

# Keras 3 for R

Deep Learning for Everyone.

Simple. Flexible. Powerful.

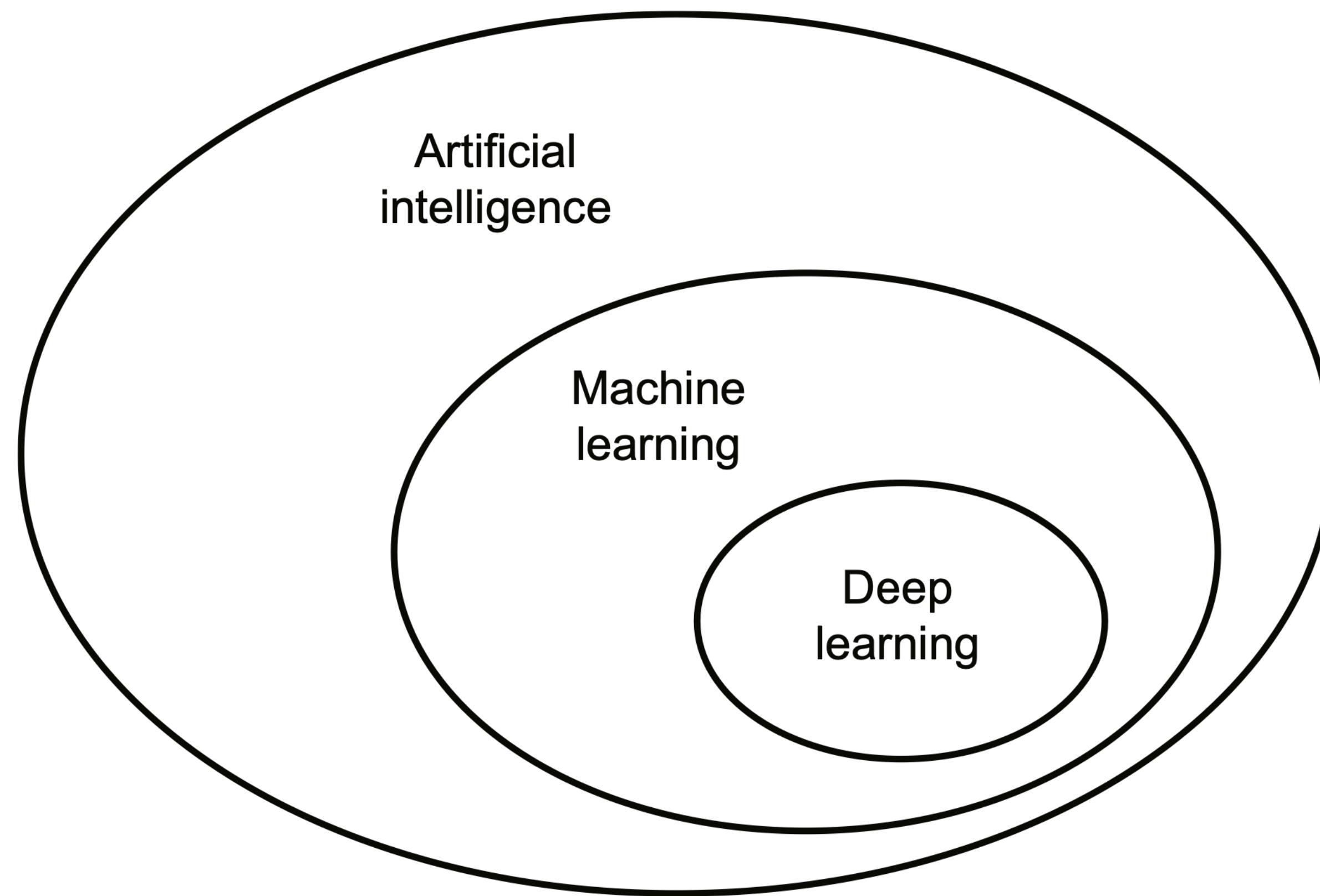
Tomasz Kalinowski / Aug-13-2024

FROM  **posit**

**posit**  
conf (2024)



# Deep Learning





## Deep Learning:

- First described over 50 years ago
- Relatively niche at first
- Today, dominant approach in:
  - Computer vision
  - Speech Recognition
  - Natural Language Processing





# Why is Deep Learning outpacing other AI/ML techniques?





The screenshot shows a web browser window with the following details:

- Title Bar:** Not Secure — www.incompleteideas.net/Incldeas/BitterLesson.htm
- Section Header:**

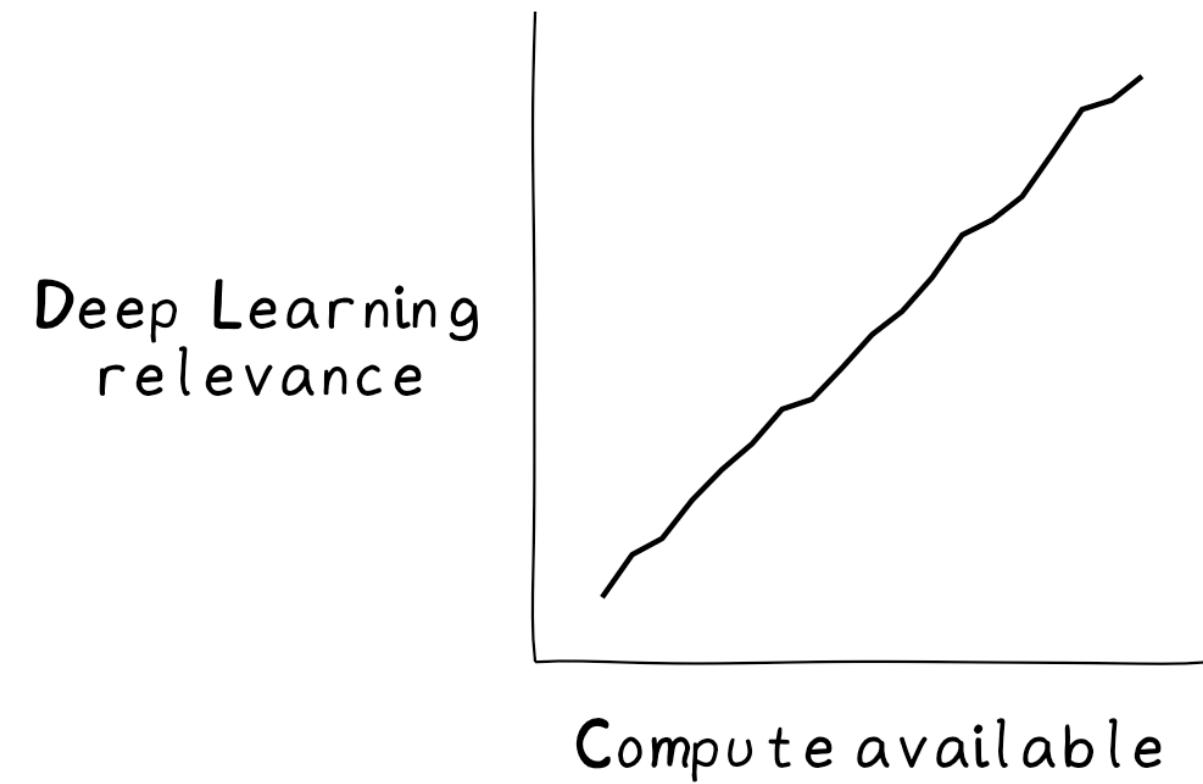
# The Bitter Lesson
- Author:** **Rich Sutton**
- Date:** March 13, 2019
- Text Content:** A detailed paragraph discussing the evolution of AI research, focusing on Moore's law and the increasing availability of computation over time, and how it contrasts with traditional knowledge-based approaches.
- Bottom Text:** A summary of the chess victory over Kasparov, highlighting the shift from human-knowledge-based methods to general computation-based methods.

In computer chess, the methods that defeated the world champion, Kasparov, in 1997, were based on massive, deep search. At the time, this was looked upon with dismay by the majority of computer-chess researchers who had pursued methods that leveraged human understanding of the special structure of chess. When a simpler, search-based approach with special hardware and software proved vastly more effective, these human-knowledge-based chess researchers were not good losers. They said that "brute force" search may have won this time, but it was not a general strategy, and anyway it was not how people played chess. These researchers wanted methods based on human input to win and were disappointed



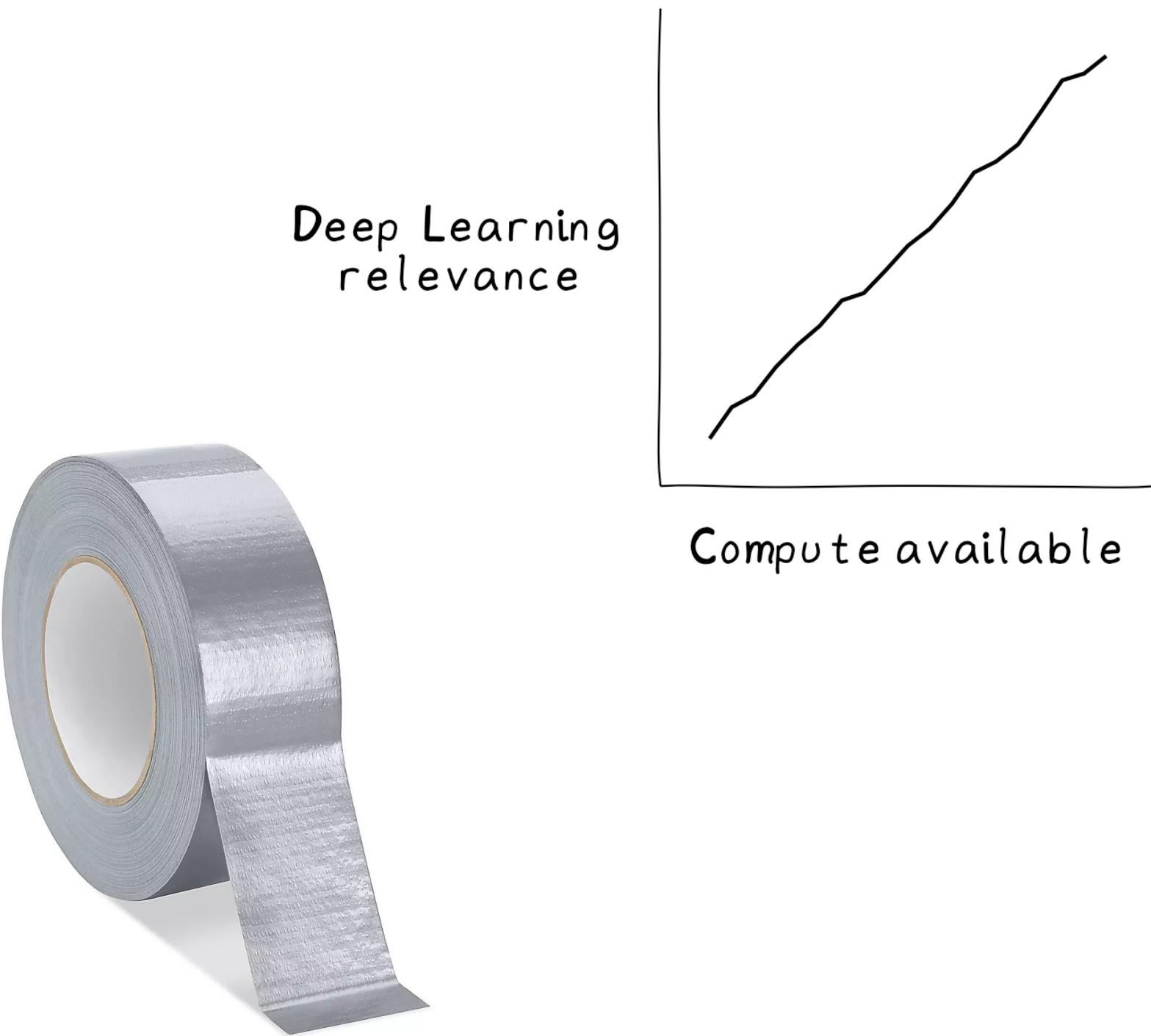
# Deep Learning

- Capabilities scale with compute



# Deep Learning

- Capabilities scale with compute
- **General purpose, versatile**



Deep Learning  
relevance

Compute available

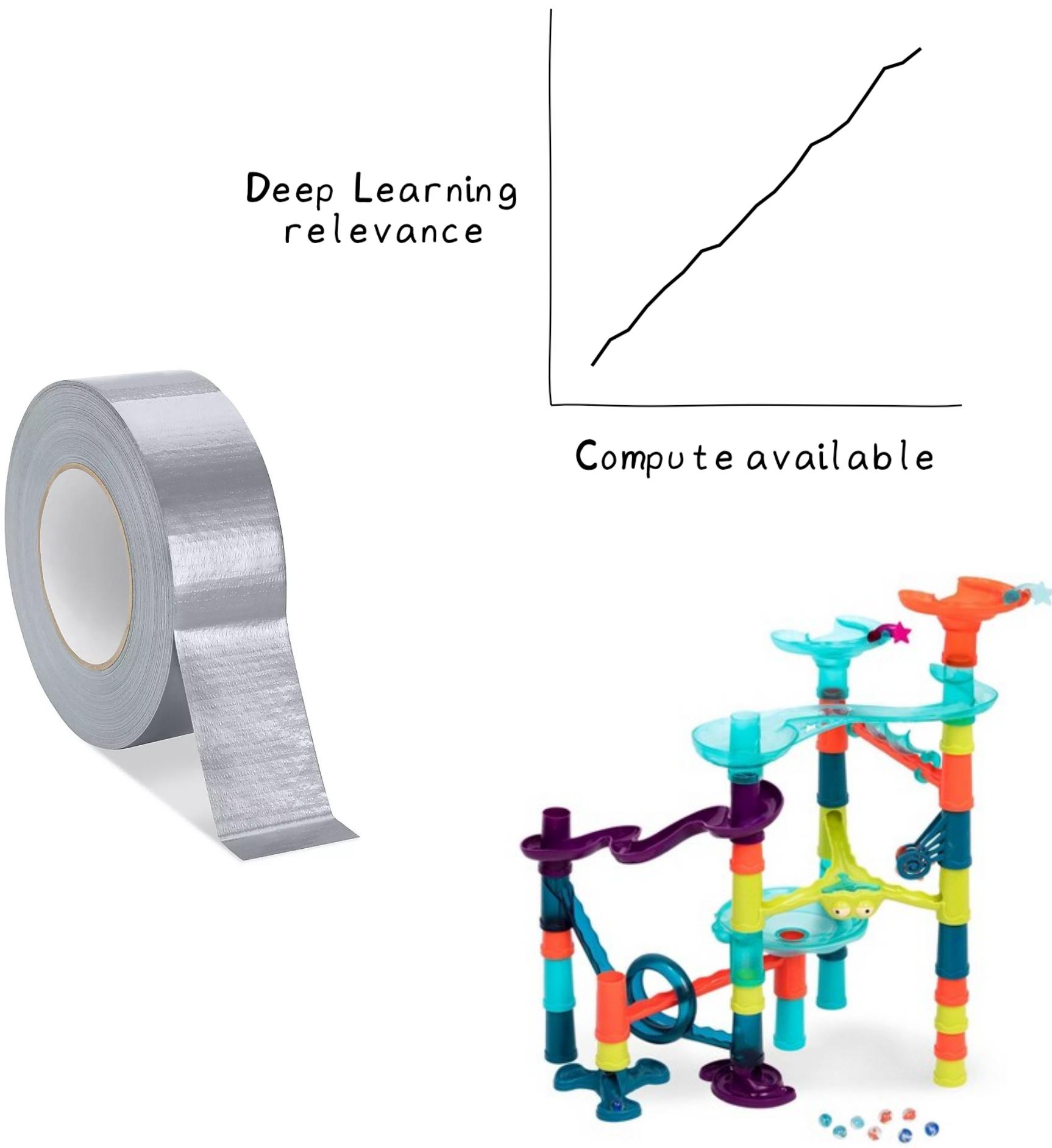
# Deep Learning: General Purpose, Versatile

- Applicable to all kinds of *data*:  
text, image, audio, video, tabular, time series,  
graphs, ...
- Applicable to all kinds of *tasks*:  
classification, regression, segmentation,  
imputation, denoising, clustering,  
similarity analysis, compression, ranking,  
recommender systems, style transfer,  
generate text/images/video ...



# Deep Learning

- Capabilities scale with compute
- General purpose, versatile
- **Forgiving, reliable**



# Keras 3: A Kit for Deep Learning.

- Comes with many kinds of layers.
- Layers compose with a common, simple, interface.
- Create paths for data (marbles).
- Designing models is fun!
- Feeding models data is fun!





## Keras 3: A Toolkit for Deep Learning.

Create, train, evaluate and deploy deep learning models

- A suite of building blocks:
  - Layers (104)
  - Metrics (46)
  - Loss functions (20)
  - Optimizers (12)
  - Data loading and preprocessing (10)
  - Operations (>200)
- High-level training API: `fit()`





MNIST Database of Handwritten Digits.

The "Hello World" of Deep Learning.

5 0 4 / 9 2 1 3 1 4 3 5

3 6 1 7 2 8 6 9 4 0 9 1

1 2 4 3 2 7 3 8 6 9 0 5

- 60,000 training images

- 10,000 test images



&gt;

## Simple MNIST Image Classifier

```
input <- keras_input(shape = c(28, 28)) # Each image is 28x28 pixels
```

&lt;

&gt;

## Simple MNIST Image Classifier

```
input <- keras_input(shape = c(28, 28))

output <- input |>
  layer_flatten() |>
  layer_dense(10, activation = "softmax")
```

&lt;

&gt;

## Simple MNIST Image Classifier

```
input <- keras_input(shape = c(28, 28))

output <- input |>
  layer_flatten() |>
  layer_dense(10, activation = "softmax")

model <- keras_model(input, output)
model
```

&lt;

&gt;

# Model Visualizations

```
> print(model)
```

**Model: "functional"**

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, 28, 28)	0
flatten (Flatten)	(None, 784)	0
dense (Dense)	(None, 10)	7,850

Total params: 7,850 (30.66 KB)

Trainable params: 7,850 (30.66 KB)

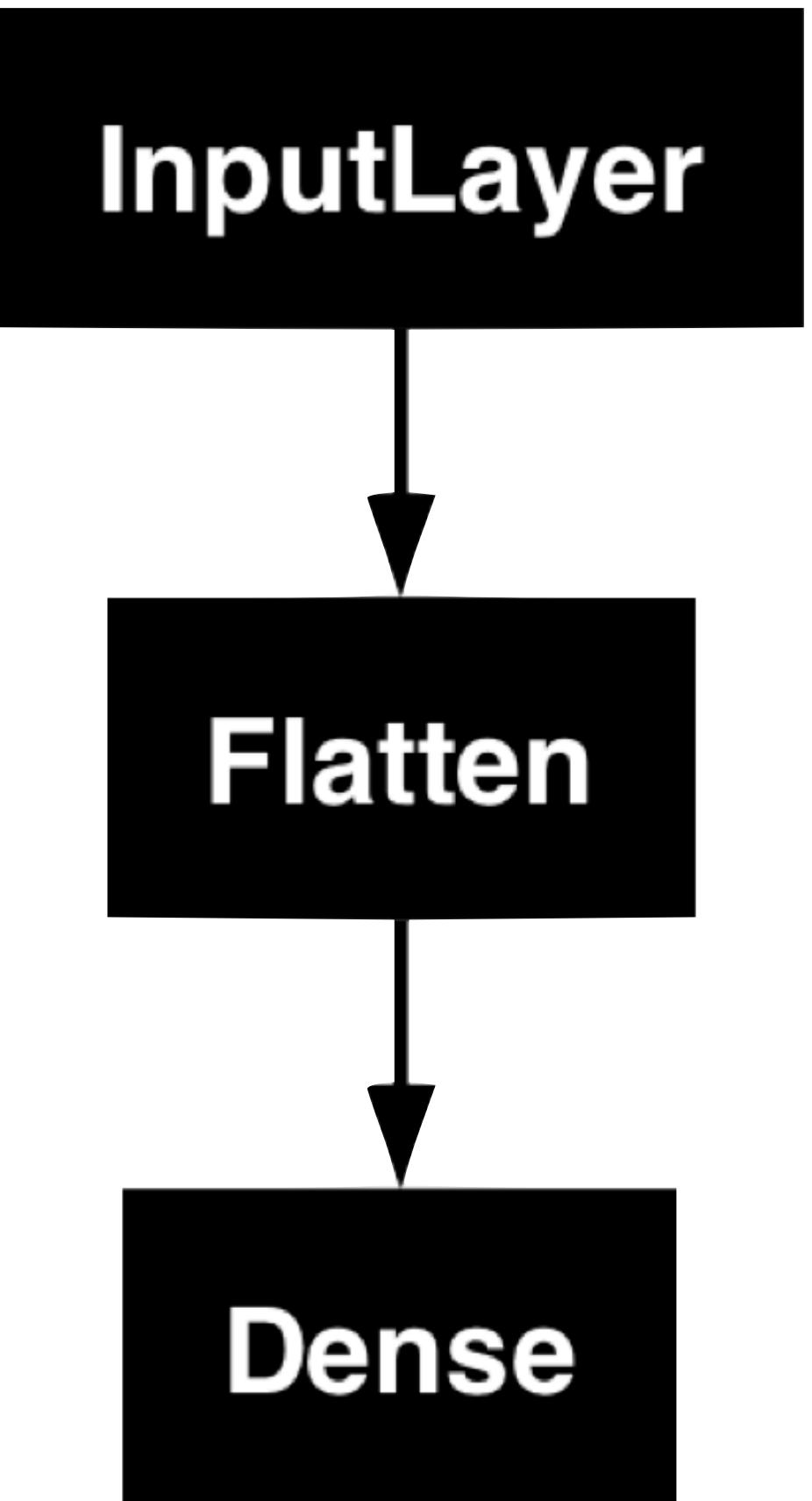
Non-trainable params: 0 (0.00 B)

&lt;

>

# Model Visualizations

> `plot(model)`

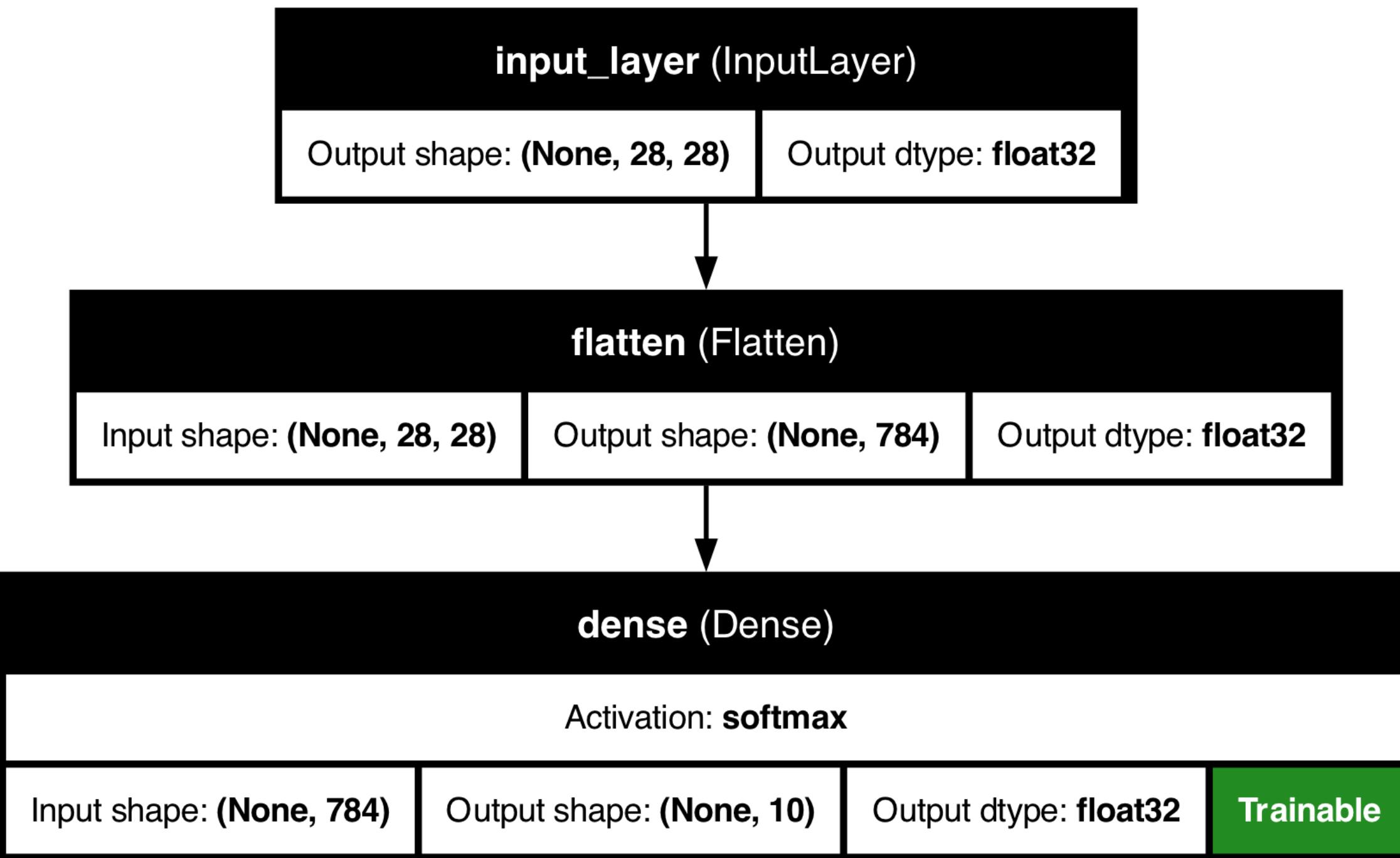


<

&gt;

# Model Visualizations

```
> plot(model,  
       show_shapes = TRUE,  
       show_dtype = TRUE,  
       show_trainable = TRUE,  
       show_layer_activations = TRUE,  
       show_layer_names = TRUE)
```



&lt;

>

## High Level Training API

`compile()`    `fit()`

<

&gt;

## High Level Training API

`compile()`    `fit()`

```
model |> compile(  
  loss = loss_sparse_categorical_crossentropy(),  
  optimizer = optimizer_adam(),  
  metrics = metric_categorical_accuracy()  
)
```

&lt;

&gt;

## High Level Training API

`compile()`    `fit()`

```
model |> compile(  
  loss = loss_sparse_categorical_crossentropy(),  
  optimizer = optimizer_adam(),  
  metrics = metric_categorical_accuracy()  
)
```

```
model |> fit(  
  x_train, y_train,  
  epochs = 10  
)
```

&lt;

&gt;

## Simple MNIST Image Classifier

```
input <- keras_input(shape = c(28, 28))  
  
output <- input |>  
  layer_flatten() |>  
  layer_dense(10, activation = "softmax")  
  
model <- keras_model(input, output)
```

Trains to 92.1%  
accuracy!

&lt;

&gt;

## Simple MNIST Image Classifier

Trains to 97.5%  
accuracy!

```
input <- keras_input(shape = c(28, 28))  
  
output <- input |>  
  layer_flatten() |>  
  layer_dense(128, activation = "relu") |>  
  layer_dropout(.5) |>  
  layer_dense(10, activation = "softmax")
```

```
model <- keras_model(input, output)  
model
```

&lt;

&gt;

## Simple MNIST Image Classifier

Trains to 98.7%  
accuracy!

```
input <- keras_input(shape = c(28, 28))  
  
output <- input |>  
  layer_reshape(c(28, 28, 1)) |>  
  layer_conv_2d(32, c(3, 3), activation = 'relu') |>  
  layer_max_pooling_2d() |>  
  layer_flatten() |>  
  layer_dense(128, activation = "relu") |>  
  layer_dense(10, activation = "softmax")  
  
model <- keras_model(input, output)  
model
```

&lt;



## Simple MNIST Image Classifier

```
input <- keras_input(shape = c(28, 28))  
  
output <- input |>  
  layer_reshape(c(28, 28, 1)) |>  
  layer_rescaling(scale = 1/255) |>  
  layer_conv_2d(32, c(3, 3), activation = 'relu') |>  
  layer_max_pooling_2d() |>  
  layer_conv_2d(64, c(3, 3), activation = 'relu') |>  
  layer_max_pooling_2d() |>  
  layer_flatten() |>  
  layer_dropout(0.5) |>  
  layer_dense(10, activation = "softmax")  
  
model <- keras_model(input, output)
```

Trains to 99.2%  
accuracy!



>

The screenshot shows an RStudio session with the following components:

- Code Editor:** The file `mnist-convnet-example.R` contains R code for building and training a Keras model. Key parts include:
 

```

15   layer_dropout(0.5) |>
16   layer_dense(10, activation = "softmax")
17
18 model <- keras_model(input, output)
19
20 model |> compile(
21   loss = loss_sparse_categorical_crossentropy(),
22   optimizer = optimizer_adam(),
23   metrics = "accuracy"
24 )
25
26 summary(model)
27
28 model |> fit(
29   x_train, y_train,
30   epochs = 10,
31   validation_split = 0.1
32 )
33
34 model |> evaluate(x_test, y_test)
35
      
```
- Environment Browser:** Shows the global environment with variables:
 

Variable	Type	Description
input	<KerasTensor shape=(None, 28, 28)...	
output	<KerasTensor shape=(None, 10), dt...	
model	<Functional name=functional_3, buil...	
x_test	Large array (7840000 elements, 31...	
x_train	Large array (47040000 elements, 18...	
y_test	int [1:10000(1d)] 7 2 1 0 4 1 4 9 5...	
y_train	int [1:60000(1d)] 5 0 4 1 9 2 1 3 1...	
- Console:** Displays the R session output, including the model summary, training progress, and evaluation results. The total parameters are 34,826 (136.04 KB). Training took 1688 steps over 10 epochs, with accuracy and validation accuracy increasing from ~0.84 to ~0.99, and loss decreasing from ~0.47 to ~0.04.
- Plots:** Two line plots show the training progress. The top plot shows accuracy and validation accuracy (blue line with circles) increasing from epoch 1 to 5. The bottom plot shows loss and validation loss (blue line with circles) decreasing from epoch 1 to 5.

&gt;

# Keras: Next Steps After MNIST



BEGINNER



INTERMEDIATE



ADVANCED



EXPERT

&lt;

[https://keras.io/guides/keras\\_nlp/getting\\_started/](https://keras.io/guides/keras_nlp/getting_started/)

>

## Keras: Next Steps After MNIST

# Progressive Disclosure of Complexity

<

&gt;

## Defining Custom Layers layer\_lambda()

```
input <- keras_input(c(28, 28), dtype = "uint8")
output <- input |>
  layer_lambda(\(x) op_cast(x, "float32") / 255) |>
  layer_... |> layer_...
```

&lt;

&gt;

## Defining Custom Layers

### `keras_model()`

```
model <- keras_model(...)

input <- keras_input(c(28, 28), dtype = "uint8")
output <- input |>
  model() |>
  layer_... |> layer_...
```

&lt;

&gt;

# Defining Custom Layers

## Layer()

```
layer_dense2 <- Layer("Dense2",
  initialize = function(...) {}, 
  call = function(...) {}, 
  build = function(...) {}, 
  ...
)

input <- keras_input(c(28, 28))
output <- input |>
  layer_dense2() |>
  layer_... |> layer_...
```

&lt;

&gt;

## Defining Custom Layers

### Layer()

```
# Subclass base Layer class  
Layer(...)
```

```
# Subclass a built-in layer  
Layer(..., inherit = layer_dense)
```

```
# Subclass a custom layer  
Layer(..., inherit = layer_dense2)
```

&lt;

>

## Subclassing API

- Define custom objects that can manage state
- All support inheritance

`Layer()`

`Callback()`

`Metric()`

`Model()`

`LearningRateSchedule()`

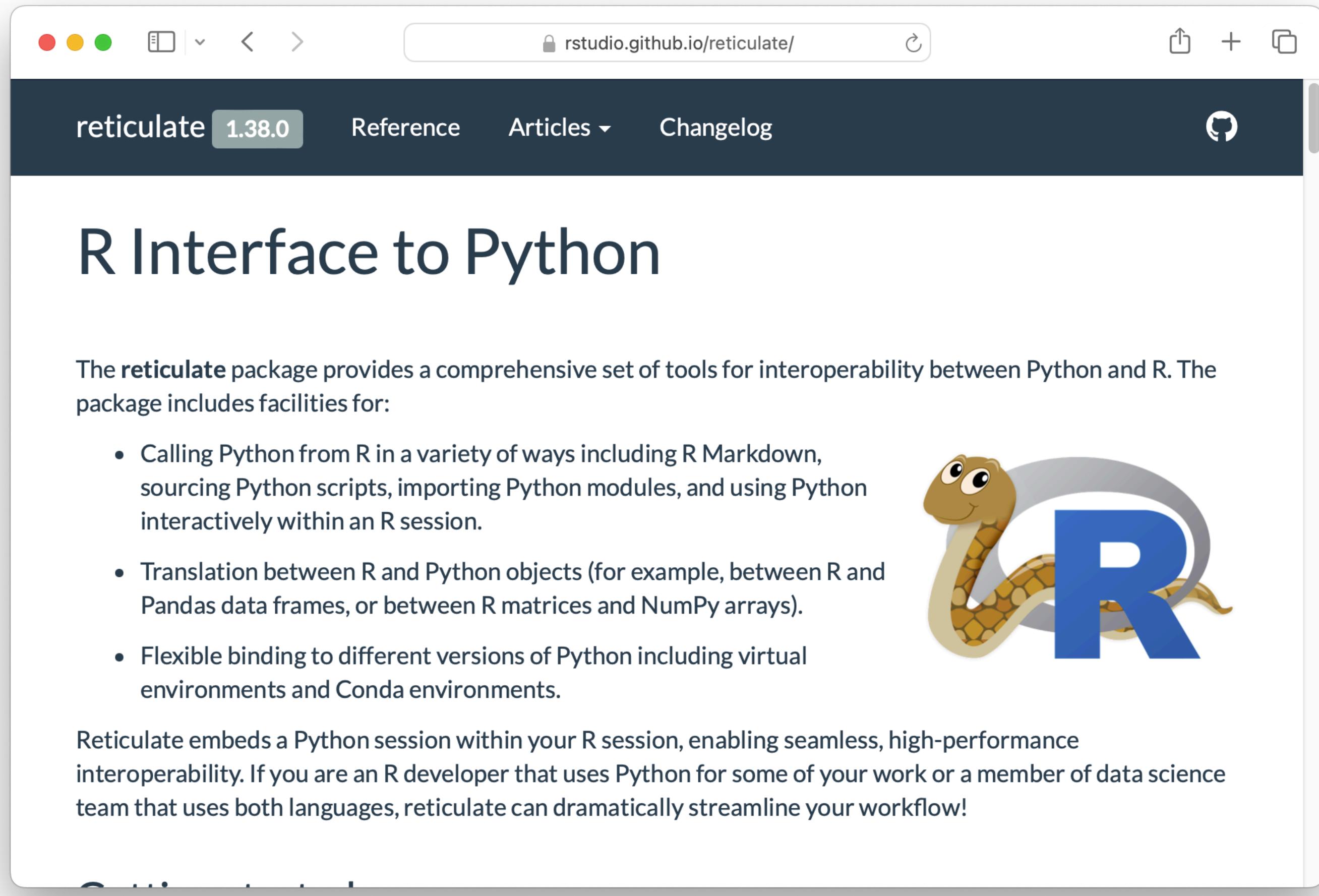
`Constraint()`

`Loss()`

<



# Reticulate: Access everything in Python from R



The screenshot shows a web browser window displaying the official documentation for the `reticulate` package. The URL in the address bar is `rstudio.github.io/reticulate/`. The page header includes the `reticulate` logo (a white cat icon), the version `1.38.0`, and navigation links for `Reference`, `Articles`, and `Changelog`. The main title is 

## R Interface to Python

. Below the title, a paragraph explains that the `reticulate` package provides tools for interoperability between Python and R, mentioning facilities for calling Python from R, translating between objects, and flexible binding. To the right of the text is the `CR` logo, where the `C` is replaced by a cartoon snake. At the bottom of the page, there is a footer with links to GitHub, CRAN, and other resources.

The `reticulate` package provides a comprehensive set of tools for interoperability between Python and R. The package includes facilities for:

- Calling Python from R in a variety of ways including R Markdown, sourcing Python scripts, importing Python modules, and using Python interactively within an R session.
- Translation between R and Python objects (for example, between R and Pandas data frames, or between R matrices and NumPy arrays).
- Flexible binding to different versions of Python including virtual environments and Conda environments.

Reticulate embeds a Python session within your R session, enabling seamless, high-performance interoperability. If you are an R developer that uses Python for some of your work or a member of data science team that uses both languages, `reticulate` can dramatically streamline your workflow!



&gt;

## Reticulate: Latest Updates

- Improved error tracebacks
- Improved Python function arguments autocomplete
- Built-in support for R binary operator generics (+, /, \*, ...)
- Updated Python environment discovery and management.

&lt;

## > Keras 3: What's New

- Return of multi-backend support
  - New Function families:
    - Ops
    - Random
  - New Saving & Serialization API



&gt;

# Keras 3: Return of Multi-Backend Support

- Backends are lower-level frameworks that provide:
  - Automatic differentiation and gradient computation
  - Accelerator support (GPUs)
- Prior to 2021, Keras supported multiple backends:
  - TensorFlow, Theano, MXNet, and CNTK
- In 2021, TensorFlow became the only supported backend, and Keras became an official part of the TensorFlow.

&lt;

&gt;

# Keras 3: Return of Multi-Backend Support

- Keras is once again a stand-alone package  
(not part of TensorFlow)
- Supported backends:
  - Jax
  - PyTorch
  - TensorFlow
  - NumPy
  - (MLX soon?)

```
library(keras3)
use_backend("jax")
```

&lt;

&gt;

# Keras 3: Return of Multi-Backend Support

- Write future-proof code.
- Easily switch backends at any time:
  - Side-step backend bugs or unimplemented feature.
  - Take advantage of the best each backend has to offer.
- Example workflow:
  - Write the model using Torch (nice interactive experience)
  - Train it using Jax (fastest for large models)
  - Deploy it using TensorFlow (most mature deployment options)

&lt;

&gt;

# Saving & Loading

New backend-agnostic file format.

Convert models between different backends

```
library(keras3)
use_backend("jax")
model <- keras_model(...)
model |> fit(...)
model |> save_model("my_model.keras")

# ---- A different R Session ---
library(keras3)
use_backend("tensorflow")
restored_model <- load_model("my_model.keras")
```

&lt;



# Deploy Keras Models to Connect

```
export_savedmodel(model, "model-for-serving")
rsconnect::deployTFModel("model-for-serving")
```

- No R or Python Runtime dependency
- Scalable, concurrent model serving

The screenshot shows the rsconnect interface for a deployed TensorFlow MNIST example. The main content area displays the following information:

- Model Name: default**
- Model Version: 1**
- A message stating: "Your TensorFlow Model API is running. To get model predictions, send a POST request to the model server prediction URL, which is the URL for this page with an added :predict suffix."
- A code snippet for the prediction endpoint: `POST https://rsc.radixu.com/content/f7d4fe14-cb43-4475-a45b-ef134c899606/v1/models/default:predict`
- A note: "You can optionally also specify a specific model version or label by appending them to the base URL:" followed by two examples: `POST https://rsc.radixu.com/content/f7d4fe14-cb43-4475-a45b-ef134c899606/v1/models/default/versions/${VERSION}:predict` and `POST https://rsc.radixu.com/content/f7d4fe14-cb43-4475-a45b-ef134c899606/v1/models/default/labels/${LABEL}:predict`
- Default Serving Signature**
- Input Data:** A note stating "The model default serving signature expects this POST data:" followed by a JSON snippet: `{ "inputs": <>A tensor of strings with shape (<unknown>)>> }`
- Response Data:** A note stating "The model default serving signature will return this data:" followed by a JSON snippet: `{ "outputs": { "classes": <>A tensor of strings with shape (<unknown>)>>, "scores": <>A tensor of floating point values with shape (<unknown>)>> } }`

The right sidebar contains navigation tabs: Info, Access (selected), Runtime, Schedule, Tags, Vars. It also shows sharing settings (All users - login required), a list of viewers (Aron Atkins aron), and a content URL field with the value `https://rsc.radixu.com/content/f7d4fe14-cb43-4475-a45b-ef134c899606`.

&gt;

# Keras 3 Multi-Backend Support: Ops

- Suite of 200+ lower-level functions for working with multi-dimensional arrays

`op_sum()`

`op_abs()`

`op_slice_update()`

`op_average_pool()`

`op_isfinite()`

`op_isnan()`

`op_relu6()`

`op_maximum()`

`op_eig()`

`op_arctan()`

...

`op_roll()`

`op_ctc_decode()`

`op_ravel()`

`op_normalize()`

`op_qr()`

`op_softmax()`

`op_one_hot()`

`op_softplus()`

`op_zeros()`

`op_argsort()`

...

`op_identity()`

`op_imag()`

`op_square()`

`op_elu()`

`op_add()`

`op_binary_crossentropy()`

`op_lu_factor()`

`op_gelu()`

`op_custom_gradient()`

`op_diagonal()`

...

&lt;

&gt;

## Keras 3 Multi-Backend Support: Ops

- Uniform API for:
  - TensorFlow, Torch, Jax Tensors
  - Eager and Graph/JIT/Tracing mode
  - NumPy nd-arrays
  - R arrays
- Write backend-agnostic code.

&lt;

&gt;

## New Operations family

- Use 'Ops' anywhere you operate with backend or symbolic tensors

```
layer_lambda()  
Layer(call = )  
compile(loss = , metrics = )
```

- Use 'Ops' as stand-alone layers

```
input <- keras_input(shape = c(28, 28))  
output <- input |>  
  op_image_pad(target_height = 30, target_width = 30)  
  
keras_model(input, output)
```

&lt;

&gt;

# Keras 3 Multi-Backend Support: Ops

Includes:

- The NumPy API (`numpy.*`)
- The TensorFlow NN API (`tf.nn.*`)
- Linear algebra ops (`scipy.linalg.*`)
- Image ops
- And more!

&lt;

&gt;

## Keras 3 Ops: For R users

- Use it as a R-friendly alternative API to using the Python modules via reticulate:
  - Consistent 1-based indexing
  - Accepts integer-ish doubles for integers
  - Uses R built-in docs, with R examples.
- All functions are pure (easy to reason about)

&lt;

## > Keras 3: What (else) is New

- Updated APIs for
  - Defining custom objects by subclassing
  - Tabular data
  - Generating random numbers
    - (supports usage in pure functions).
  - Distributed training
  - Quantization support
  - Custom training steps/loops



> Keras 3 Package website: <https://keras3.posit.co>

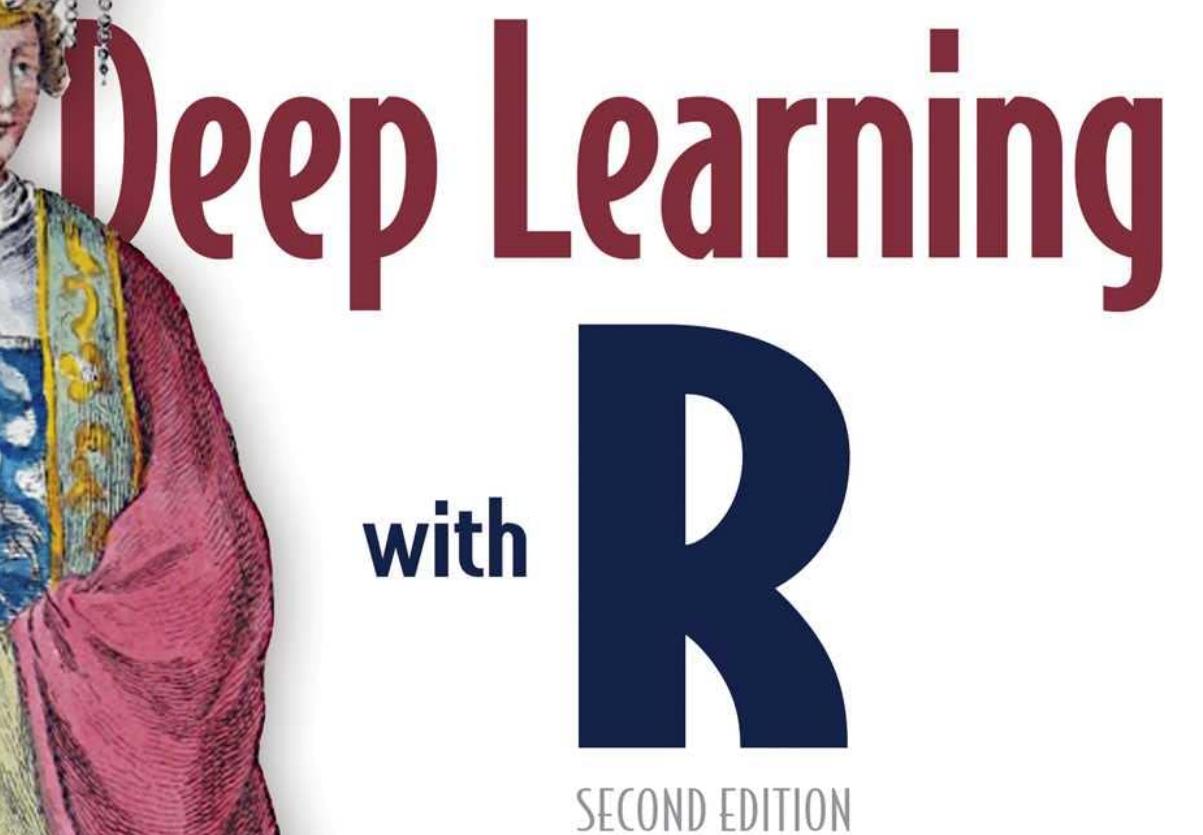
The screenshot shows a web browser window displaying the Keras 3 package website at <https://keras3.posit.co>. The page has a red header bar with the text "keras3 1.0.0" and navigation links for "Getting Started", "Guides", "Examples", "Reference", and "News". A search bar and a GitHub icon are also in the header. The main content features a large "R interface to Keras" title with a red square logo containing a white "K". Below the title, there are badges for "R-CMD-check passing", "CRAN 2.15.0", and "license MIT". A paragraph describes Keras as a high-level neural networks API developed with a focus on enabling fast experimentation. It lists key features: running on CPU or GPU, a user-friendly API for prototyping, and built-in support for convolutional and recurrent networks. On the right side, there are sections for "Links" (View on CRAN, Browse source code, Report a bug), "License" (MIT + file LICENSE), "Citation" (Citing keras3), and "Developers" (Tomasz Kalinowski, JJ Allaire, François Chollet, Posit Software, PBC). A small "Copyright holder funder" note is visible at the bottom of the developer section.



## Learn More

The **Deep Learning with R** book shows you how to get started with Keras in R, even if you have no background in mathematics or data science. The book covers:

- Deep learning from first principles
- Image classification and image segmentation
- Time series forecasting
- Text classification and machine translation
- Text generation, neural style transfer, and image generation



François Chollet  
with Tomasz Kalinowski  
and J. J. Allaire



**posit**  
conf(2024)

Thank you.

