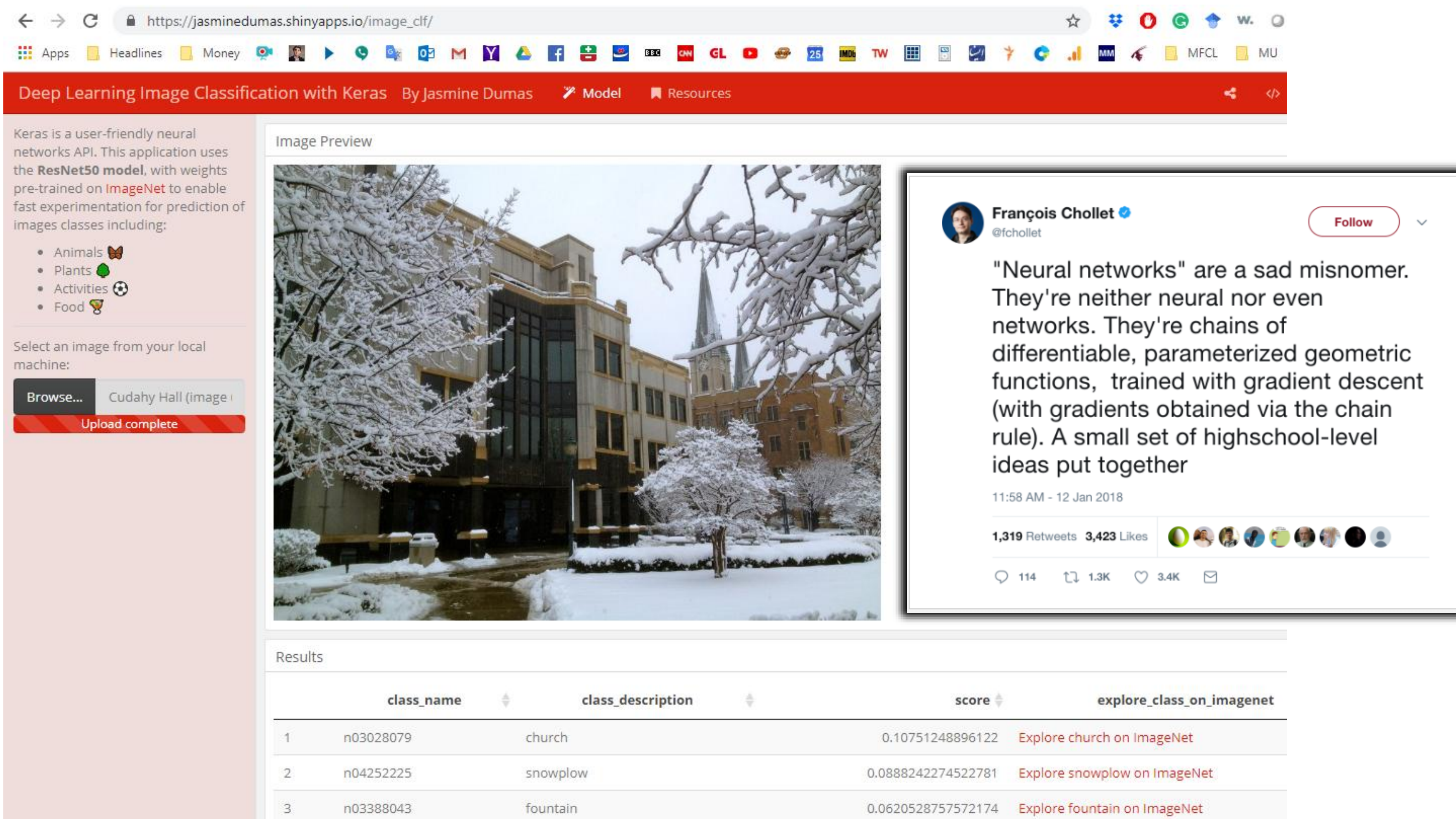



```
summary(vgg16_imagenet_model)

Layer (type)                 Output Shape              Param #
==============================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
--------------------------------------------------------------
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
--------------------------------------------------------------
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
--------------------------------------------------------------
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
--------------------------------------------------------------
... (entire model not shown) ...
--------------------------------------------------------------
fc2 (Dense)                  (None, 4096)              16781312
--------------------------------------------------------------
predictions (Dense)          (None, 1000)              4097000
==============================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
--------------------------------------------------------------
```

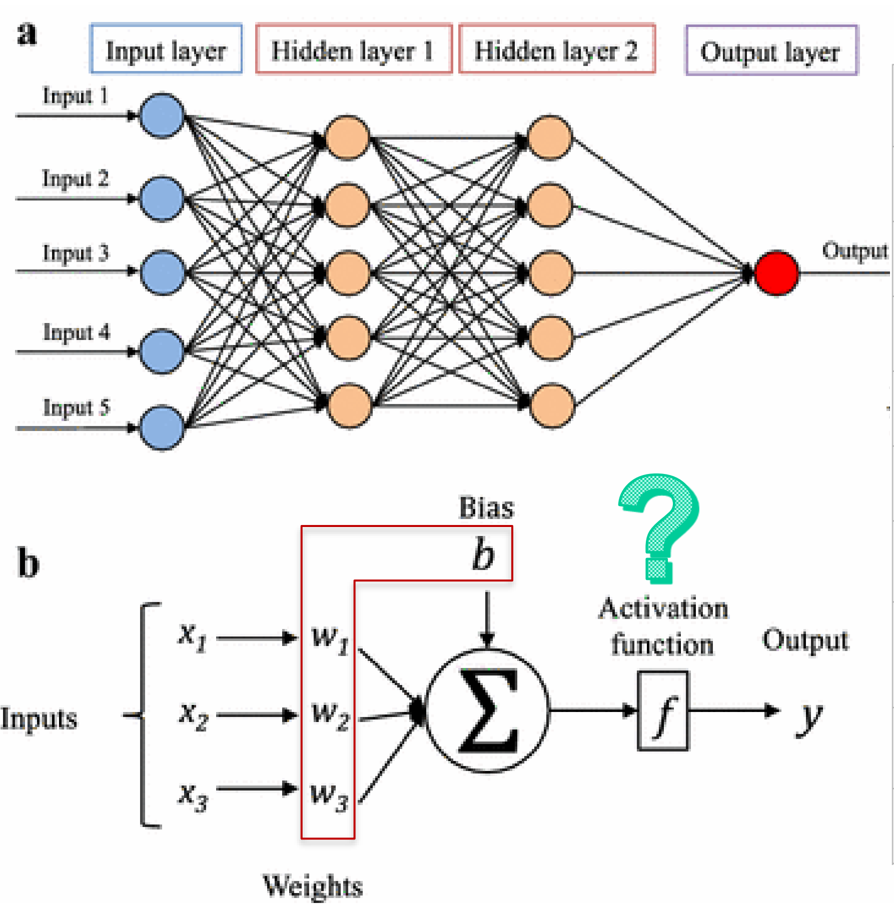
Is Deep Learning really a Black Box?

- Deep Learning Image Classification
  - [AiCSD Image Classification Demo](#).



MARQUETTE UNIVERSITY
Be The Difference.

A NEURAL NETWORK – PARAMETERS- ACTIVATION FUNCTION



Activation functions

Name	Plot	Equation	Derivative
Identity		$f(x) = x$	$f'(x) = 1$
Binary step		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x \neq 0 \\ ? & \text{for } x = 0 \end{cases}$
Logistic (a.k.a Soft step)		$f(x) = \frac{1}{1+e^{-x}}$	$f'(x) = f(x)(1 - f(x))$
TanH		$f(x) = \tanh(x) = \frac{2}{1+e^{-2x}} - 1$	$f'(x) = 1 - f(x)^2$
ArcTan		$f(x) = \tan^{-1}(x)$	$f'(x) = \frac{1}{x^2+1}$
Rectified Linear Unit (ReLU)		$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Parameteric Rectified Linear Unit (PReLU) [2]		$f(x) = \begin{cases} \alpha x & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
Exponential Linear Unit (ELU) [3]		$f(x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$	$f'(x) = \begin{cases} f(x) + \alpha & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$
SoftPlus		$f(x) = \log_e(1 + e^x)$	$f'(x) = \frac{1}{1+e^{-x}}$

LINEAR MODEL – BEST LINEAR UNBIASED EST.

Consider the model

$$Y = X\beta + \epsilon$$

where $Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}$ $X = \begin{pmatrix} 1 & X_{11} & X_{12} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & \dots & X_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & X_{np} \end{pmatrix}$ $\beta = \begin{pmatrix} \beta_0 \\ \beta_1 \\ \vdots \\ \beta_p \end{pmatrix}$ $\epsilon = \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$

Based on this model we get the following expansion for the first subject:

$$Y_1 = \beta_0 + \beta_1 X_{11} + \beta_2 X_{12} + \ldots + \beta_p X_{1p} + \epsilon_1$$

Then using matrix calculus we find that the least squares estimate for β is given by

$$\hat{\beta} = (X^T X)^{-1} X^T Y$$

Hence, the least squares regression line is $\hat{Y} = X\hat{\beta}$.

$$\hat{e} = \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \vdots \\ \hat{e}_n \end{pmatrix} := Y - \hat{Y}$$

In R:

- `lm.fit <- lm(Y~X)`
- `Y.hat <- lm.fit$fitted`
- `e.hat <- lm.fit$resid`

Redidual (Sur)Realism – Reverse Inverse Problem

- The end user provides $\hat{e} = \begin{pmatrix} \hat{e}_1 \\ \hat{e}_2 \\ \vdots \\ \hat{e}_n \end{pmatrix}$ and $\hat{Y} = \begin{pmatrix} \hat{Y}_1 \\ \hat{Y}_2 \\ \vdots \\ \hat{Y}_n \end{pmatrix}$
- Redidual (Sur)Realism provides (generates)

$$Y = \begin{pmatrix} Y_1 \\ Y_2 \\ \vdots \\ Y_n \end{pmatrix}$$

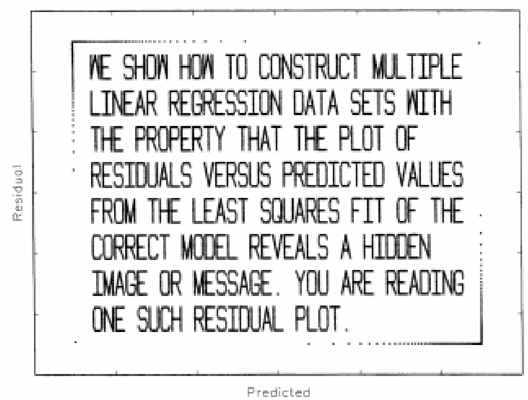
and

$$X = \begin{pmatrix} 1 & X_{11} & X_{12} & \dots & X_{1p} \\ 1 & X_{21} & X_{22} & \dots & X_{2p} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & X_{n1} & X_{n2} & \dots & X_{np} \end{pmatrix}$$

Statistical Computing and Graphics

Residual (Sur)Realism

Leonard A. Stefanski



For example, imagine the reaction of a student who, upon completing an assignment of fitting a given multiple linear regression model and examining residual plots, is confronted with the residual plot in Figure 1(a), which contradicts G.E.P. Box's famous quotation about all models being wrong in the same way that Professor Jones's discovery under the Roman numeral ten contradicted his assertion that **X** never marks the spot (a residual plot version of which appears in Figure 1(b)). Of course, if the regression assignment is due just prior to a "big game," then the student might be more intrigued by residual plots of the sort in Figures 1(c) and (d). Figure 1(e) depicts Homer Simpson explaining how to embed images in regression residual plots. And if the residual plots in (a)–(e) are not attention-getting enough, the student who is unexpectedly confronted with the residual plot in Figure 1(f) is certainly going to be buffaloed (perhaps "bisoned" is taxonomically more correct but not grammatically).

1. INTRODUCTION

REDIDUAL (SUR)REALISM – SIMULATION

- Here $\hat{e}$ and $\hat{Y}$ are vectors of length 10118
- Using "Redidual (Sur)Realism" we generate
- $\boldsymbol{X} = [X_1 \quad X_2 \quad \cdots \quad X_5]$ and Y, where X_j's and Y are vectors of length 10118

For interested readers:
Regression Shiny App
Code to generate similar data
"mu-logo.csv"

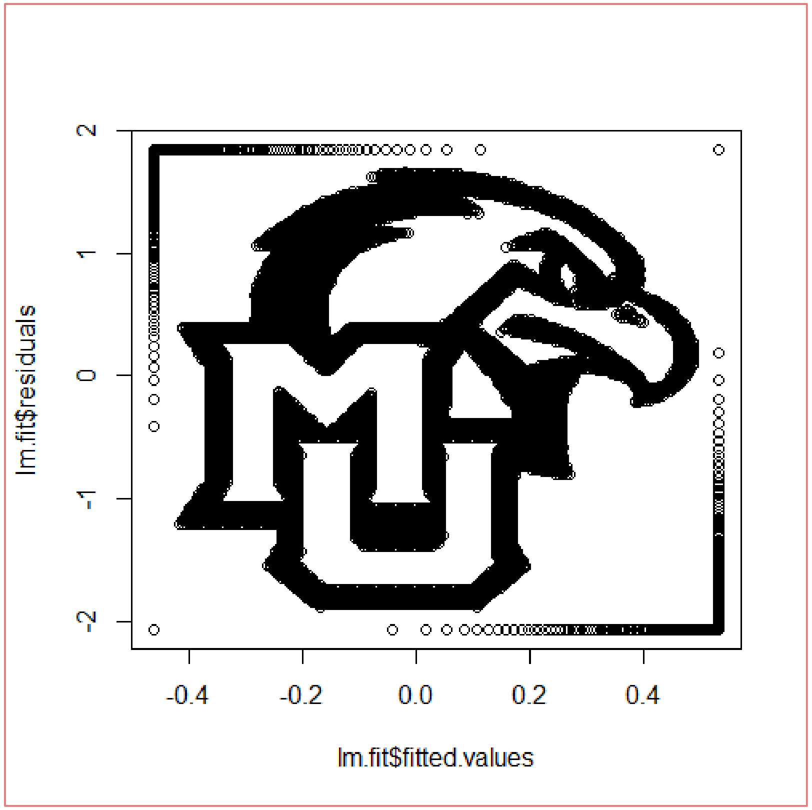
```
> library(dplyr)
> as.tbl(mu.logo)
# A tibble: 10,118 x 6
```

	Y	X1	X2	X3	X4	X5
	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1	1.65	-23.3	-0.487	1.97	-0.198	1.65
2	1.58	6.51	-0.370	-1.54	0.793	-0.803
3	1.59	4.68	0.294	-2.08	-0.0318	0.0216
4	1.59	8.22	-0.435	-2.31	0.470	-0.350
5	1.60	-8.80	1.08	1.73	-1.31	-1.16
6	1.60	-7.05	-1.26	0.912	0.0508	-0.417
7	1.61	-1.06	1.34	-1.16	1.78	0.108
8	1.61	3.98	-0.993	-1.26	1.32	-0.481
9	1.62	0.585	0.0143	-0.446	0.939	-0.730
10	1.62	-10.9	0.182	0.330	2.26	0.709

```
# ... with 10,108 more rows
```

```
➢ mu.logo <- data.frame(read.csv("mu-logo.csv"))
➢ lm.fit <- lm(Y~X1+X2+X3+X4+X5, data=mu.logo)
➢ plot(lm.fit$fitted.values, lm.fit$residuals)
```

Residual plot



$\hat{e}$

$\hat{Y}$

MARQUETTE UNIVERSITY
Be The Difference.

Some Deep Learning Packages in R

R Package	Description
nnet	Software for feed-forward neural networks with a single hidden layer, and for multinomial log-linear models.
neuralnet	Training of neural networks using backpropagation
h2o	R scripting functionality for H2O
RSNNS	Interface to the Stuttgart Neural Network Simulator (SNNS)
tensorflow	Interface to TensorFlow
deepnet	Deep learning toolkit in R
darch	Package for Deep Architectures and Restricted Boltzmann Machines
rnn	Package to implement Recurrent Neural Networks (RRNs)
FCNN4R	Interface to the FCNN library that allows user-extensible ANNs
rcppDL	Implementation of basic machine learning methods with many layers (deep learning), including dA (Denoising Autoencoder), SdA (Stacked Denoising Autoencoder), RBM (Restricted Boltzmann machine) and DBN (Deep Belief Nets)
deepr	Package to streamline the training, fine-tuning and predicting processes for deep learning based on darch and deepnet
MXNetR	Package that brings flexible and efficient GPU computing and state-of-art deep learning to R

Ref: https://www.datacamp.com/community/tutorials/keras-r-deep-learning

Regression using Neural Network

- `library("neuralnet");`
- `net1 <- neuralnet(Y~X1+X2+X3+X4+X5, data=mu.logo, hidden=0, act.fct=function(x) {x})`
- `plot(net1)`

```
> lm.fit

Call:
lm(formula = Y ~ X1 + X2 + X3 + X4 + X5, data = mu.logo)

Coefficients:
(Intercept)           X1           X2           X3           X4           X5
  0.9761206    0.1886051    0.1203780    0.9476638    0.0123327    0.9670491
```

```
net2 <- neuralnet(Y~X1+X2+X3+X4+X5, data=mu.logo,
   hidden=10, act.fct=function(x) {x})
```

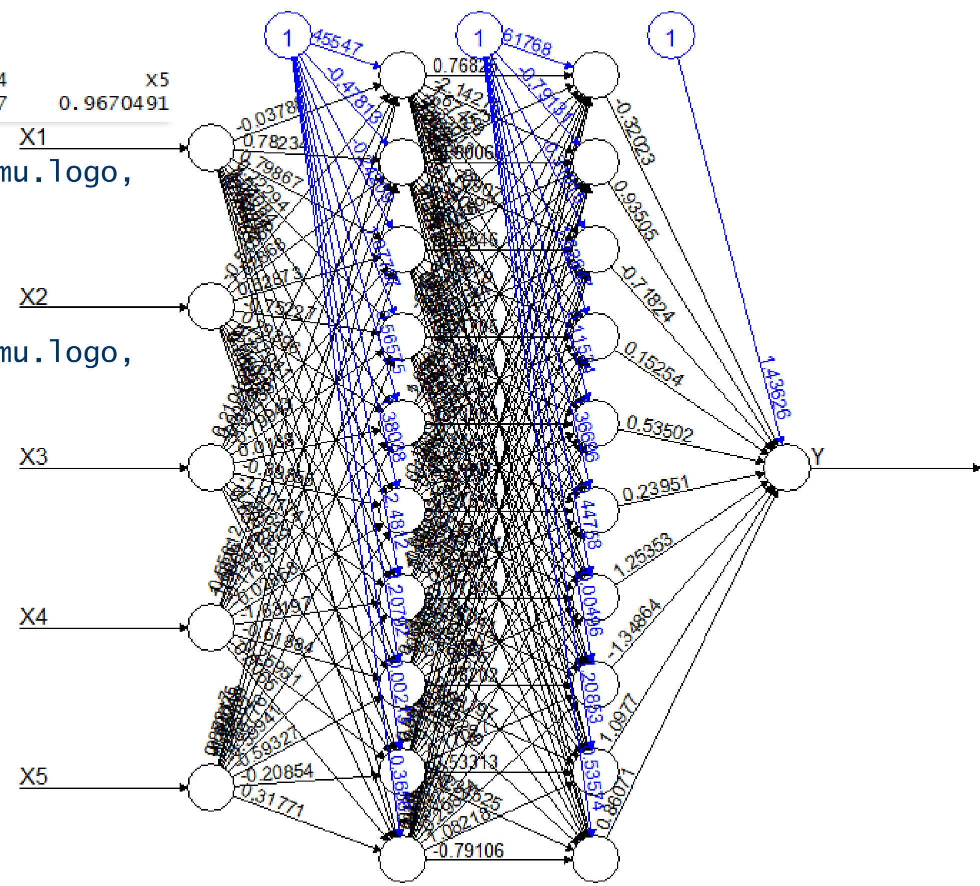
- `plot(net2)`

```
net3 <- neuralnet(Y~X1+X2+X3+X4+X5, data=mu.logo,
         hidden=c(10,10),
         act.fct=function(x) {x})
```

- `plot(net3)`

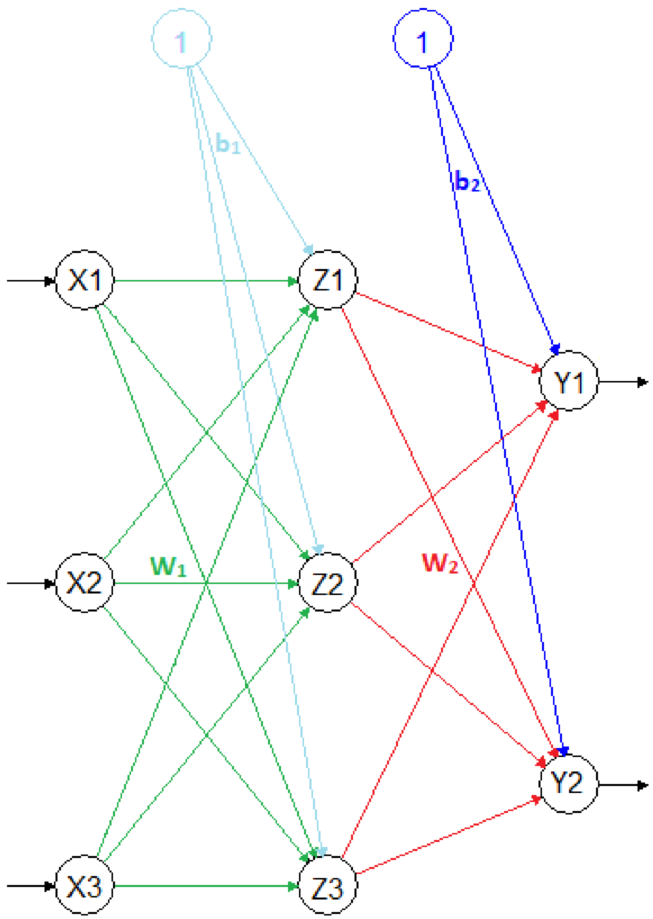
```
errors <- c(net1$result.matrix[1],
            net2$result.matrix[1],
            net3$result.matrix[1])
```

```
> errors
[1] 5058.5 5058.5 5058.5
> net1$err.fct
function (x, y)
{
    1/2 * (y - x)^2
}
```



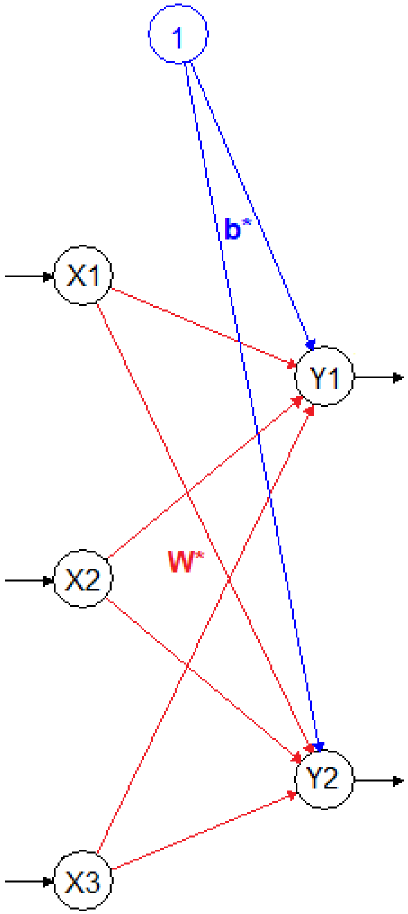

```
> sum(lm.fit$residuals^2)/2
[1] 5058.5
```

Linear Activation Function – Hidden Layers Disappears



- $Z = W_1 X + b_1$
- $Y = W_2 Z + b_2$
- $Y = W_2\{W_1 X + b_1\} + b_2$
  $= \{W_2 W_1\} X + W_2 b_1 + b_2$

LINEAR ACTIVATION FUNCTION – HIDDEN LAYERS DISAPPEARS



- $Z = W_1 X + b_1$
- $Y = W_2 Z + b_2$
- $Y = W_2\{W_1 X + b_1\} + b_2$
  $= \{W_2 W_1\} X + W_2 b_1 + b_2$

- $Y = \boldsymbol{W}^* X + \boldsymbol{b}^*$

MARQUETTE UNIVERSITY
Be The Difference.

Linear Regression using Neural Network

```
> par(mfrow=c(2,2))
> plot(lm.fit$fitted.values,lm.fit$residuals)
> plot(net1$net.result[[1]], net1$data$Y-net1$net.result[[1]])
> plot(net2$net.result[[1]], net1$data$Y-net2$net.result[[1]])
> plot(net3$net.result[[1]], net1$data$Y-net3$net.result[[1]])
```

Track the estimated weights (last slide):

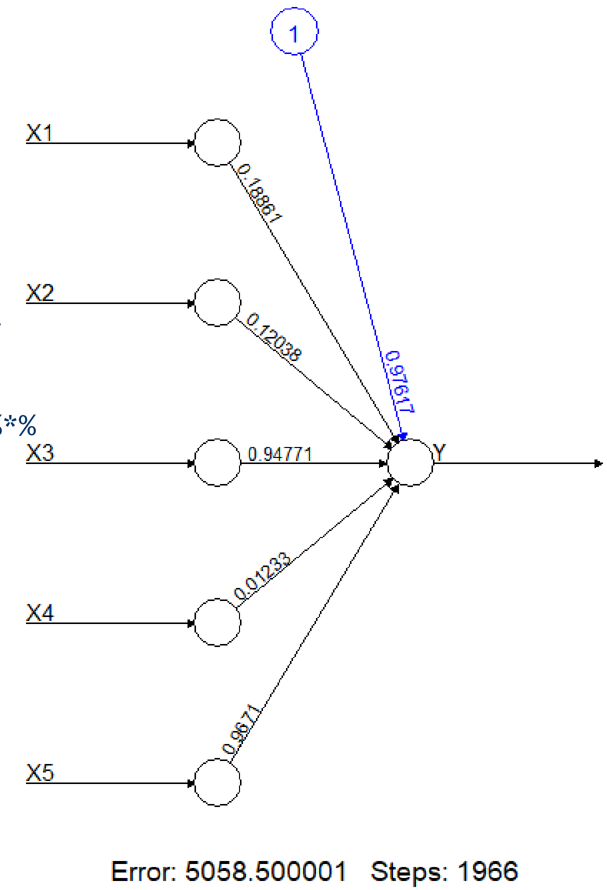
```
> lm.fit$coefficients
(Intercept)        x1        x2       x3        x4        x5
     0.9761    0.1886    0.1204    0.947    0.0123    0.9670

> net1$weights[[1]][[1]]

> net2$weights[[1]][[1]]%*%net2$weights[[1]][[2]][-1,]+
        c(net2$weights[[1]][[2]][1,],rep(0,5))

> net3$weights[[1]][[1]]%*%net3$weights[[1]][[2]][-1,]%*%
        net3$weights[[1]][[3]][-1,]+
        c(net3$weights[[1]][[2]][1,]%*%
        net3$weights[[1]][[3]][-1,]+
        net3$weights[[1]][[3]][1,],rep(0,5))

             [,1]
[1,] 0.9761[illegible]
[2,] 0.18860[illegible]
[3,] 0.1203[illegible]
[4,] 0.9477[illegible]
[5,] 0.012332[illegible]
[6,] 0.9670[illegible]
```



Neural Network

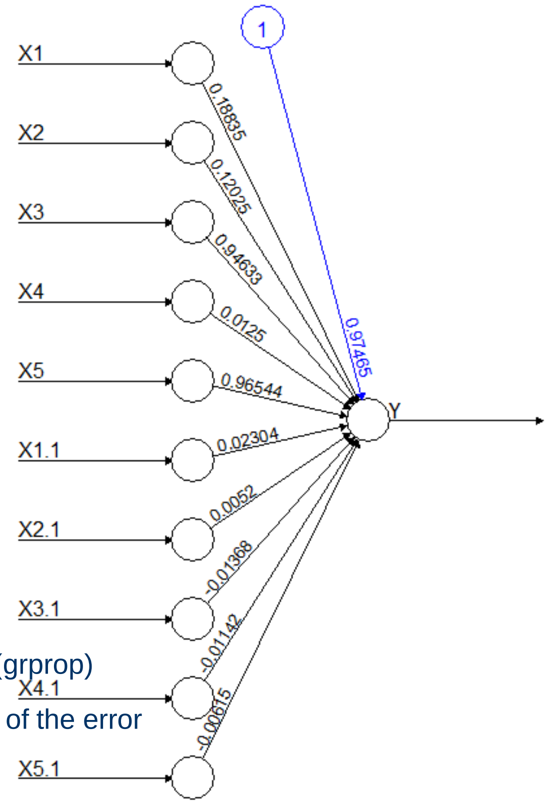
```r
mulogo <- data.frame(cbind(mu.logo, matrix(rnorm(prod(dim(net1$covariate))),nc=5)) )
```

```r
net4 <- neuralnet(Y~X1+X2+X3+X4+X5+X1.1+X2.1+X3.1+X4.1+X5.1, data=mulogo,
        hidden=0,linear.output=TRUE, act.fct=function(x) {x})
```

```r
plot(lm.fit$fitted.values, lm.fit$residuals)
```

```r
plot(net4$net.result[[1]], net4$data$Y-net4$net.result[[1]])
```

```r
plot(net4)
```



Function "neuralnet" Arguments:

```r
neuralnet(formula, data, hidden = 1,
        threshold = 0.01, stepmax = 1e+05, rep = 1,
        startweights = NULL,
        learningrate = NULL, learningrate.limit = NULL,
        algorithm = "rprop+",
        err.fct = "sse",
        act.fct = "logistic",
        exclude = NULL, constant.weights = NULL)
```

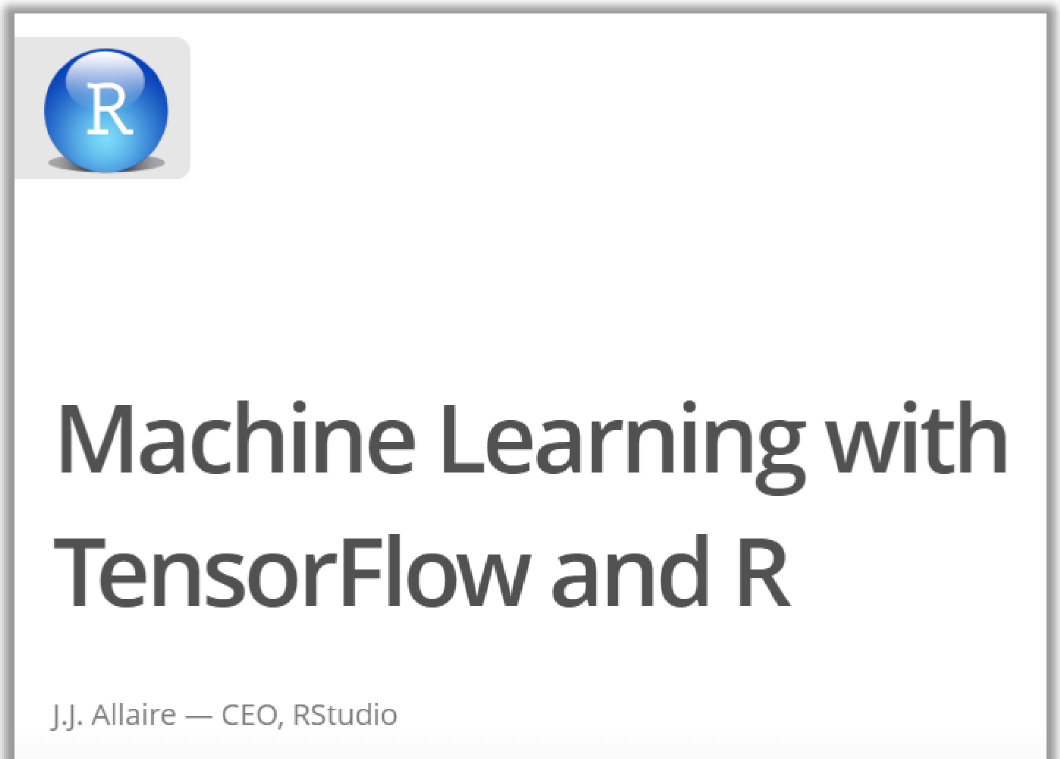
`algorithm` : The following algorithms are possible:

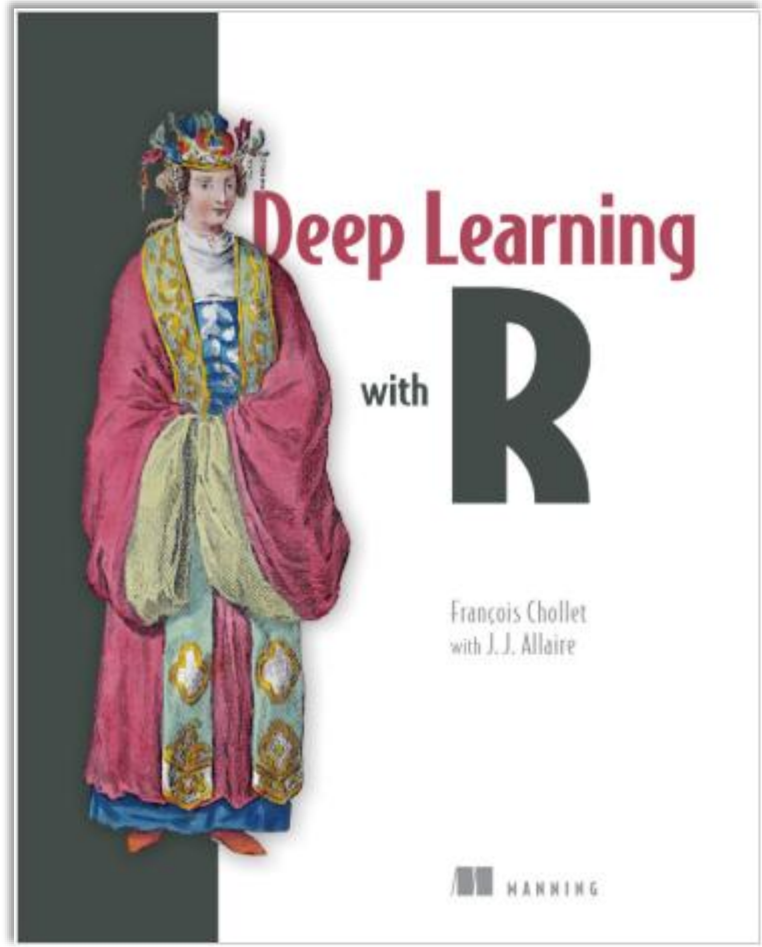
- 'backprop' : backpropagation
- 'rprop+' : the resilient backpropagation with weight backtracking
- 'rprop-' : the resilient backpropagation without weight backtracking
- 'sag' and 'slr' : induce the usage of the modified globally convergent algorithm (grprop)

`err.fct` : 'sse' and 'ce' or a differentiable function that is used for the calculation of the error

`act.fct` : 'logistic' and 'tanh' or a user defined differentiable activation function

Deep Learning with R
TensorFlow – Keras





Comparison of Deep Learning Software

Software	Initial Release	Software license[a]	Open source	Written in	OpenMPsupport	CUDA support	Automatic differentiation[1]	Has pretrained models	Recurrent nets	Convolutional nets	RBM/DBNs	Parallel execution (multi node)	Actively Developed
Wolfram Mathematica	1988	Proprietary	No	C++, Wolfram Language, CUDA	Yes	Yes	Yes	Yes[64]	Yes	Yes	Yes	Under Development	Yes
MATLAB + Neural Network Toolbox		Proprietary	No	C, C++, Java, MATLAB	No	Yes	No	Yes[18][19]	Yes[18]	Yes[18]	No	With Parallel Computing Toolbox[20]	Yes
Microsoft Cognitive Toolkit(CNTK)	2016	MIT license[21]	Yes	C++	Yes[26]	Yes	Yes	Yes[27]	Yes[28]	Yes[28]	No[29]	Yes[30]	Yes
PyTorch	2016	BSD	Yes	Python, C, CUDA	Yes	Yes	Yes	Yes	Yes	Yes		Yes	Yes
Apache MXNet	2015	Apache 2.0	Yes	Small C++core library	Yes	Yes	Yes[36]	Yes[37]	Yes	Yes	Yes	Yes[38]	Yes
Keras	2015	MIT license	Yes	Python	Only if using Theano as backend	Yes	Yes	Yes[15]	Yes	Yes	Yes	Yes[16]	Yes
TensorFlow	2015	Apache 2.0	Yes	C++, Python, CUDA	No	Yes	Yes[46]	Yes[47]	Yes	Yes	Yes	Yes	Yes
Chainer	2015	BSD	Yes	Python	No	Yes	Yes	Yes	Yes	Yes	No	Yes	Yes
Theano	2007	BSD	Yes	Python	Yes	Yes	Yes[49][50]	Through Lasagne's model zoo[51]	Yes	Yes	Yes	Yes[52]	No
Torch	2002	BSD	Yes	C, Lua	Yes	Yes[59][60]	Through Twitter's Autograd[61]	Yes[62]	Yes	Yes	Yes	Yes[63]	No
BigDL	2016	Apache 2.0	Yes	Scala		No		Yes	Yes	Yes			
Caffe	2013	BSD	Yes	C++	Yes	Yes	Yes	Yes[4]	Yes	Yes	No	?	
Neural Designer		Proprietary	No	C++	Yes	No	?	?	No	No	No	?	
OpenNN	2003	GNU LGPL	Yes	C++	Yes	Yes	?	?	No	No	No	?	
Intel Math Kernel Library		Proprietary	No		Yes[13]	No	Yes	No	Yes[14]	Yes[14]		No	
Deeplearning4j	2014	Apache 2.0	Yes	C++, Java	Yes	Yes[6][7]	Computational Graph	Yes[8]	Yes	Yes	Yes	Yes[9]	
Intel Data Analytics Acceleration Library	2015	Apache License 2.0	Yes	C++, Python, Java	Yes	No	Yes	No		Yes		Yes	
Dlib	2002	Boost Software License	Yes	C++	Yes	Yes	Yes	Yes	No	Yes	Yes	Yes	
Apache SINGA	2015	Apache 2.0	Yes	C++	No	Yes	?	Yes	Yes	Yes	Yes	Yes	

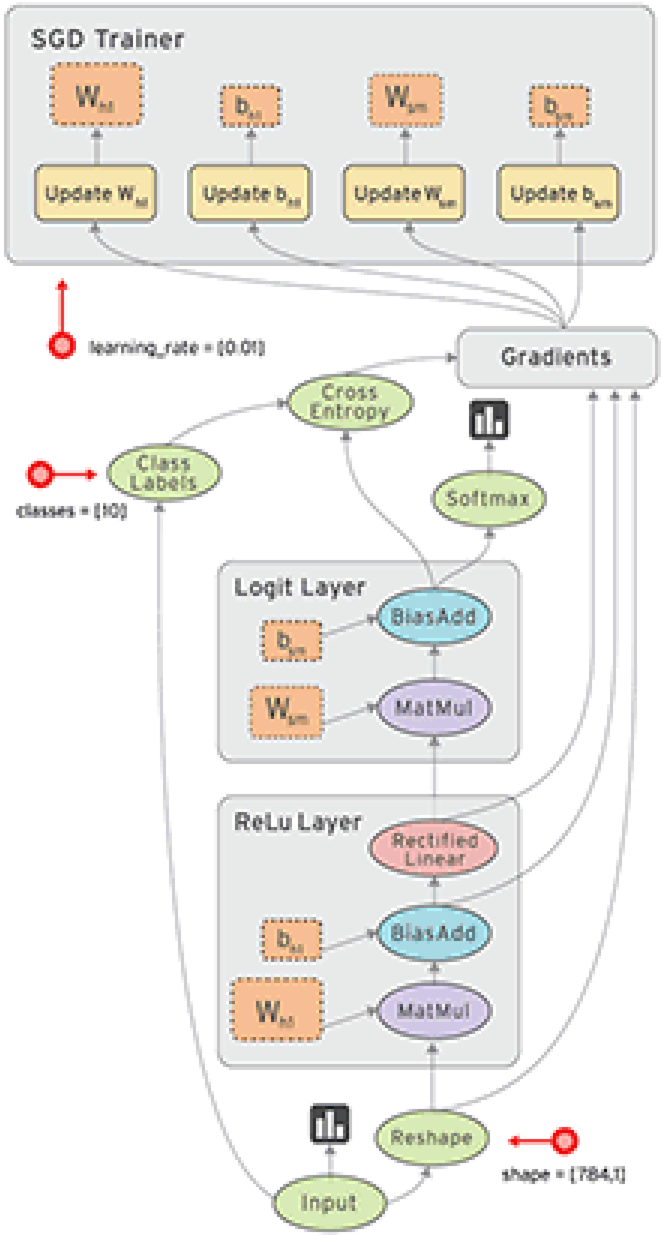
- Source: Wikipedia

WHY TENSORFLOW IN R?

- Hardware independent
  - CPU (via Eigen and BLAS)
  - GPU (via CUDA and cuDNN)
  - TPU (Tensor Processing Unit)
- Supports automatic differentiation
- Distributed execution and large datasets
- Very general built-in optimization algorithms (SGD, Adam) that don't require that all data is in RAM
- TensorFlow models can be deployed with a low-latency C++ runtime
- R has a lot to offer as an interface language for TensorFlow

WHAT IS TENSOR "FLOW"?

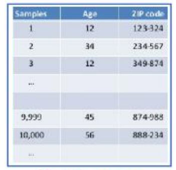
- You define the graph in R
- Graph is compiled and optimized
- Graph is executed on devices
- Nodes represent computations
- Data (tensors) flows between them



MARQUETTE UNIVERSITY
Be The Difference.

Real-World Examples of Data Tensors

■ 2D tensors



Samples	Age	ZIP code
1	12	123324
2	34	234567
3	12	349874
...		
9,999	45	874988
10,000	56	888234
...		

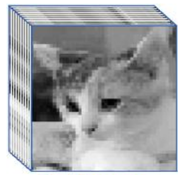
2-d tensor

– Vector data—(samples, features)

```
head(data.matrix(iris), n = 10)
```

```
      Sepal.Length Sepal.Width Petal.Length Petal.Width Species
 [1,]          5.1         3.5          1.4         0.2       1
 [2,]          4.9         3.0          1.4         0.2       1
 [3,]          4.7         3.2          1.3         0.2       1
 [4,]          4.6         3.1          1.5         0.2       1
 [5,]          5.0         3.6          1.4         0.2       1
 [6,]          5.4         3.9          1.7         0.4       1
 [7,]          4.6         3.4          1.4         0.3       1
 [8,]          5.0         3.4          1.5         0.2       1
 [9,]          4.4         2.9          1.4         0.2       1
[10,]          4.9         3.1          1.5         0.1       1
```

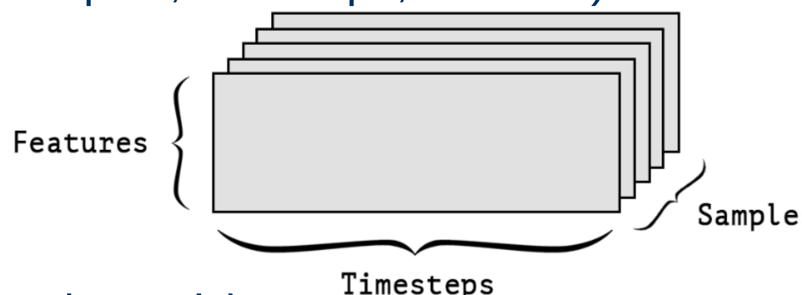
■ 3D tensors



3-d tensor

– Grayscale Images—(samples, height, width)

– Time-series data or sequence data—(samples, timesteps, features)

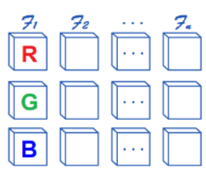


■ 4D tensors

4-d tensor

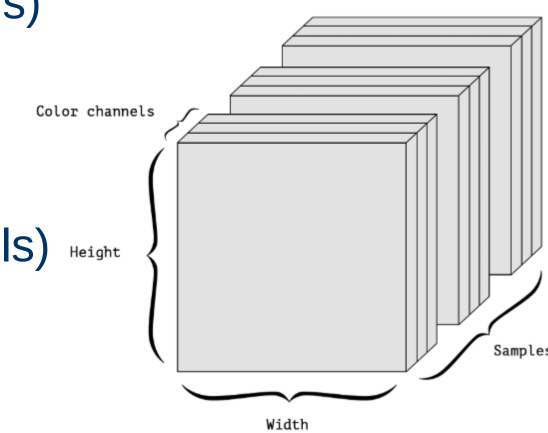
– Color Images—(samples, height, width, channels)

■ 5D tensors



5-d tensor

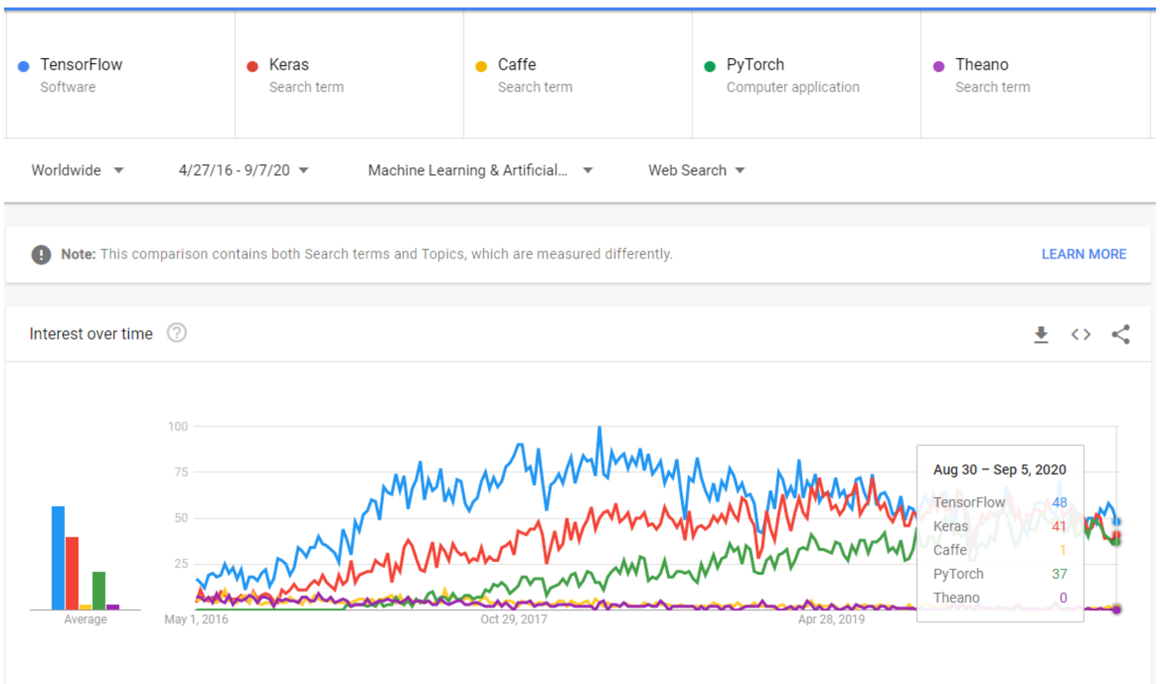
– Video—(samples, frames, height, width, channels)

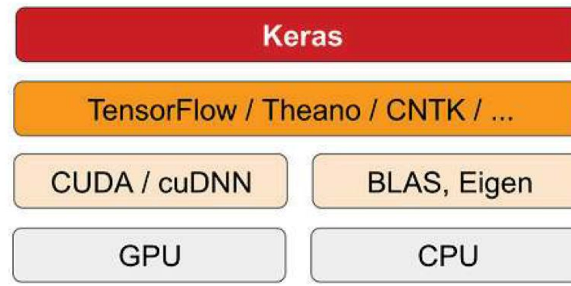


MARQUETTE UNIVERSITY
Be The Difference.

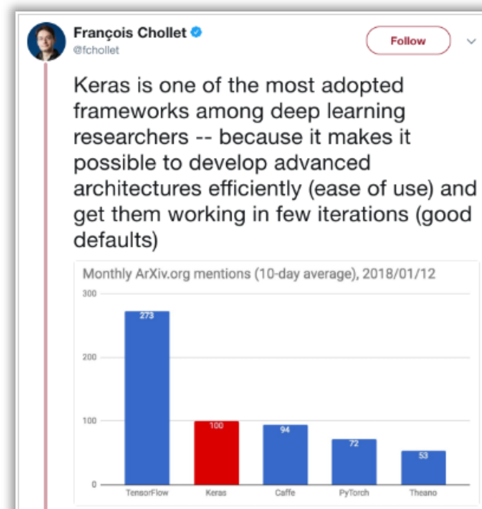
Why Keras?

- It allows the same code to run seamlessly on CPU or GPU.
- It has a user-friendly API that makes it easy to quickly prototype deep-learning models.





The deep-learning software and hardware stack



INSTALLING KERAS

- First, install the keras R package:
  - remotes::install_github("rstudio/keras3") OR
  - Install.packages("keras3")

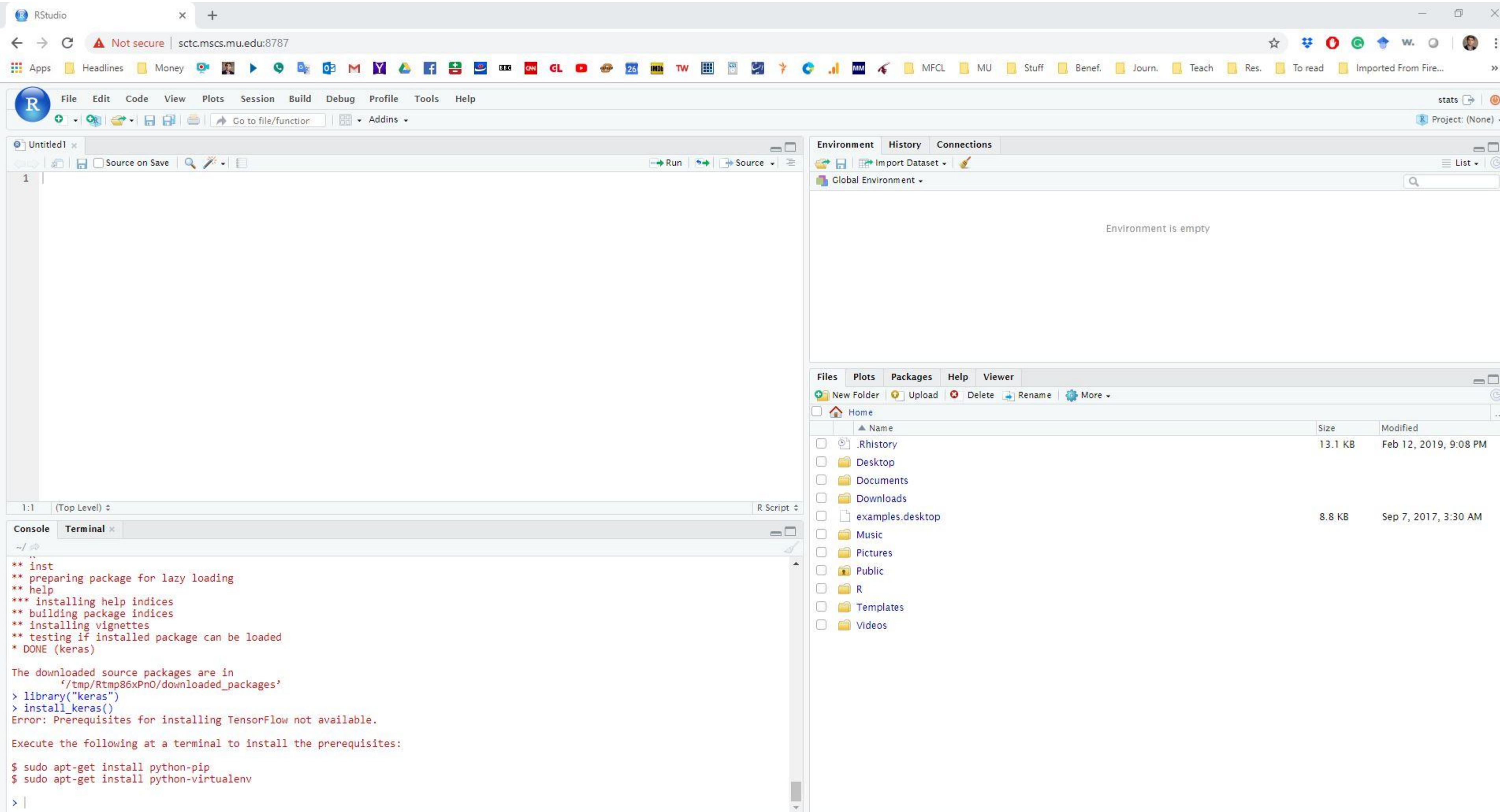
- To install both the core Keras library as well as the TensorFlow backend
  - library(keras3)
  - keras3::install_keras(backend = "tensorflow")

- You need Python installed before installing TensorFlow
  - Anaconda (Python distribution), a free and open-source software

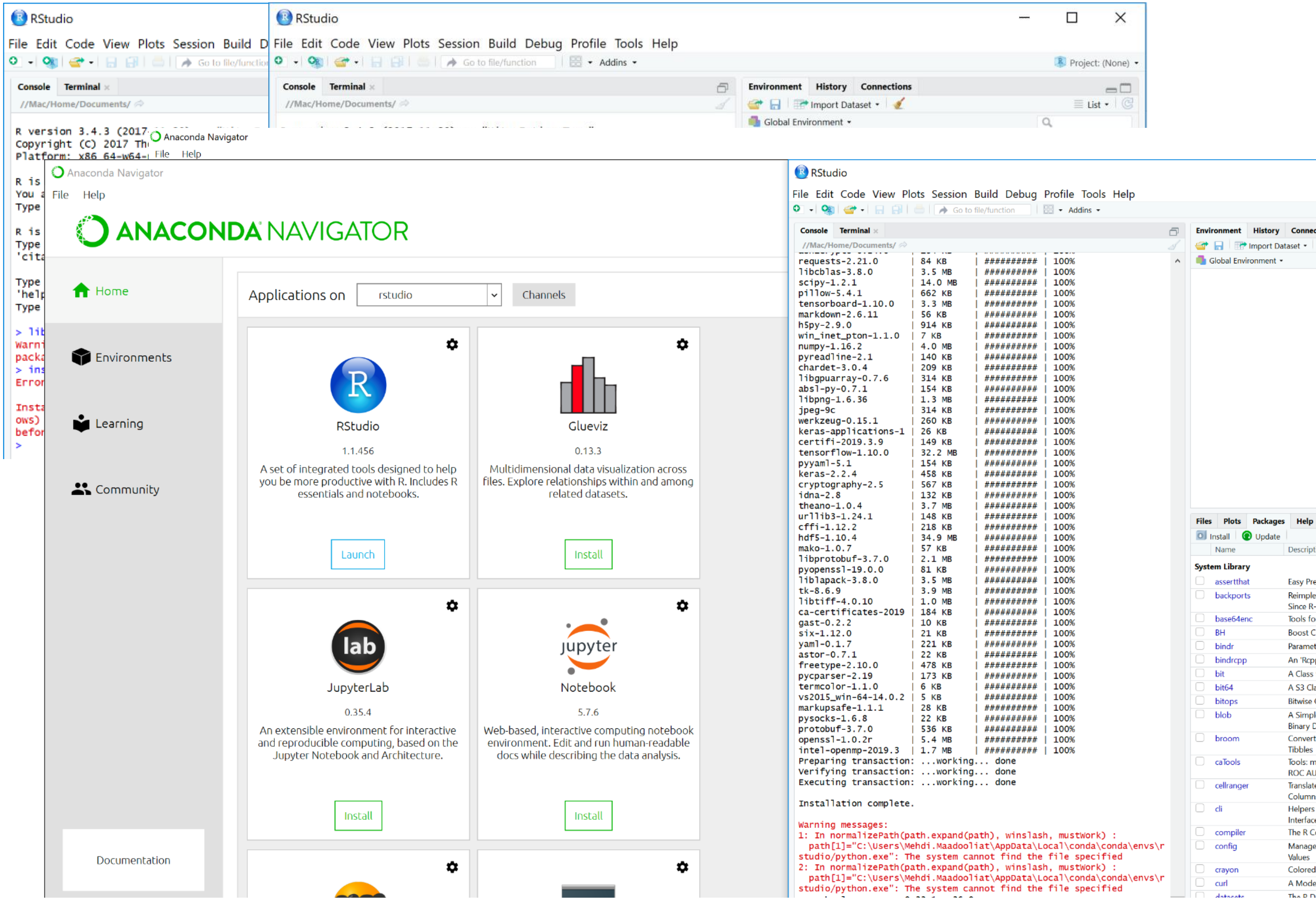
- You can install TensorFlow with GPU support
  - required NVIDIA® drivers,
  - CUDA Toolkit v9.0, and
  - cuDNN v7.0

  are needed: https://tensorflow.rstudio.com/tools/local_gpu.html

INSTALLING KERAS (MAC AND LINUX) CONT…



INSTALLING KERAS (WINDOWS) CONT…



MARQUETTE UNIVERSITY
Be The Difference.

DEVELOPING A DEEP NN WITH KERAS

- Step 1 - Define your training data:
  – input tensors and target tensors.
- Step 2 - Define a network of layers (or *model*)
  – that maps your inputs to your targets.
- Step 3 - Configure the learning process by choosing
  – a loss function,
  – an optimizer,
  – and some metrics to monitor.
- Step 4 - Iterate on your training data by calling the
  – **fit()** method of your model.

KERAS: STEP 1 – DATA PREPROCESSING

- `library(keras)`

- `# Load MNIST (Modified National Institute of Standards and Technology) images datasets`
  `c(c(x_train, y_train), c(x_test, y_test)) %<-% dataset_mnist()`

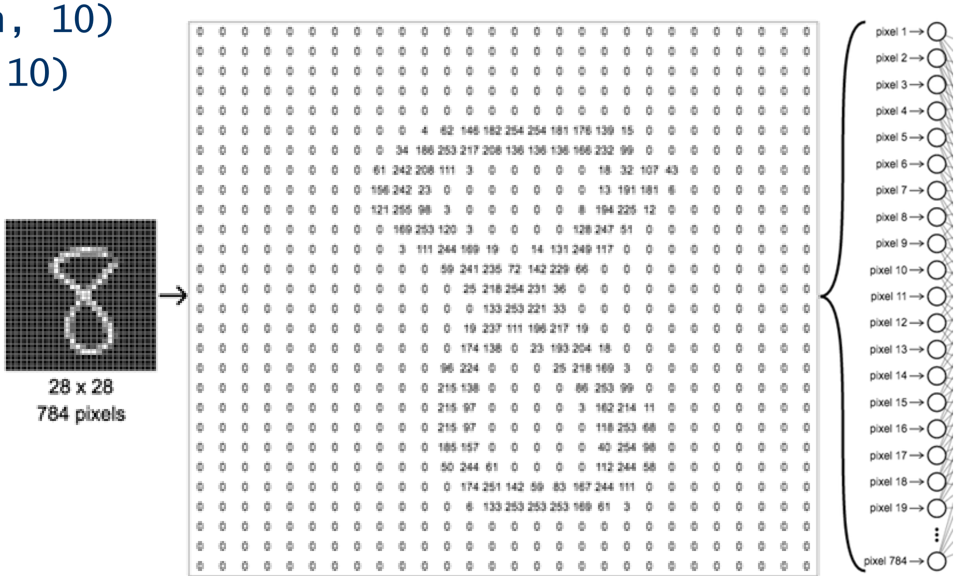
- `# Flatten images and transform RGB values into [0,1] range`
- `x_train <- array_reshape(x_train, c(nrow(x_train), 784))`
- `x_test <- array_reshape(x_test, c(nrow(x_test), 784))`
- `x_train <- x_train / 255`
- `x_test <- x_test / 255`

- `# Convert class vectors to binary class matrices`
- `y_train <- to_categorical(y_train, 10)`
- `y_test <- to_categorical(y_test, 10)`

```
> dim(x_train)
[1] 60000    28    28
> dim(x_test)
[1] 10000    28    28
```

```
> dim(y_train)
[1] 60000
> dim(y_test)
[1] 10000
> head(y_train)
[1] 5 0 4 1 9 2
```

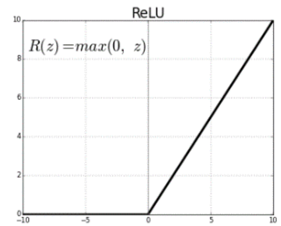
```
> to_categorical(c(5,0,4,1,9,2),10)
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10]
[1,]    0    0    0    0    0    1    0    0    0     0
[2,]    1    0    0    0    0    0    0    0    0     0
[3,]    0    0    0    0    1    0    0    0    0     0
[4,]    0    1    0    0    0    0    0    0    0     0
[5,]    0    0    0    0    0    0    0    0    0     1
[6,]    0    0    1    0    0    0    0    0    0     0
```



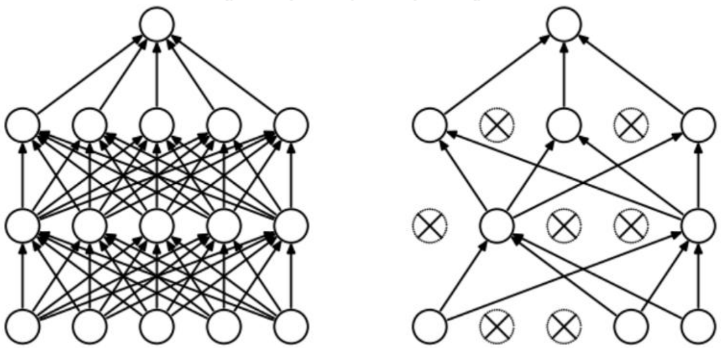
KERAS: STEP 2 – MODEL DEFINITION

- model <- keras_model_sequential(input_shape = c(784))
- model %>%

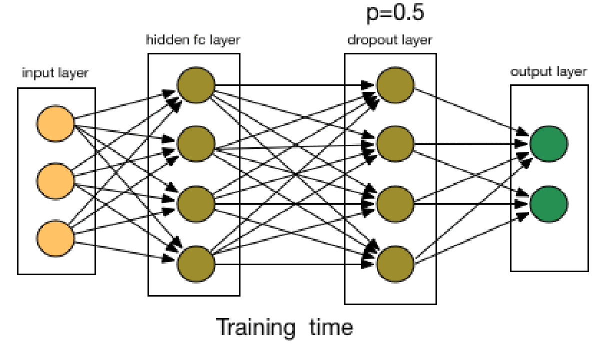
```
    layer_dense(units = 256, activation = 'relu') %>%
    layer_dropout(rate = 0.4) %>%
    layer_dense(units = 128, activation = 'relu') %>%
    layer_dropout(rate = 0.3) %>%
    layer_dense(units = 10, activation = 'softmax')
```




```
> summary(model)
________________________________________________________________________
Layer (type)                    Output Shape                  Param #
========================================================================
dense_4 (Dense)                 (None, 256)                   200960
________________________________________________________________________
dropout_3 (Dropout)             (None, 256)                   0
________________________________________________________________________
dense_5 (Dense)                 (None, 128)                   32896
________________________________________________________________________
dropout_4 (Dropout)             (None, 128)                   0
________________________________________________________________________
dense_6 (Dense)                 (None, 10)                    1290
========================================================================
Total params: 235,146
Trainable params: 235,146
Non-trainable params: 0
________________________________________________________________________
```



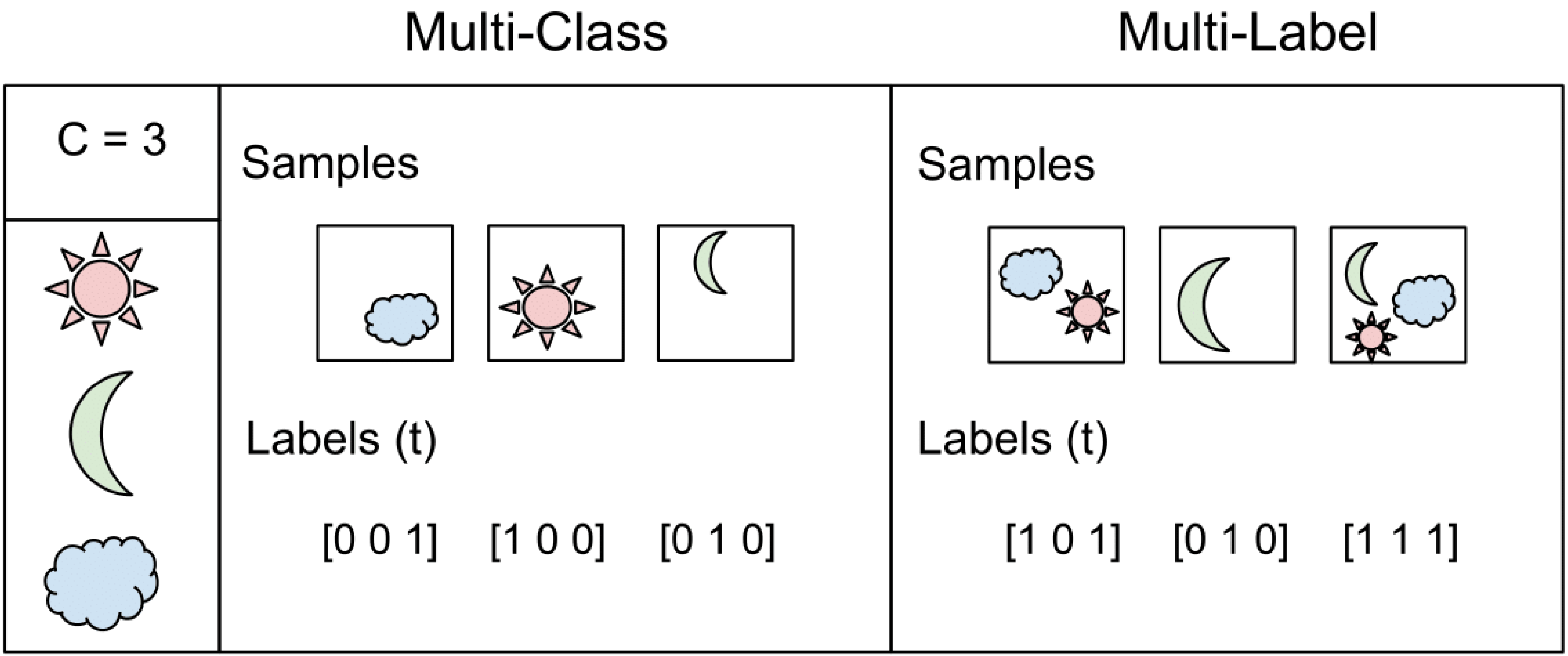
(a) Standard Neural Net (b) After applying dropout.



- Number of parameters in the model:
  - (784+1)*256 + (256+1)*128 + (128+1)*10 =
  - 200,960 + 32,896 + 1,290 = 235,146

MARQUETTE UNIVERSITY
Be The Difference.

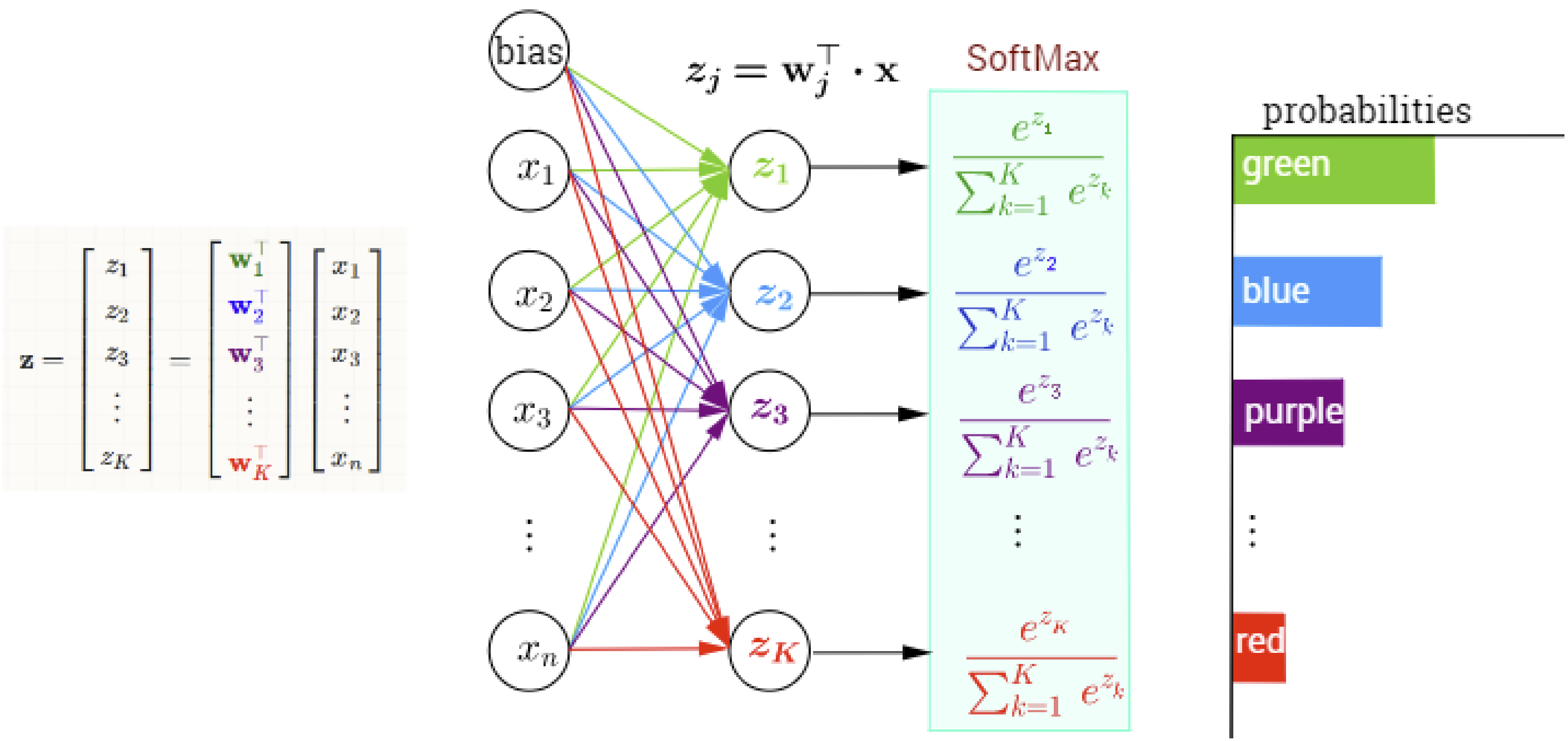
MULTI-CLASS VS MULTI-LABEL CLASSIFICATION



MARQUETTE UNIVERSITY
Be The Difference.

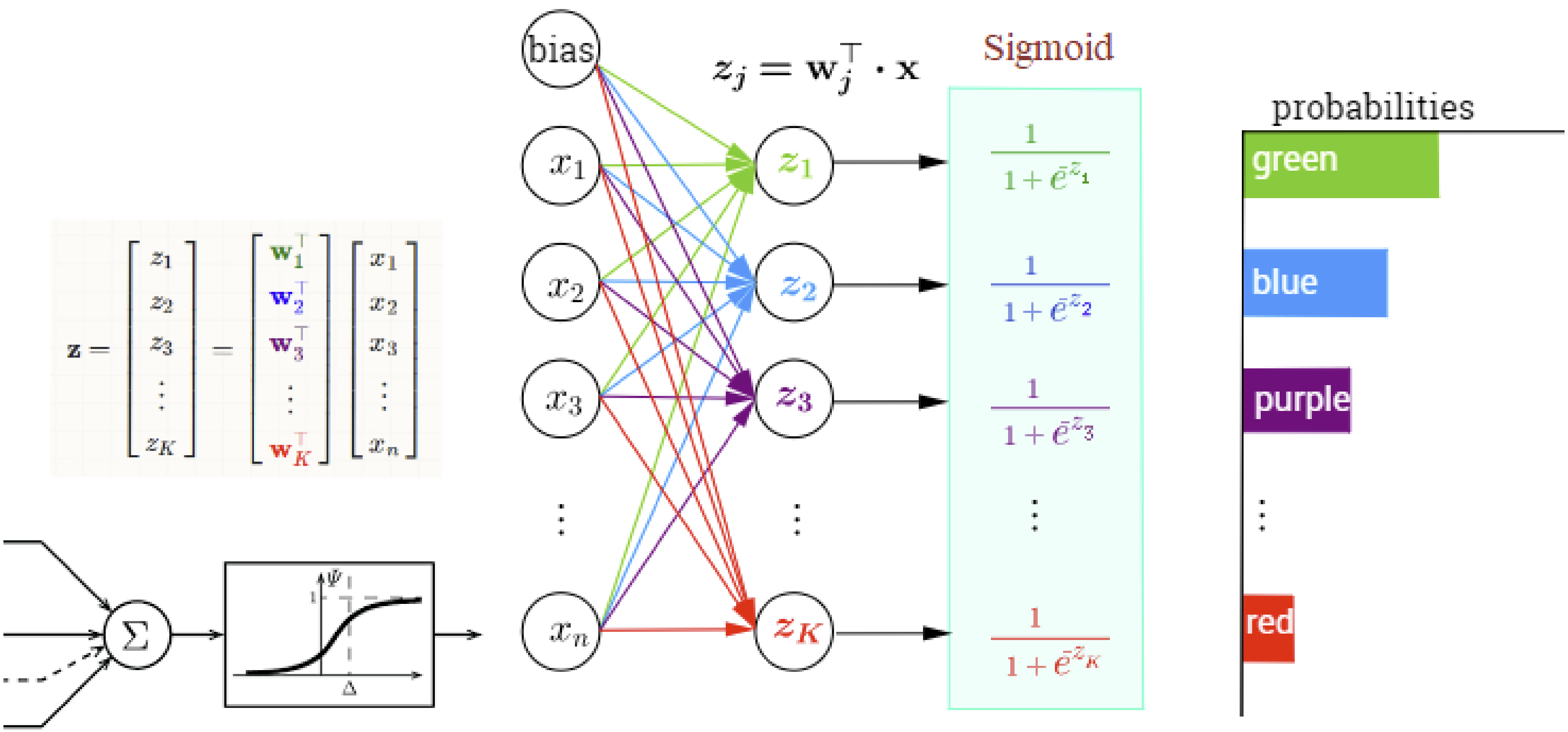
Multi-Class vs Multi-Label Classification Cont…

Multi-Class Classification with NN and SoftMax Function



Multi-Class vs Multi-Label Classification Cont…

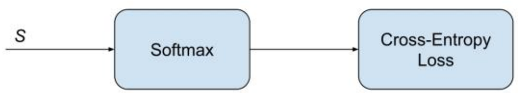
Multi–Label Classification with NN and Sigmoid Function



Keras: Step 3 – Compile Model

- Model compilation prepares the model for training by:
  1. Converting the layers into a TensorFlow graph
  2. Applying the specified loss function and optimizer
  3. Arranging for the collection of metrics during training

```
model %>% compile(
  loss = 'categorical_crossentropy',
  optimizer = optimizer_rmsprop(),
  metrics = c('accuracy')
)
```



$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \qquad CE = -\sum_i^C t_i log(f(s)_i)$$

$$\hat{\mathbf{y}} = \begin{bmatrix} 0.1 \\ 0.5 \\ 0.4 \end{bmatrix} \qquad D(\hat{\mathbf{y}}, \mathbf{y}) = -\sum_j y_j \ln \hat{y}_j \qquad \mathbf{y} = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}$$

Choosing the right last-layer activation and loss function for your model

Problem type	Last-layer activation	Loss function
Binary classification	`sigmoid`	`binary_crossentropy`
Multiclass, single-label classification	`softmax`	`categorical_crossentropy`
Multiclass, multilabel classification	`sigmoid`	`binary_crossentropy`
Regression to arbitrary values	`None`	`mse`
Regression to values between 0 and 1	`sigmoid`	`mse` or `binary_crossentropy`

Keras API: Losses

https://tensorflow.rstudio.com/keras/reference/#section-losses

```
loss_binary_crossentropy()
loss_categorical_crossentropy()
loss_categorical_hinge()
loss_cosine_proximity()
loss_hinge()
loss_kullback_leibler_divergence()
loss_logcosh()
loss_mean_absolute_error()
loss_mean_absolute_percentage_error()
loss_mean_squared_error()
loss_mean_squared_logarithmic_error()
loss_poisson()
loss_sparse_categorical_crossentropy()
loss_squared_hinge()
```

Keras API: Optimizers

https://tensorflow.rstudio.com/keras/reference/#section-optimizers

```
optimizer_adadelta()
optimizer_adagrad()
optimizer_adam()
optimizer_adamax()
optimizer_nadam()
optimizer_rmsprop()
optimizer_sgd()
```

Keras API: Metrics

https://tensorflow.rstudio.com/keras/reference/#section-metrics

```
metric_binary_accuracy()
metric_binary_crossentropy()
metric_categorical_accuracy()
metric_categorical_crossentropy()
metric_cosine_proximity()
metric_hinge()
metric_kullback_leibler_divergence()
metric_mean_absolute_error()
metric_mean_absolute_percentage_error()
metric_mean_squared_error()
metric_mean_squared_logarithmic_error()
metric_poisson()
metric_sparse_categorical_crossentropy()
metric_sparse_top_k_categorical_accuracy()
metric_squared_hinge()
metric_top_k_categorical_accuracy()
```

KERAS: STEP 4 – MODEL TRAINING

- Use the **fit()** function to train the model for 10 epochs using batches of 128 images:

```
history <- model %>% fit(
  x_train, y_train,
  batch_size = 128,
  epochs = 10,
  validation_split = 0.2
)
```

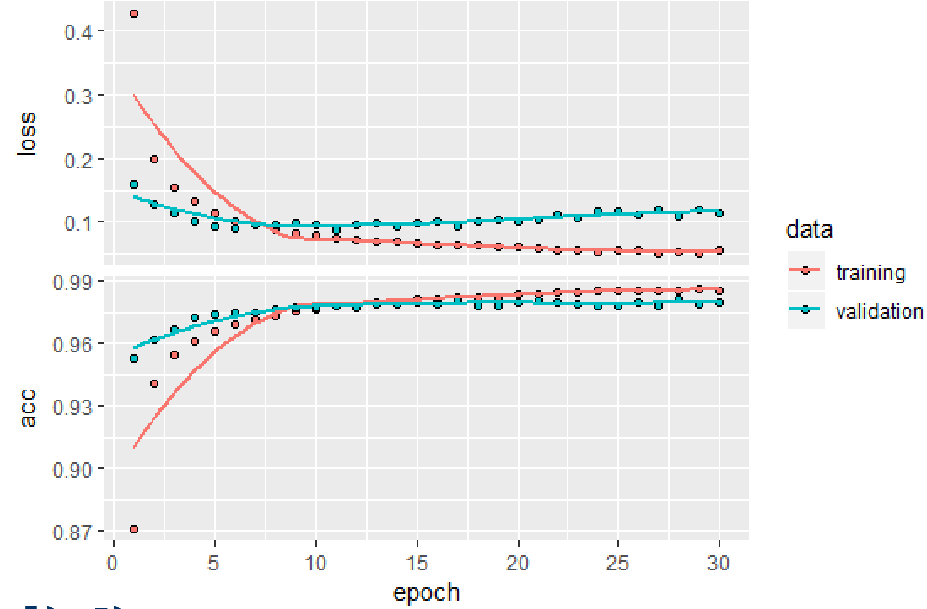
- Feed 128 samples at a time to the model (batch_size = 128)
- Traverse the input dataset 10 times (epochs = 10)
- Hold out 20% of the data for validation (validation_split = 0.2)

```
> model %>% evaluate(x_test, y_test)
10000/10000 [==============================] - 0s 29us/step
$`loss`
[1] 0.1085284375

$acc
[1] 0.9816
```

KERAS: EVALUATION AND PREDICTION

- `plot(history)`



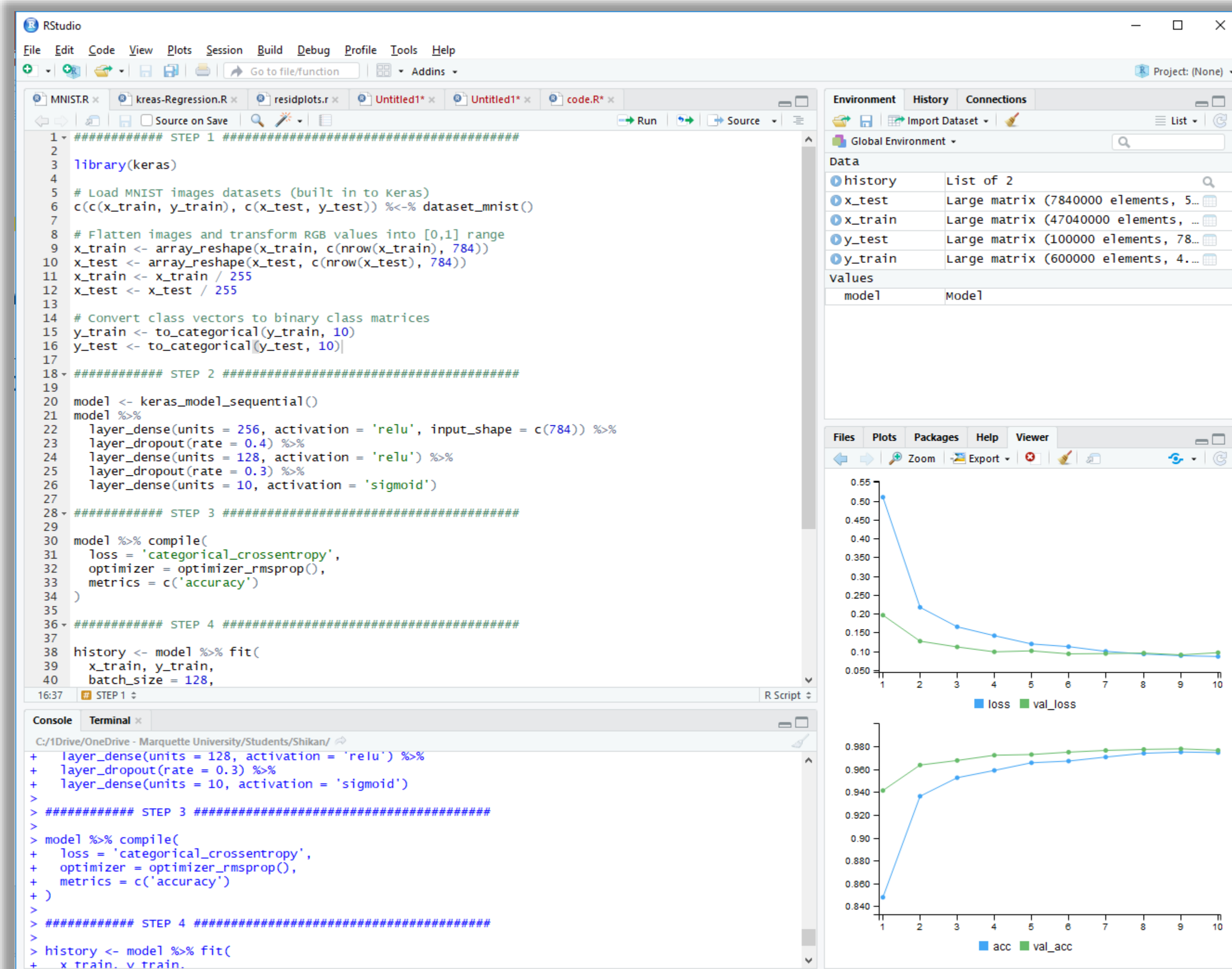
- model %>% predict(x_test[1:100,]) %>% apply(1, which.max)-1

```
 [1] 7 2 1 0 4 1 4 9 6 9 0 6 9 0 1 5 9 7
[19] 3 4 9 6 6 5 4 0 7 4 0 1 3 1 3 4 7 2
[37] 7 1 2 1 1 7 4 2 3 5 1 2 4 4 6 3 5 5
[55] 6 0 4 1 9 5 7 8 9 3 7 4 6 4 3 0 7 0
[73] 2 9 1 7 3 2 9 7 7 6 2 7 8 4 7 3 6 1
[91] 3 6 9 3 1 4 1 7 6 9
```

- `round(model %>% predict(x_test[1:9,]),5)`

```
     [,1] [,2] [,3]    [,4] [,5]    [,6]    [,7] [,8] [,9]   [,10]
[1,]    0    0    0 0.00000    0 0.00000 0.00000    1    0 0.00000
[2,]    0    0    1 0.00000    0 0.00000 0.00000    0    0 0.00000
[3,]    0    1    0 0.00000    0 0.00000 0.00000    0    0 0.00000
[4,]    1    0    0 0.00000    0 0.00000 0.00000    0    0 0.00000
[5,]    0    0    0 0.00000    1 0.00000 0.00000    0    0 0.00000
[6,]    0    1    0 0.00000    0 0.00000 0.00000    0    0 0.00000
[7,]    0    0    0 0.00000    1 0.00000 0.00000    0    0 0.00000
[8,]    0    0    0 0.00002    0 0.00000 0.00000    0    0 0.99998
[9,]    0    0    0 0.00000    0 0.01416 0.98584    0    0 0.00000
```

KERAS DEMO



- https://keras3.posit.co/articles/getting_started.html

MARQUETTE UNIVERSITY
Be The Difference.

KERAS API: LAYERS

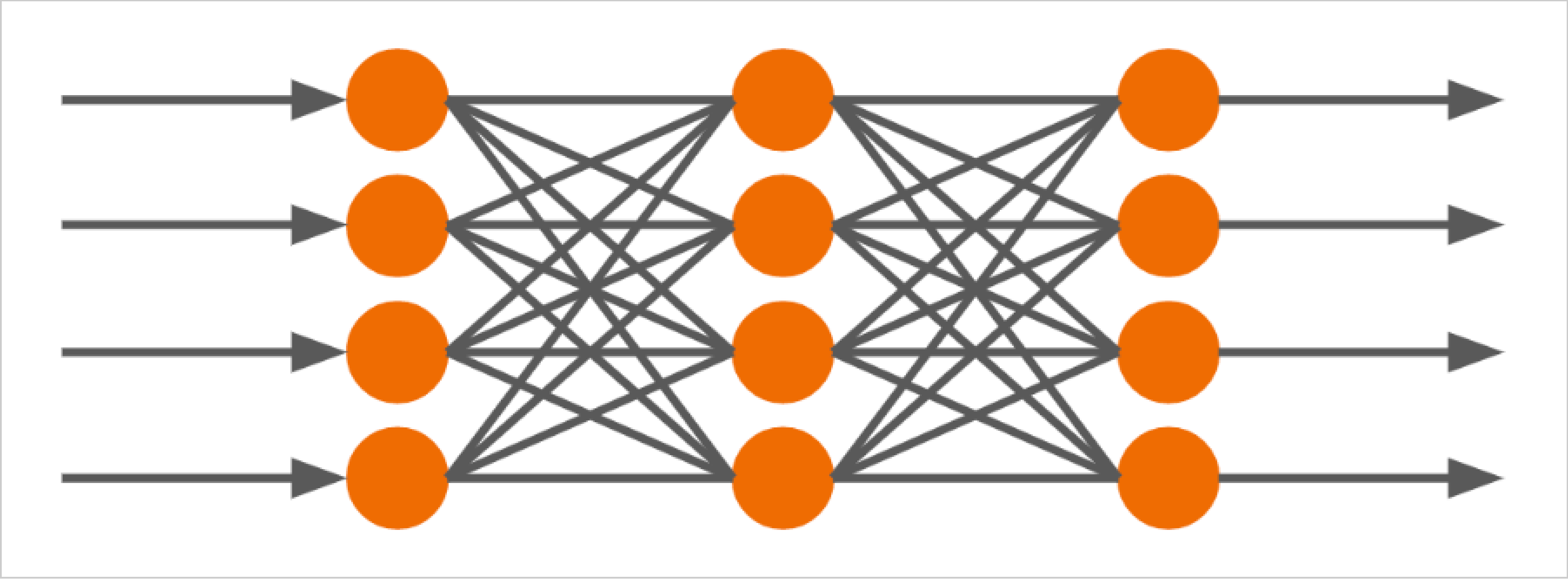
- 90+ layers available (you can also create your own)

layer_dense()	Add a densely-connected NN layer to an output.
layer_dropout()	Applies Dropout to the input.
layer_batch_normalization()	Batch normalization layer (Ioffe and Szegedy, 2014).
layer_conv_2d()	2D convolution layer (e.g. spatial convolution over images).
layer_max_pooling_2d()	Max pooling operation for spatial data.
layer_gru()	Gated Recurrent Unit - Cho et al.
layer_lstm()	Long-Short Term Memory unit.
layer_embedding()	Turns positive integers (indexes) into dense vectors of fixed size.
layer_reshape()	Reshapes an output to a certain shape.
layer_flatten()	Flattens an input.

KERAS API: DENSE LAYERS

- Classic "fully connected" neural network layers

  - `layer_dense()`

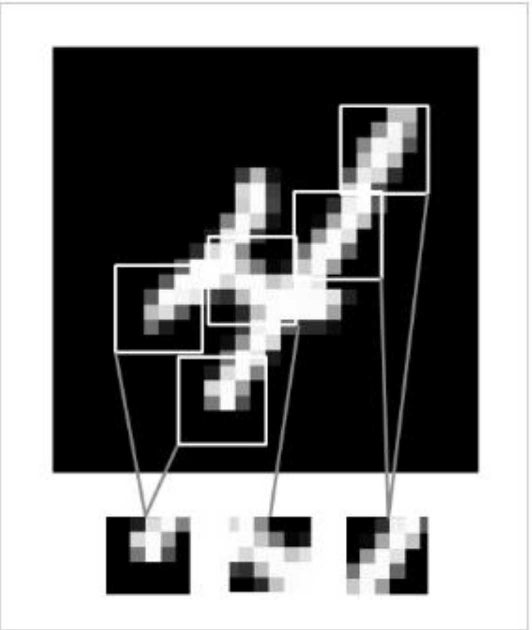


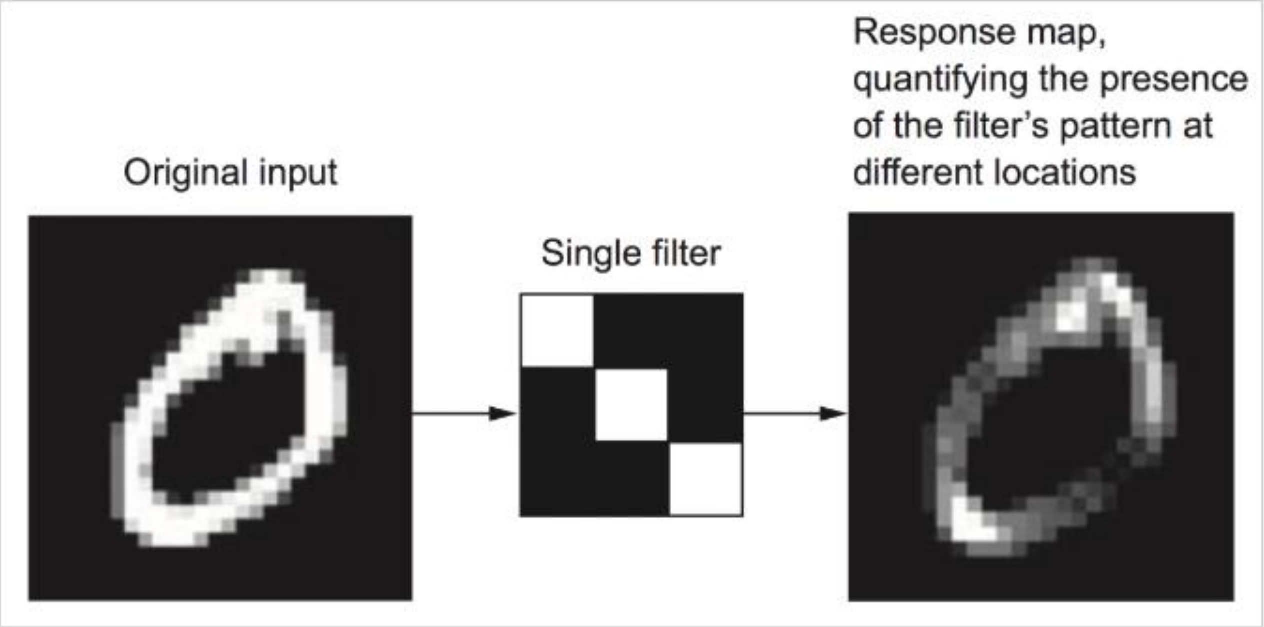
  - `layer_dense(units = 64, kernel_regularizer = regularizer_l1(0.01))`
  - `layer_dense(units = 64, bias_regularizer = regularizer_l2(0.01))`

KERAS API: CONVOLUTIONAL LAYERS

- Filters for learning local (translation invariant) patterns in data

  - `layer_conv_2d()`





CONVOLUTION (MATHEMATICAL DEFINITION)

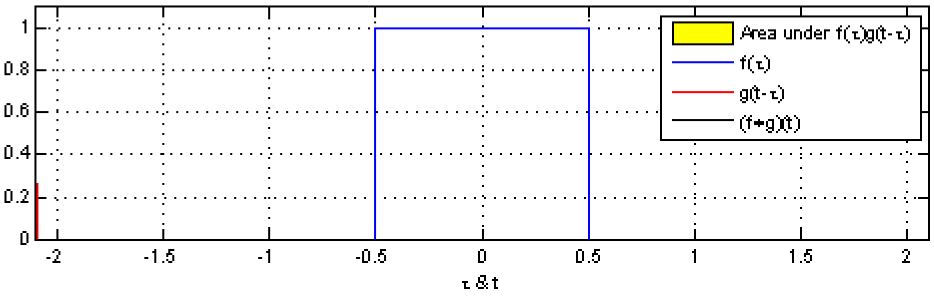
Definition [edit]

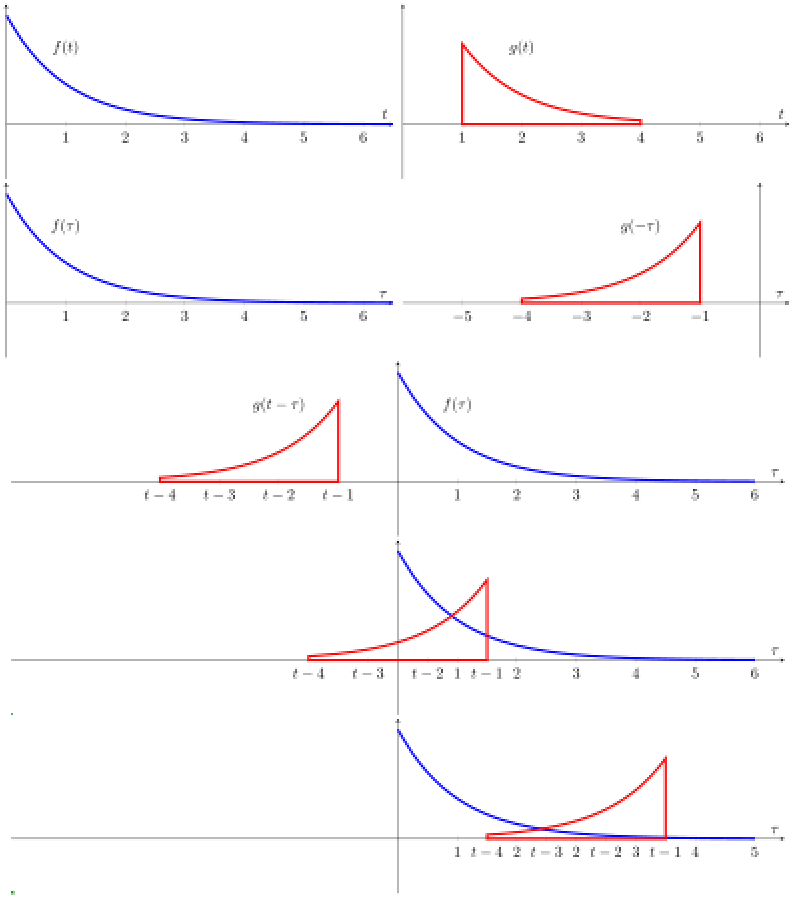
The convolution of f and g is written $f*g$, denoting the operator with the symbol $*$.[B] It is defined as the integral of the product of the two functions after one is reversed and shifted. As such, it is a particular kind of integral transform:

$$(f*g)(t) \triangleq \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\,d\tau.$$

An equivalent definition is (see commutativity):

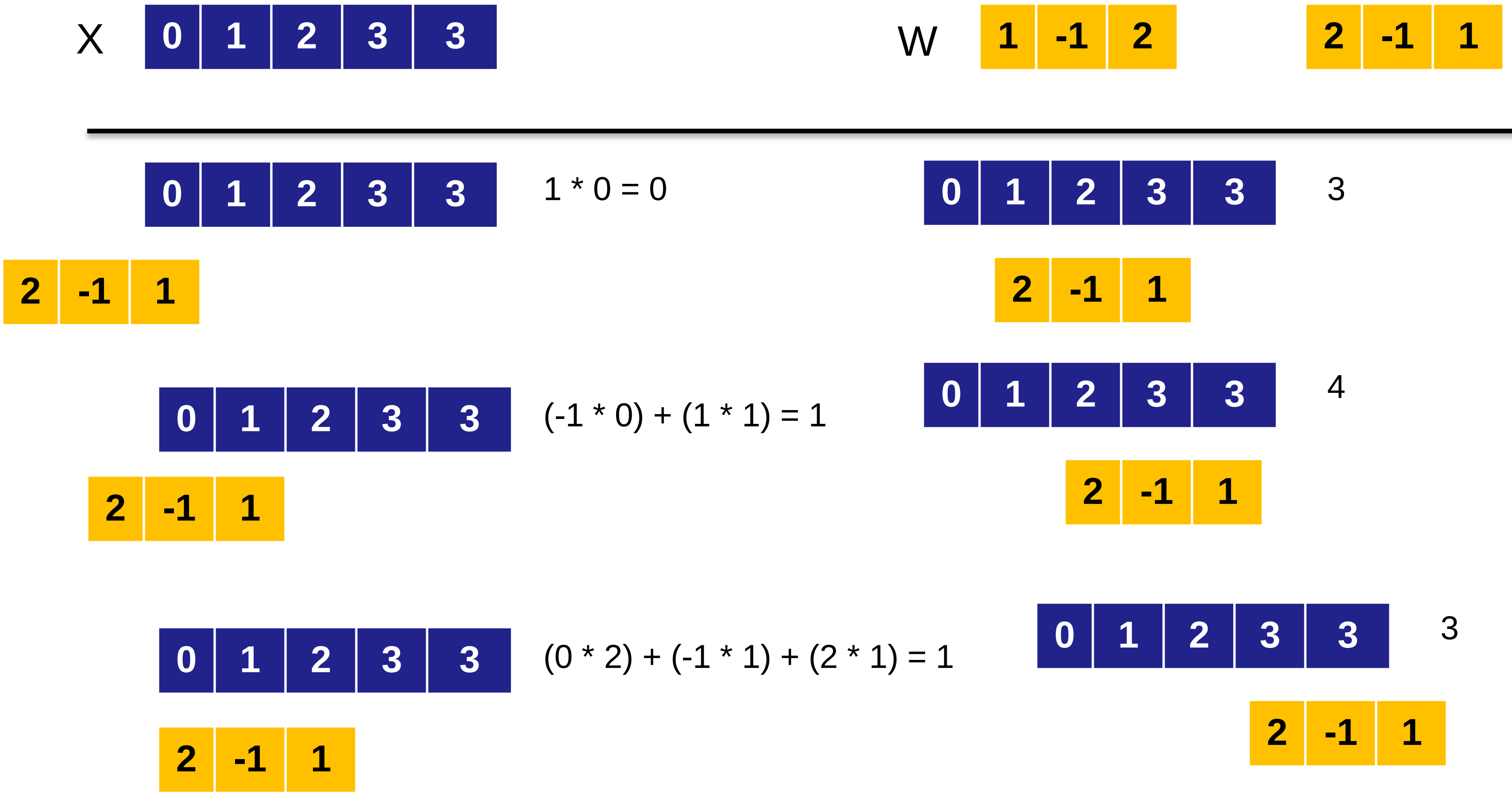
$$(f*g)(t) \triangleq \int_{-\infty}^{\infty} f(t-\tau)g(\tau)\,d\tau.$$





CONVOLUTION IN ENGINEERING WORLD

- In mathematics, Convolution is an operation which does the integral of the product of 2 functions (e.g., 2 signals), with one of the signals flipped.



MARQUETTE UNIVERSITY
Be The Difference.

CONVOLUTION

0	1	2	3	3

6

2	-1	1

Output: X

0	1	1	3	4	3	6

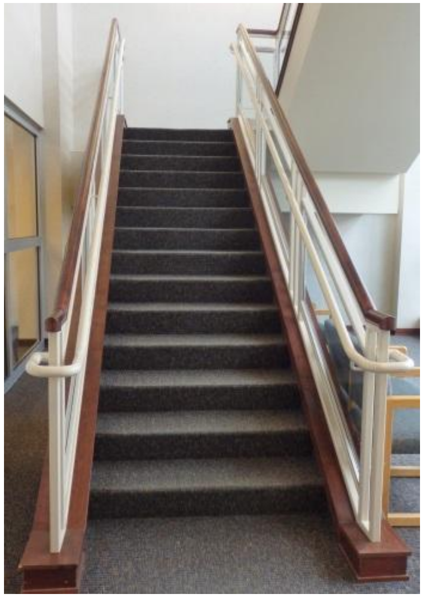
MARQUETTE UNIVERSITY
Be The Difference.

CONVOLUTIONAL NEURAL NETWORKS: WHY?

- Why do shallow fully connected neural networks not work when the input is an image?

- There are two main reasons:

(1) The input consists of 42,000 numbers, therefore many weights are needed for each node in the hidden Layer. Saying 100 nodes in the first layer, this corresponds to 4,200,000 weight parameters required to define only this layer. More **parameters** mean that **more training data** is needed to prevent **overfitting**. This leads to more time required to train the model.



[100 * 140 * 3] = 42,000

(2) Processing by Fully Connected Deep Feed Forward Networks requires that the image data be transformed into a linear 1-D vector. This results in a **loss of structural information**, including correlation between pixel values in 2-D.

CONVOLUTIONAL NEURAL NETWORKS: THE LAYERS

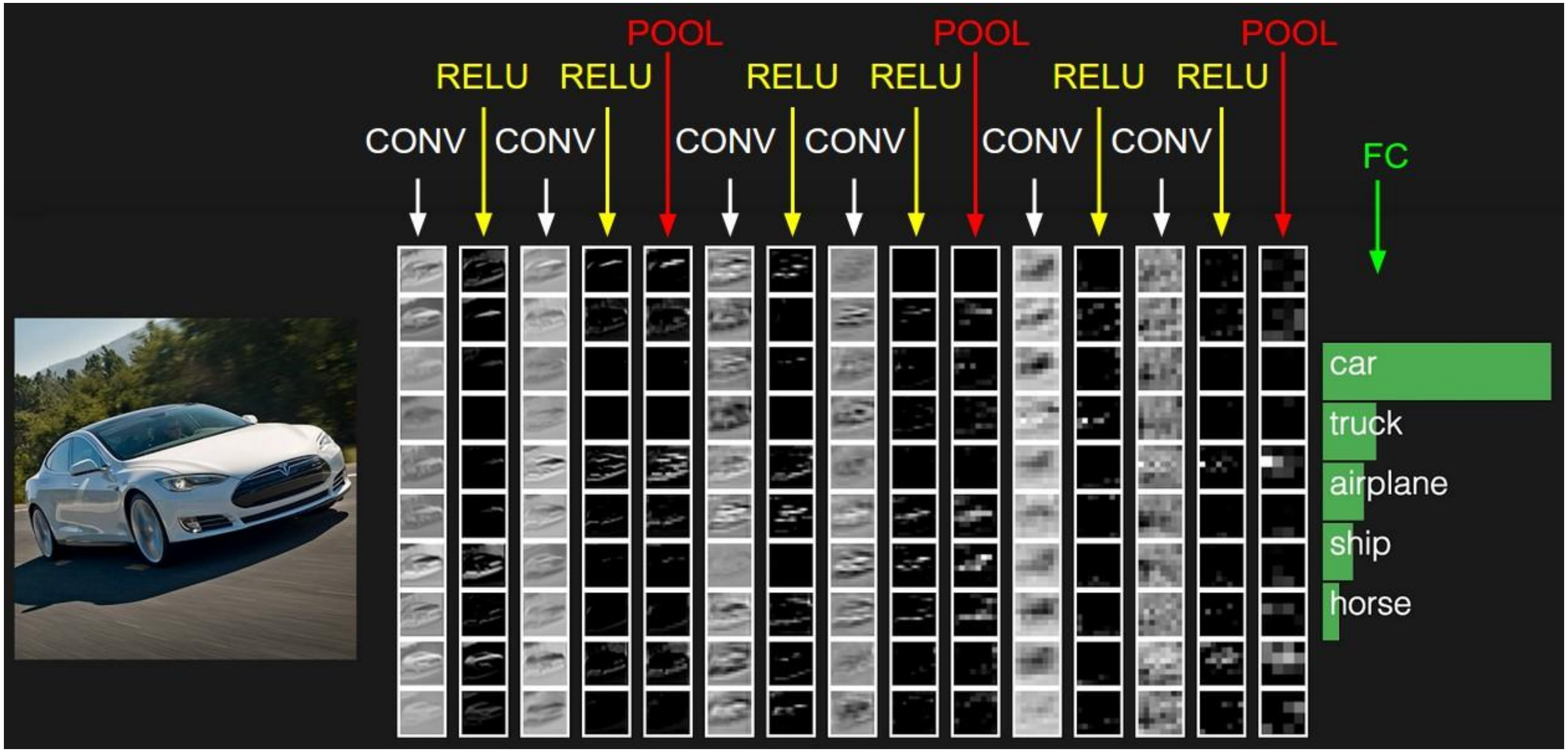
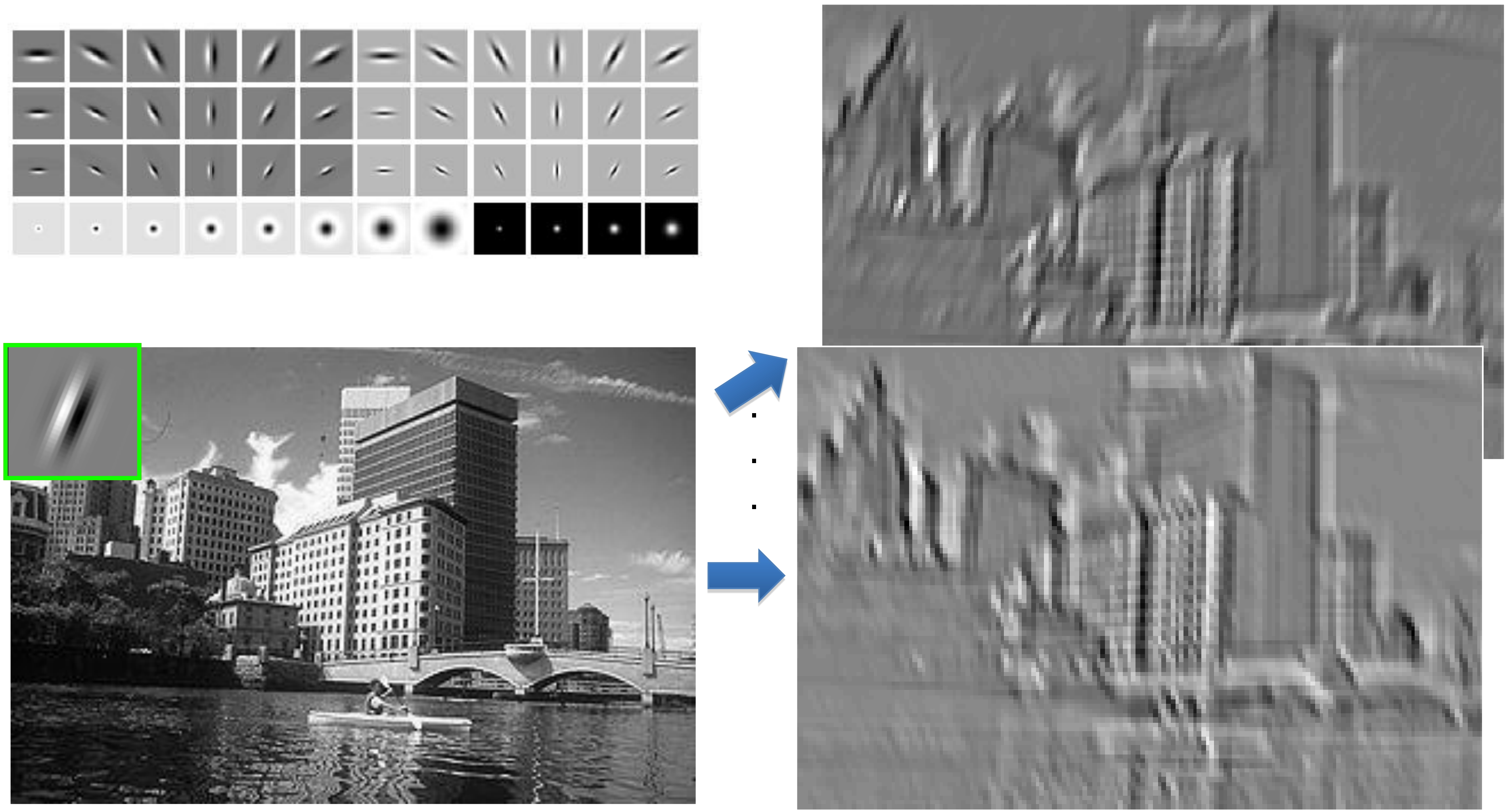


Image from: http://cs231n.github.io/convolutional-networks/

CONVOLUTION AS FEATURE EXTRACTION

bank of K filters

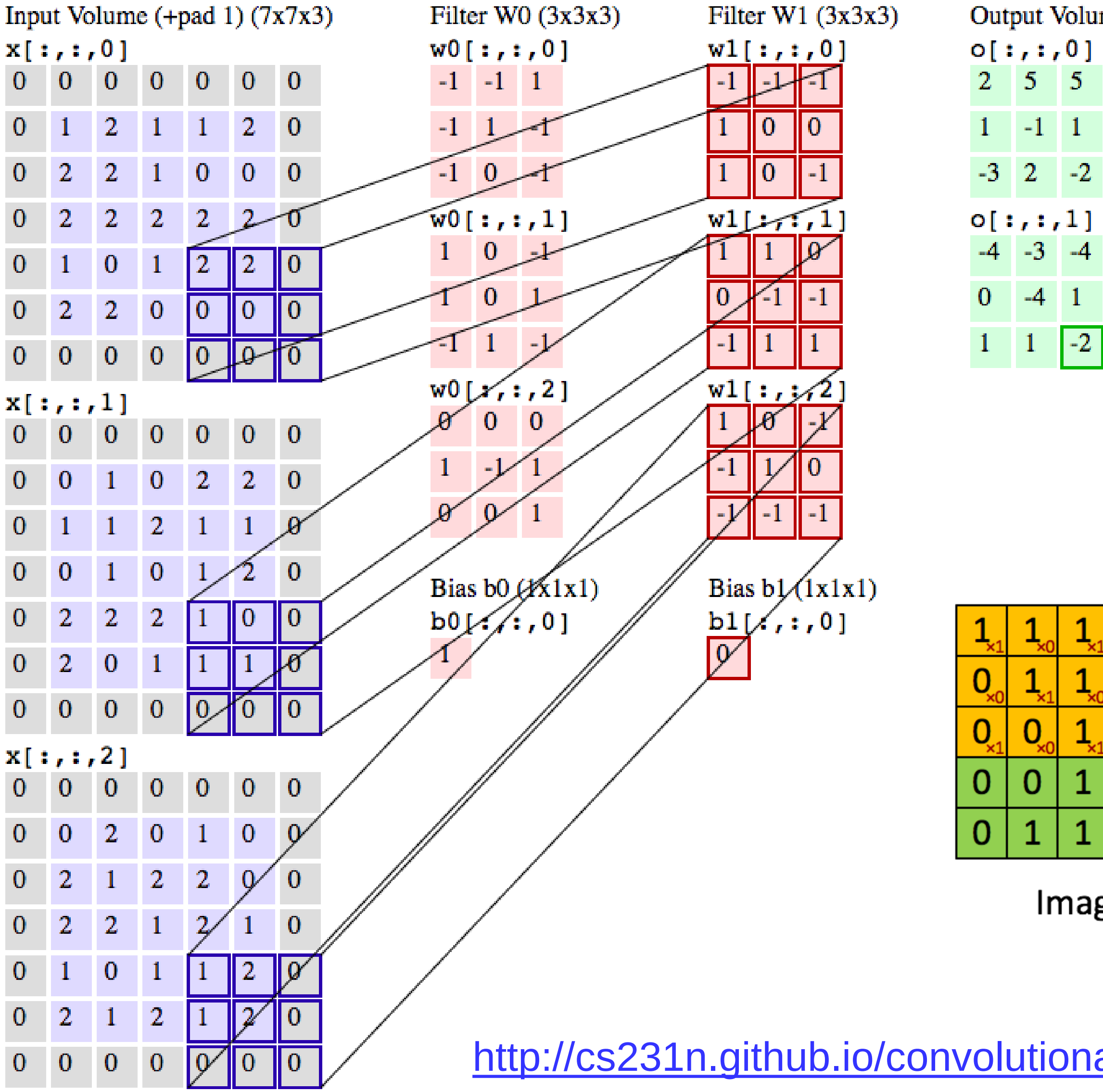
K feature maps

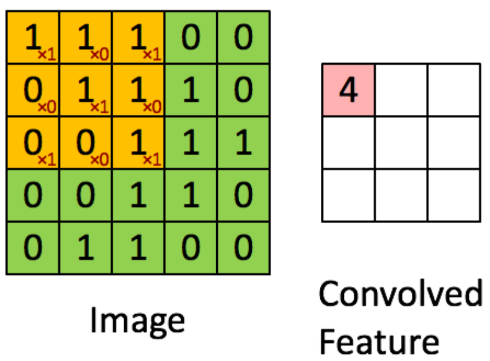


image

feature map

CONVOLUTIONAL LAYER DEMO





http://cs231n.github.io/convolutional-networks/#conv

CONVOLUTIONAL LAYER

- In CNN, we are working with **multiple filters**. Each filter looks for a specific kind of **feature/pattern/concept** in the input image. For example, we want our convolution layer to look for 6 different patterns. So, our convolution layer will have 6 number of 5x5x3 filters, each one looks for a specific pattern on the image.

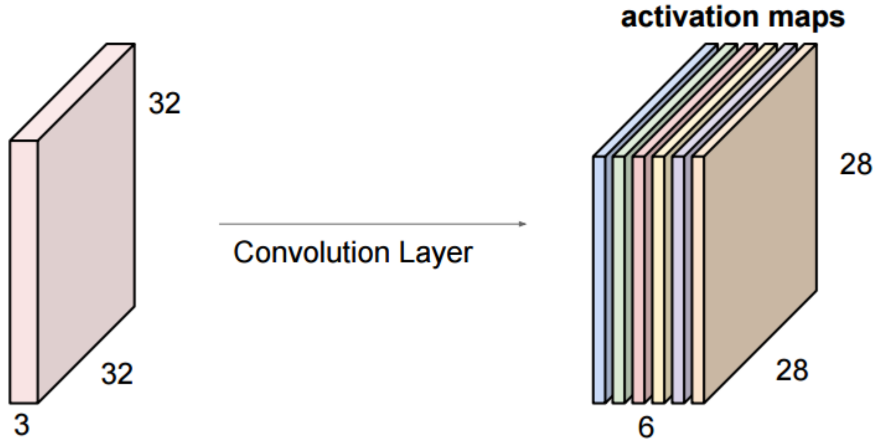
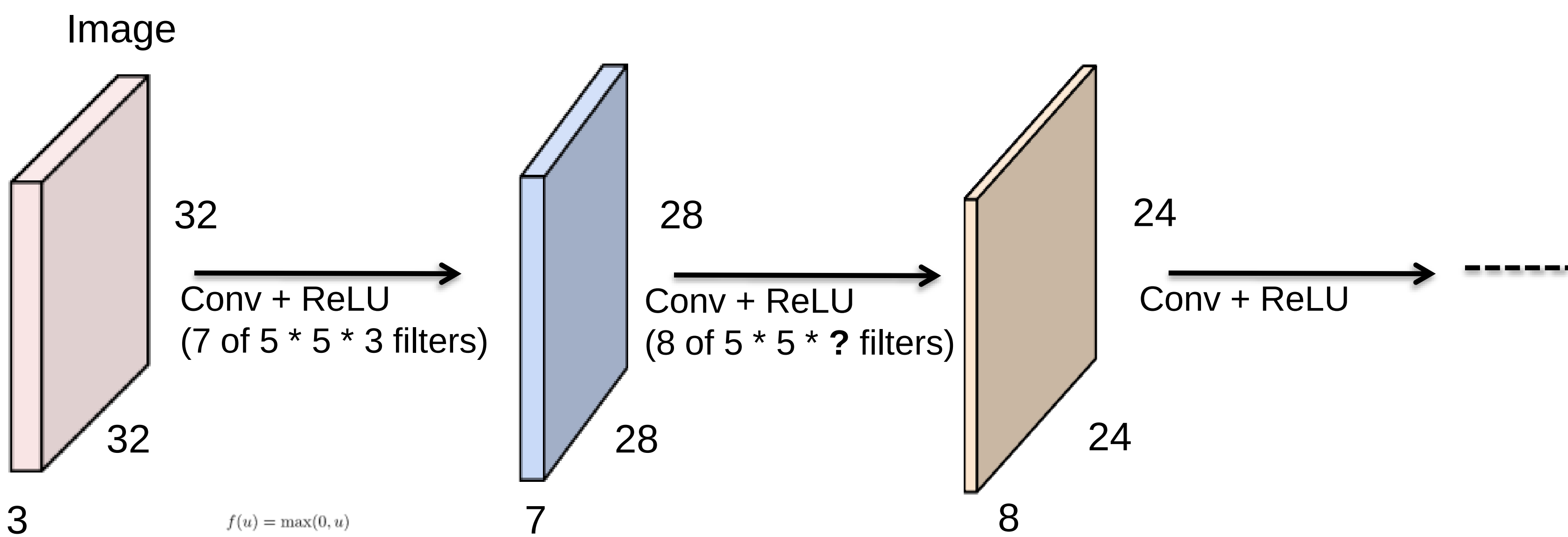


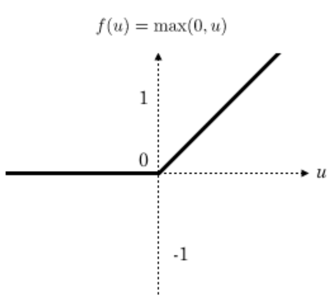
Image from: https://legacy.gitbook.com/book/leonardoaraujosantos/artificial-inteligence

- Stacking these up to make a new image of size 28 * 28 * 6

CONVOLUTIONAL LAYER

- Convolution itself is a linear kind of operation. There is a need to add at the end of the convolution layer a non-linear layer, called ReLU activation. ReLU is the max function(x,0) with input x matrix from a convolved image. ReLU then sets all negative values in the matrix x to zero and all other values are kept constant.

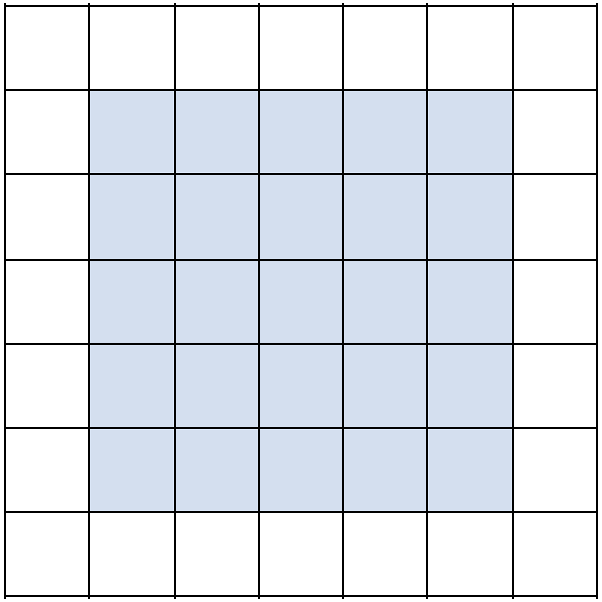




MARQUETTE UNIVERSITY
Be The Difference.

CONVOLUTIONAL NEURAL NETWORKS: A CLOSER LOOK

Input image: 7 * 7
Filter size: 3 * 3
Stride: 1

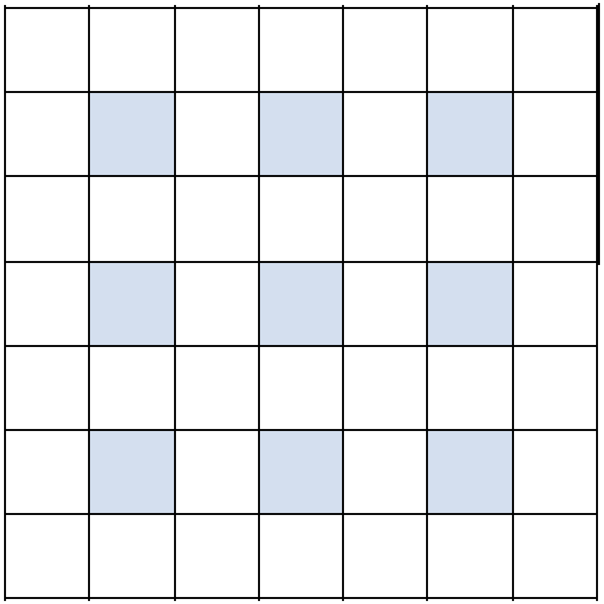


Output: 5 * 5

CONVOLUTIONAL NEURAL NETWORKS: A CLOSER LOOK

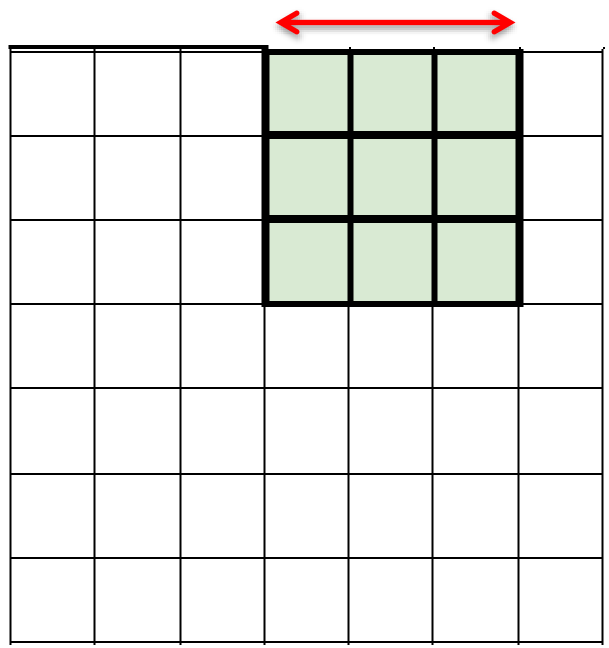
Input image: 7 * 7
Filter size: 3 * 3
Stride: 2

Output: 3 * 3



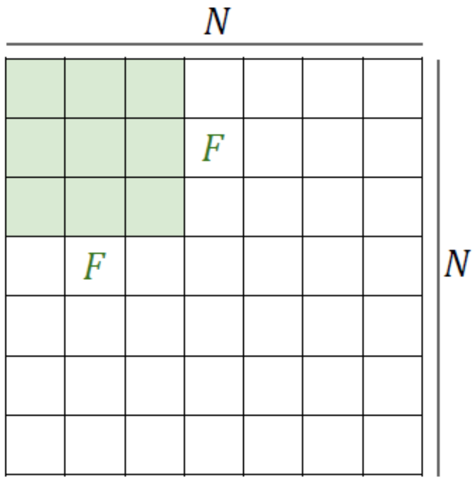
CONVOLUTIONAL NEURAL NETWORKS: A CLOSER LOOK

Input image: 7 * 7
Filter size: 3 * 3
Stride: 3



CONVOLUTIONAL NEURAL NETWORKS: A CLOSER LOOK

Output Size = (N – F) / Stride + 1



N = 7 F = 3

Stride = 1 Output Size = (7 – 3) / 1 +1 = 5

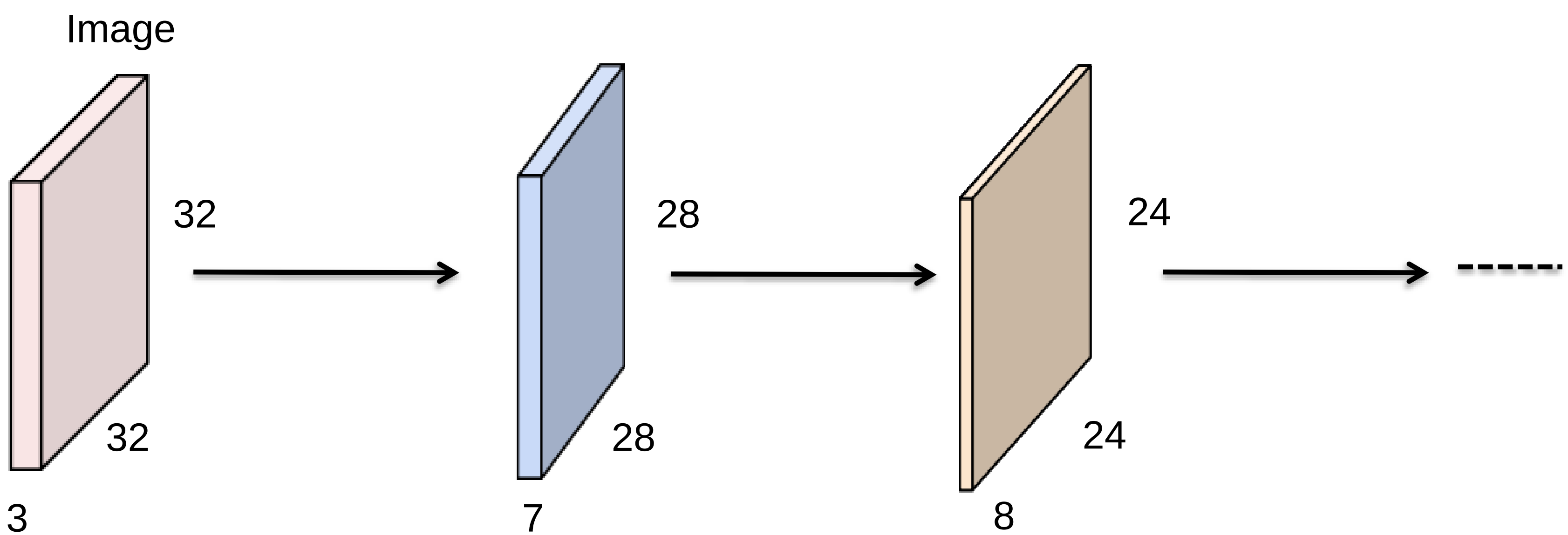
Stride = 2 Output Size = (7 – 3) / 2 + 1 = 3

Stride = 3 Output Size = (7 – 3) / 3 +1 = 2.33

Image from: http://cs231n.github.io/convolutional-networks/

CONVOLUTIONAL NEURAL NETWORKS

- We are condensing the data spatially! Too fast! What does that mean?



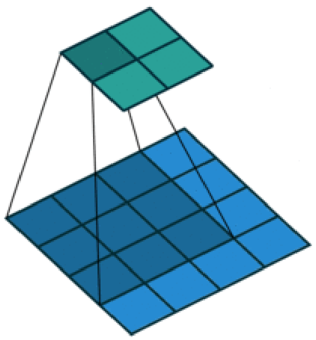
- Solution?

Zero Padding (pad): Add zeros on the image border to let the convolution output size be the same as the input image size.

MARQUETTE UNIVERSITY
Be The Difference.

CONVOLUTIONAL NEURAL NETWORKS

Input Size: 4 * 4
Filter Size: 3 * 3
Stride: 1
Padding: 0 (No Padding)



Input Size: 5 * 5
Filter Size: 3 * 3
Stride: 1
Padding: 1

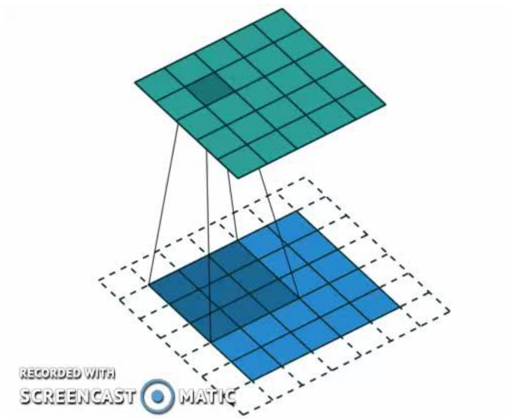


Image from: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolutional_neural_networks.html

Convolutional Neural Networks

Convolutional Layer:

It takes a data volume of size $W_1 * H_1 * D_1$

Hyper Parameters:

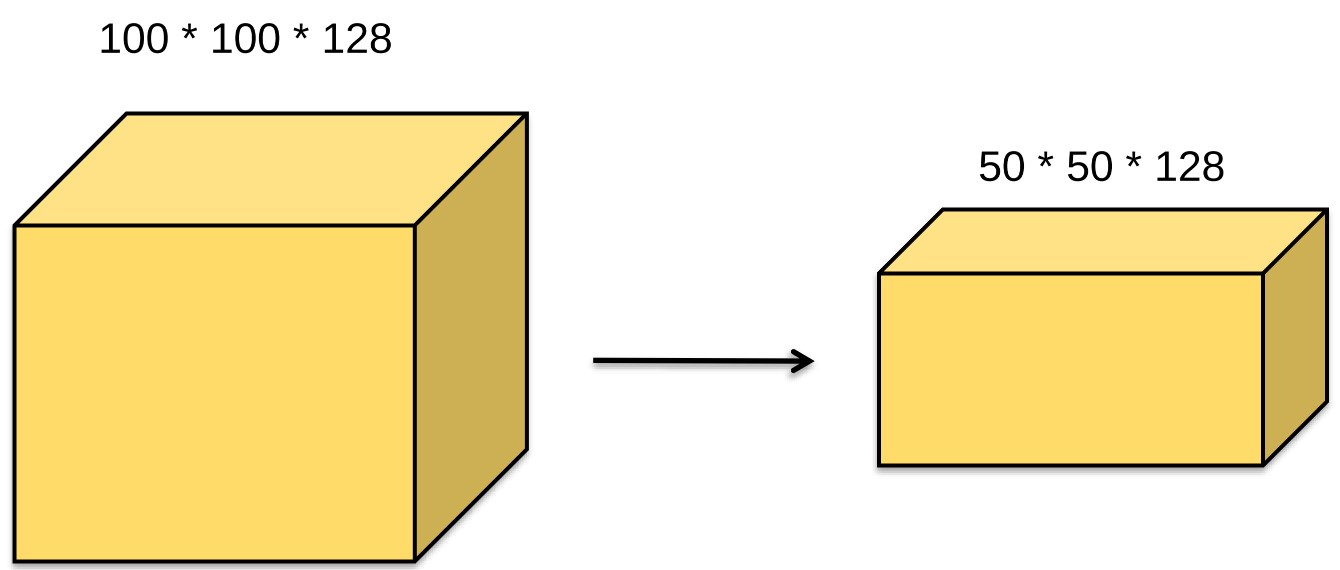
- Number of Filters (K)
- Filter Size (F)
- Stride (S)
- Zero Padding (P)

Common Configurations:

- K = 32, 64, 128, …
- F = 3, S = 1, P = 1
- F = 5, S = 1, P = 2
- F = 5, S = 2, P = 2

MAX POOL (POOLING)

It performs downsampling across the spatial dimensions (width, height). The representation would be smaller and more manageable.



MAX POOL (POOLING): HOW IT WORKS?

Filter Size: 2 * 2
Stride: 2

5	6	7	8
2	10	4	11
7	9	3	5
8	6	7	1

5	6	7	8
2	10	4	11
7	9	3	5
8	6	7	1

10	11
9	7

MARQUETTE UNIVERSITY
Be The Difference.

CONVOLUTIONAL NEURAL NETWORKS

Max Pool Layer:

It takes a data volume of size $W_1 * H_1 * D_1$

Hyper Parameters:

- Filter Size (F)
- Stride (S)

Common Configurations:

- F = 2, S = 2
- F = 3, S = 2

FULLY CONNECTED LAYER

- It computes the class scores.

- This layer takes an input volume (the output of the Conv + ReLU + Pooling layer preceding it) and outputs an **N** dimensional vector, where **N** is the number of classes that we want to choose. For example, if we want to develop an object detection for Doors, Stairs, and Signs, then **N** would be 3.

- Each number in this **N** dimensional vector shows the probability of a class. For example, if the resulting vector for is [.1 .1 .80] for [Doors, Stair, Sign], then this represents a 10% probability that the image is a door, 10% probability that the image is a stair, and 80% probability that the image is sign.

CNN Architecture: Review

- **Input:** In our scenario, it holds the raw pixel values of an image (e.g., an image of width 32, height 32, and with three color channels R,G,B).

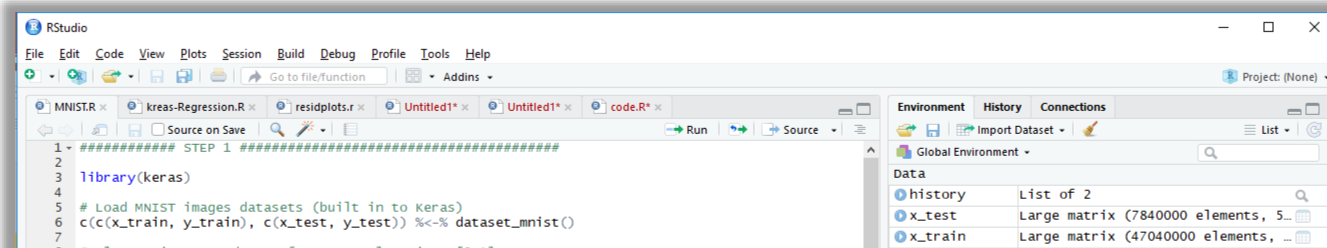
- **Convolutional Layer:** This layer filters (convolve) the inputs to provide very useful information appropriate for object modeling. These convolutional layers help to automatically extract the most valuable information for the task at hand without human designed feature selection. This layer will result in data volume such as [32 * 32 * 16] if we used for example 16 filters.

- **ReLU Layer:** will apply a pixelwise activation function, such as the $max(0,x)$ thresholding at zero. This layer keeps the size of the data volume unchanged (e.g., [32 * 32 * 16]).

- **Pooling Layer**: It does a downsampling operation across the spatial dimensions (width, height), and will result in data volume such as [16 * 16 * 16].

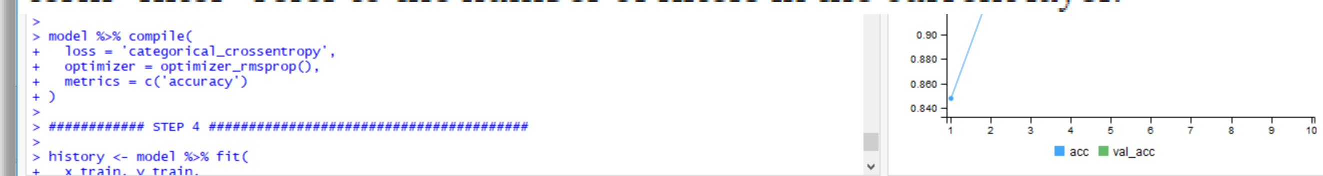
- **Fully Connected Layer:** This layer computes the class scores, and it will result in volume of size [1 * 1 * 3], where each of those 3 numbers correspond to a class score, such as among the 3 categories (doors, stairs, signs).

KERAS DEMO (MNIST CNN)



CONV layer: Number of Parameters

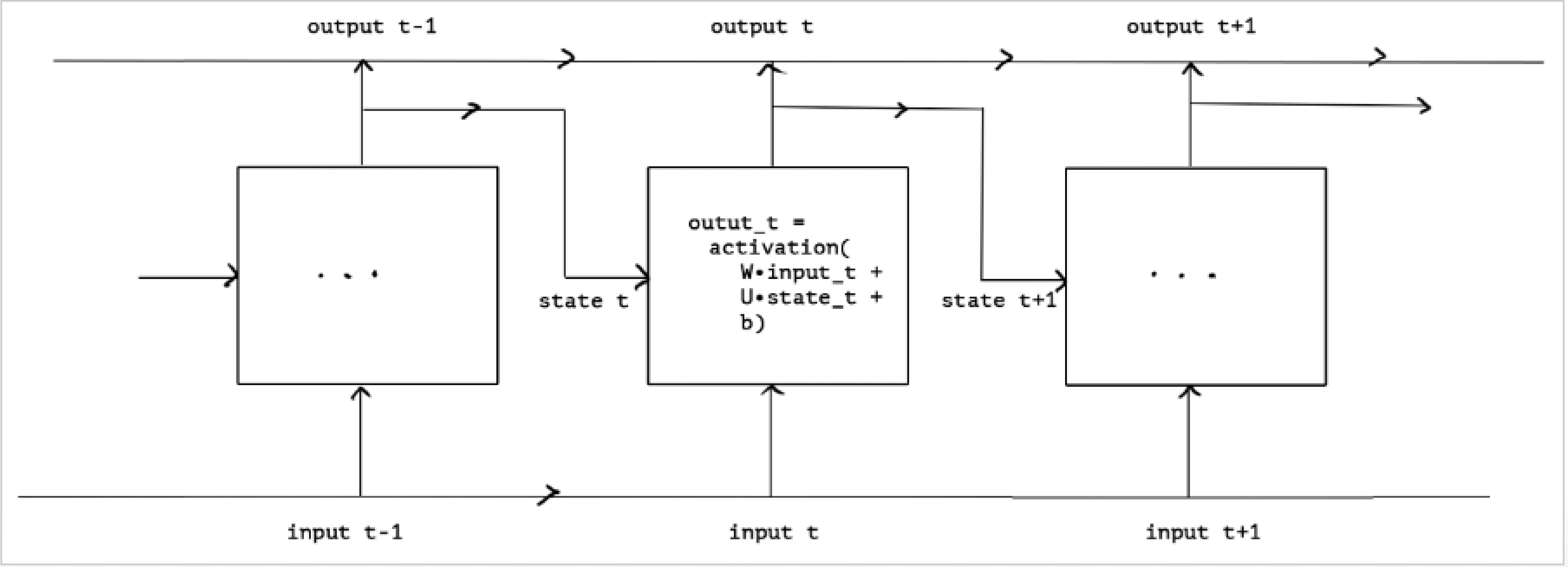
To calculate the learnable parameters here, all we have to do is just multiply the by the shape of **width m, height n, previous layer's filters d** and account for all such filters **k in the current layer.** Don't forget the bias term for each of the filter. Number of parameters in a CONV layer would be : **((m * n * d)+1)* k)**, added 1 because of the bias term for each filter. The same expression can be written as follows: **((shape of width of the filter * shape of height of the filter * number of filters in the previous layer+1)*number of filters).** Where the term "filter" refer to the number of filters in the current layer.



- https://keras3.posit.co/articles/examples/vision/mnist_convnet.html

KERAS API: RECURRENT LAYERS

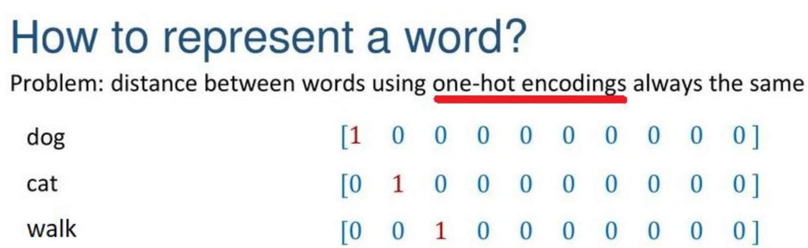
- Layers that maintain state based on previously seen data
  - `layer_simple_rnn()`
  - `layer_gru()`
  - `layer_lstm()`

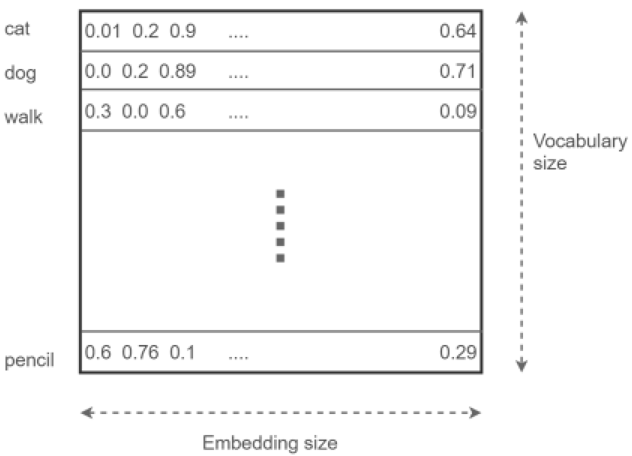


KERAS API: EMBEDDING LAYERS

- Vectorization of text that reflects semantic relationships between words

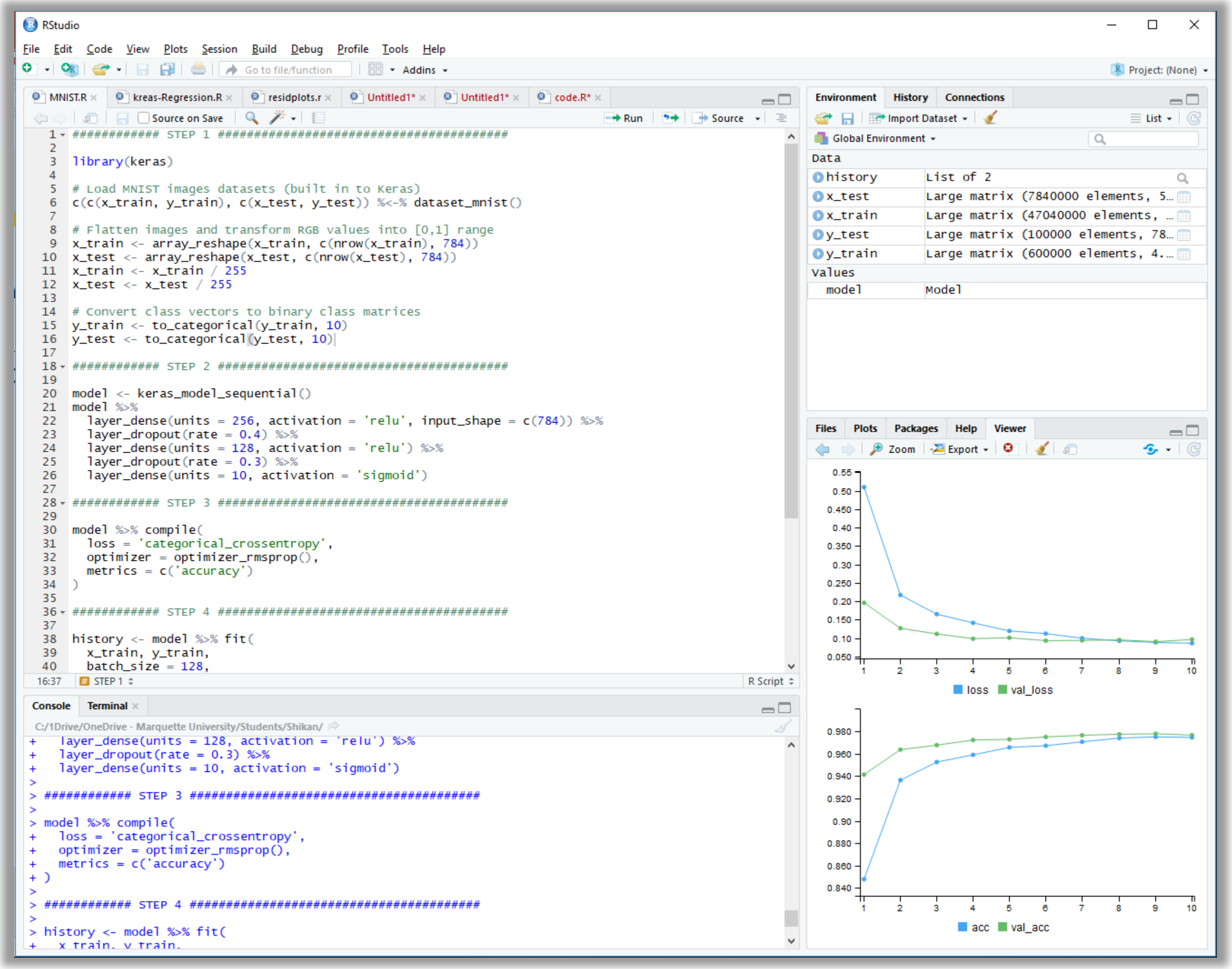
```
model <- keras_model_sequential() %>%
  layer_embedding(input_dim = 10000, output_dim = 8,
                  input_length = 20) %>%
  layer_flatten() %>%
  layer_dense(units = 1, activation = "sigmoid")
```





- Learn the embeddings jointly with the main task you care about (e.g. classification); or
- Load pre-trained word embeddings (e.g. Word2vec, GloVe)

KERAS DEMO (TEXT CLASSIFICATION)



■ https://keras3.posit.co/articles/examples/nlp/text_classification_from_scratch.html

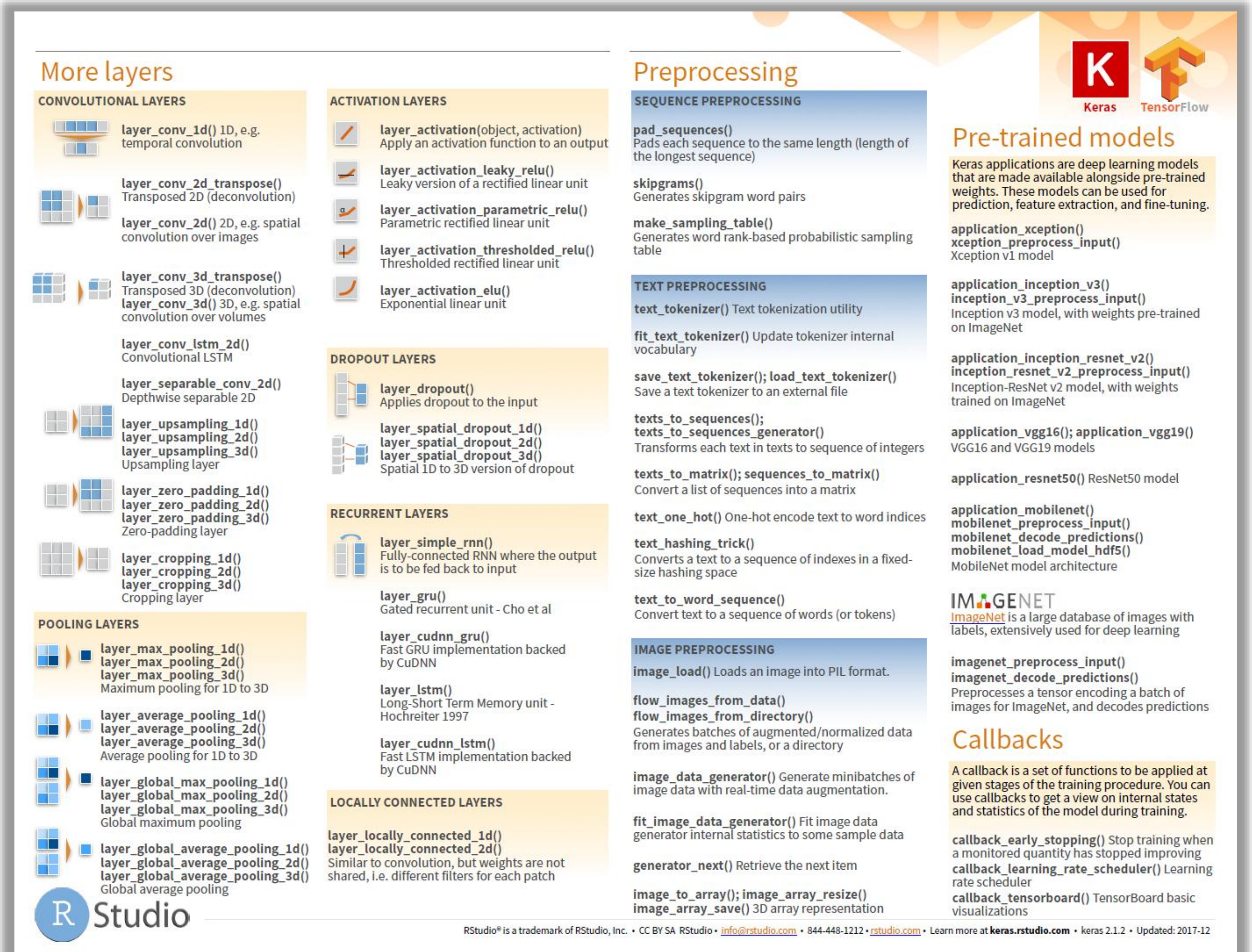
MARQUETTE UNIVERSITY
Be The Difference.

RECOMMENDED READINGS

- Datacamp Tutorials:
  - Keras: Deep Learning in R
    - https://www.datacamp.com/community/tutorials/keras-r-deep-learning
  - Keras Tutorial: Deep Learning in Python
    - https://www.datacamp.com/community/tutorials/deep-learning-python
  - TensorFlow Tutorial For Beginners
    - https://www.datacamp.com/community/tutorials/tensorflow-tutorial
- Deep Learning Specializations in Coursera
- Keras in R
  - https://keras.rstudio.com/
  - https://tensorflow.rstudio.com/guides/keras/basics
- Tensorflow in R
  - https://tensorflow.rstudio.com/
  - https://tensorflow.rstudio.com/tutorials/

KERAS FOR R CHEATSHEET

- https://rstudio.github.io/cheatsheets/html/keras.html



More layers

CONVOLUTIONAL LAYERS

layer_conv_1d() 1D, e.g. temporal convolution

layer_conv_2d_transpose() Transposed 2D (deconvolution)

layer_conv_2d() 2D, e.g. spatial convolution over images

layer_conv_3d_transpose() Transposed 3D (deconvolution)
layer_conv_3d() 3D, e.g. spatial convolution over volumes

layer_conv_lstm_2d() Convolutional LSTM

layer_separable_conv_2d() Depthwise separable 2D

layer_upsampling_1d()
layer_upsampling_2d()
layer_upsampling_3d()
Upsampling layer

layer_zero_padding_1d()
layer_zero_padding_2d()
layer_zero_padding_3d()
Zero-padding layer

layer_cropping_1d()
layer_cropping_2d()
layer_cropping_3d()
Cropping layer

POOLING LAYERS

layer_max_pooling_1d()
layer_max_pooling_2d()
layer_max_pooling_3d()
Maximum pooling for 1D to 3D

layer_average_pooling_1d()
layer_average_pooling_2d()
layer_average_pooling_3d()
Average pooling for 1D to 3D

layer_global_max_pooling_1d()
layer_global_max_pooling_2d()
layer_global_max_pooling_3d()
Global maximum pooling

layer_global_average_pooling_1d()
layer_global_average_pooling_2d()
layer_global_average_pooling_3d()
Global average pooling

ACTIVATION LAYERS

layer_activation(object, activation) Apply an activation function to an output

layer_activation_leaky_relu() Leaky version of a rectified linear unit

layer_activation_parametric_relu() Parametric rectified linear unit

layer_activation_thresholded_relu() Thresholded rectified linear unit

layer_activation_elu() Exponential linear unit

DROPOUT LAYERS

layer_dropout() Applies dropout to the input

layer_spatial_dropout_1d()
layer_spatial_dropout_2d()
layer_spatial_dropout_3d()
Spatial 1D to 3D version of dropout

RECURRENT LAYERS

layer_simple_rnn() Fully-connected RNN where the output is to be fed back to input

layer_gru() Gated recurrent unit - Cho et al

layer_cudnn_gru() Fast GRU implementation backed by CuDNN

layer_lstm() Long-Short Term Memory unit - Hochreiter 1997

layer_cudnn_lstm() Fast LSTM implementation backed by CuDNN

LOCALLY CONNECTED LAYERS

layer_locally_connected_1d()
layer_locally_connected_2d()
Similar to convolution, but weights are not shared, i.e. different filters for each patch

Preprocessing

SEQUENCE PREPROCESSING

pad_sequences() Pads each sequence to the same length (length of the longest sequence)

skipgrams() Generates skipgram word pairs

make_sampling_table() Generates word rank-based probabilistic sampling table

TEXT PREPROCESSING

text_tokenizer() Text tokenization utility

fit_text_tokenizer() Update tokenizer internal vocabulary

save_text_tokenizer(); load_text_tokenizer() Save a text tokenizer to an external file

texts_to_sequences(); texts_to_sequences_generator() Transforms each text in texts to sequence of integers

texts_to_matrix(); sequences_to_matrix() Convert a list of sequences into a matrix

text_one_hot() One-hot encode text to word indices

text_hashing_trick() Converts a text to a sequence of indexes in a fixed-size hashing space

text_to_word_sequence() Convert text to a sequence of words (or tokens)

IMAGE PREPROCESSING

image_load() Loads an image into PIL format.

flow_images_from_data()
flow_images_from_directory()
Generates batches of augmented/normalized data from images and labels, or a directory

image_data_generator() Generate minibatches of image data with real-time data augmentation.

fit_image_data_generator() Fit image data generator internal statistics to some sample data

generator_next() Retrieve the next item

image_to_array(); image_array_resize()
image_array_save() 3D array representation

Keras TensorFlow

Pre-trained models

Keras applications are deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning.

application_xception()
xception_preprocess_input()
Xception v1 model

application_inception_v3()
inception_v3_preprocess_input()
Inception v3 model, with weights pre-trained on ImageNet

application_inception_resnet_v2()
inception_resnet_v2_preprocess_input()
Inception-ResNet v2 model, with weights trained on ImageNet

application_vgg16(); application_vgg19() VGG16 and VGG19 models

application_resnet50() ResNet50 model

application_mobilenet()
mobilenet_preprocess_input()
mobilenet_decode_predictions()
mobilenet_load_model_hdf5()
MobileNet model architecture

IMAGENET
ImageNet is a large database of images with labels, extensively used for deep learning

imagenet_preprocess_input()
imagenet_decode_predictions()
Preprocesses a tensor encoding a batch of images for ImageNet, and decodes predictions

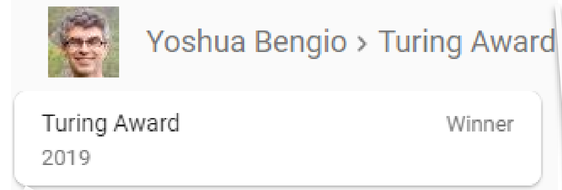
Callbacks

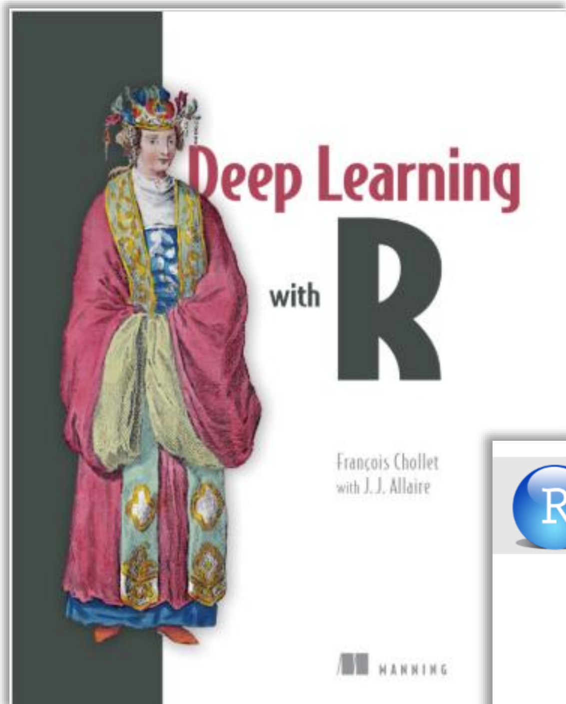
A callback is a set of functions to be applied at given stages of the training procedure. You can use callbacks to get a view on internal states and statistics of the model during training.

callback_early_stopping() Stop training when a monitored quantity has stopped improving
callback_learning_rate_scheduler() Learning rate scheduler
callback_tensorboard() TensorBoard basic visualizations

R Studio

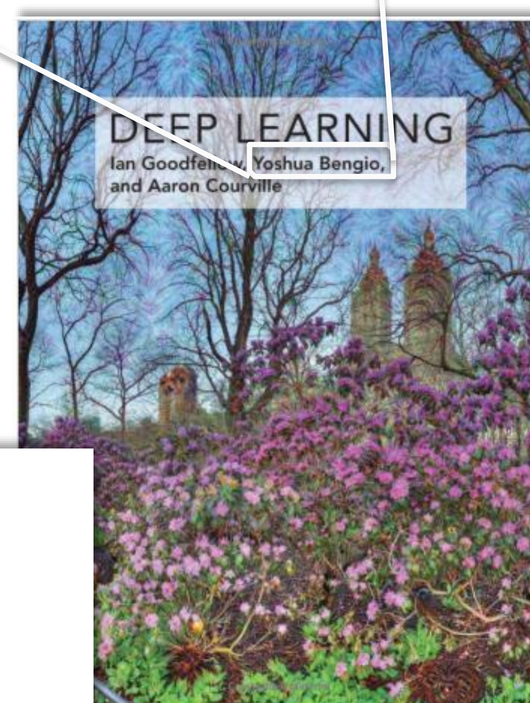
REFERENCES





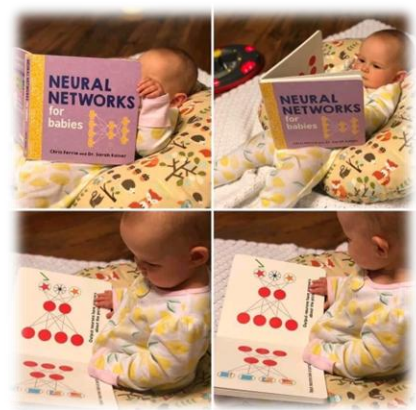
QUESTIONS?

THANK YOU!



Machine Learning with TensorFlow and R

J.J. Allaire — CEO, RStudio



MARQUETTE UNIVERSITY
Be The Difference.