Keras 3 for R

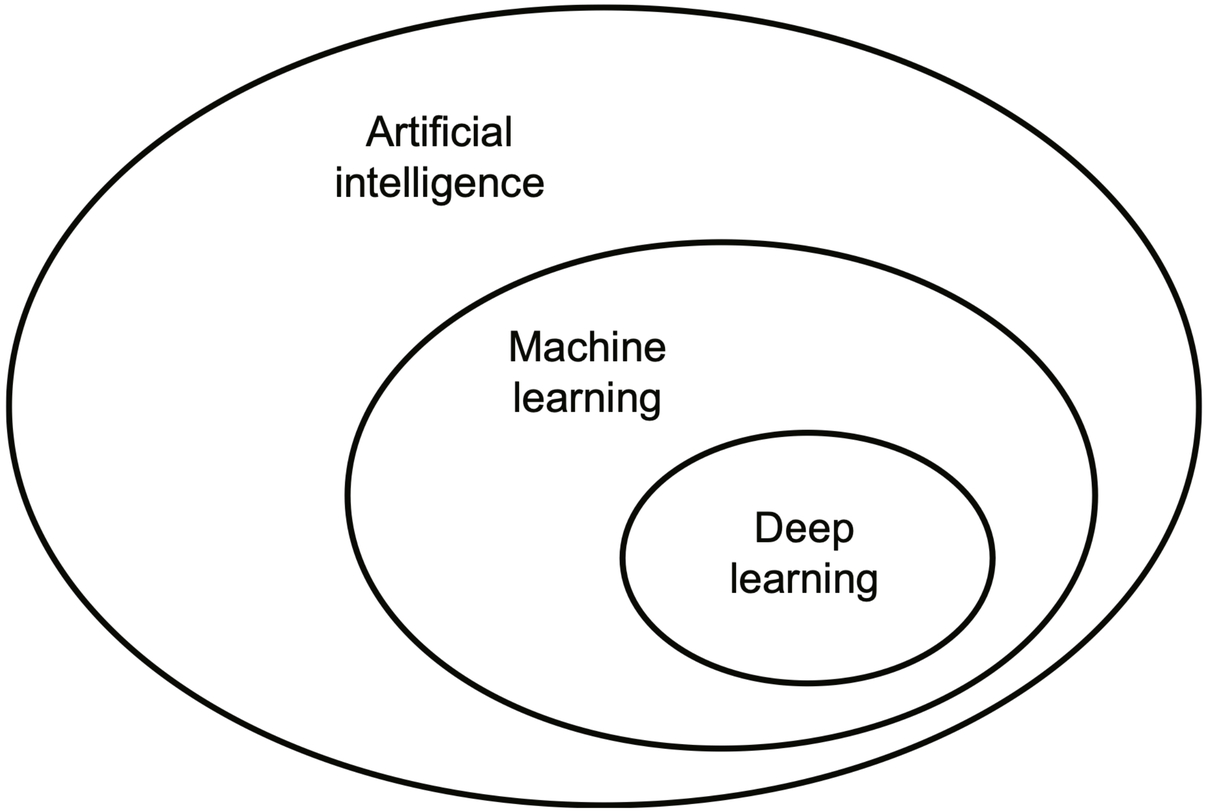
Deep Learning for Everyone.
Simple. Flexible. Powerful.

Tomasz Kalinowski / Aug-13-2024

posit conf(2024)

FROM posit

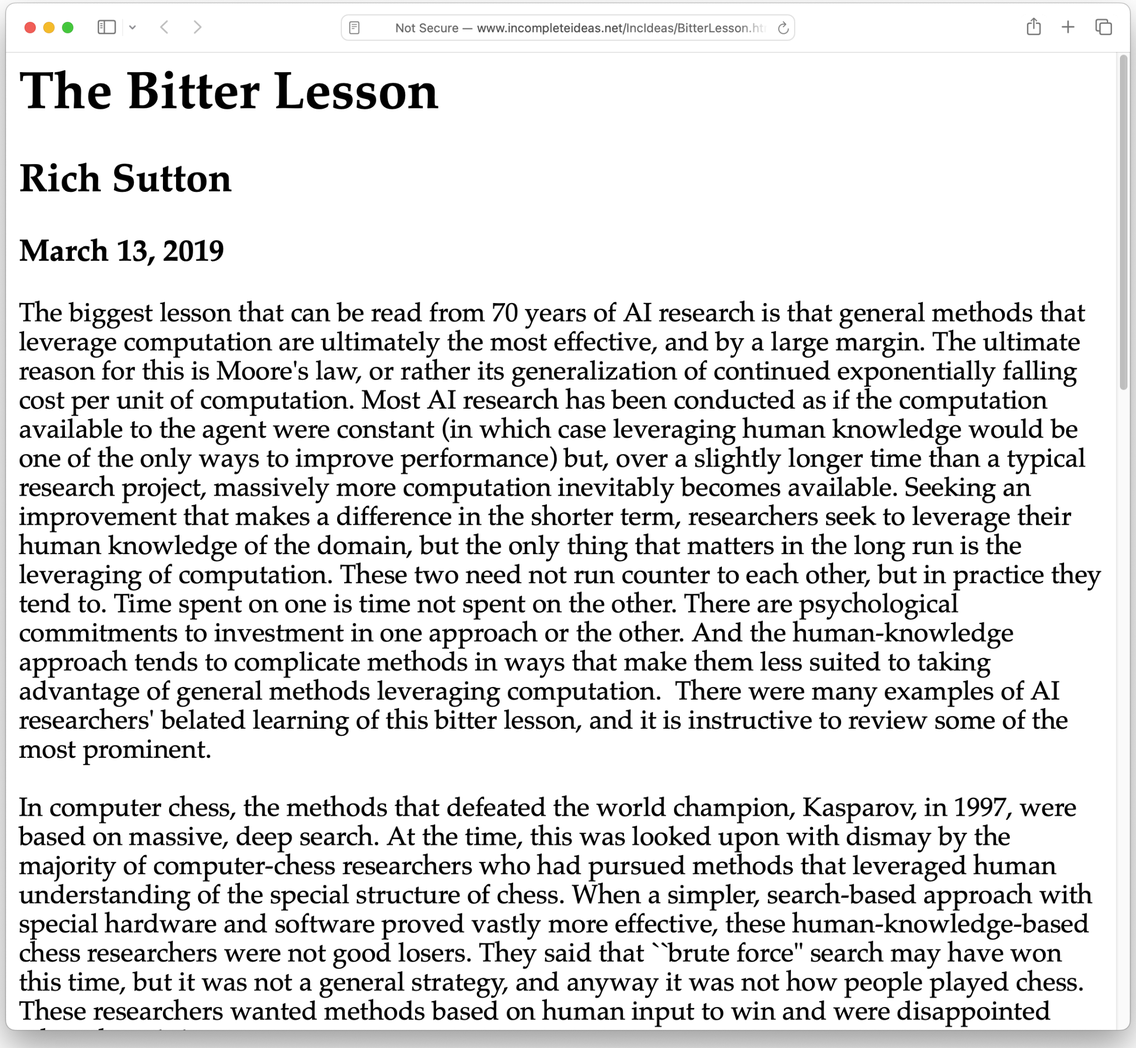
Deep Learning



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 Bitter Lesson

Rich Sutton

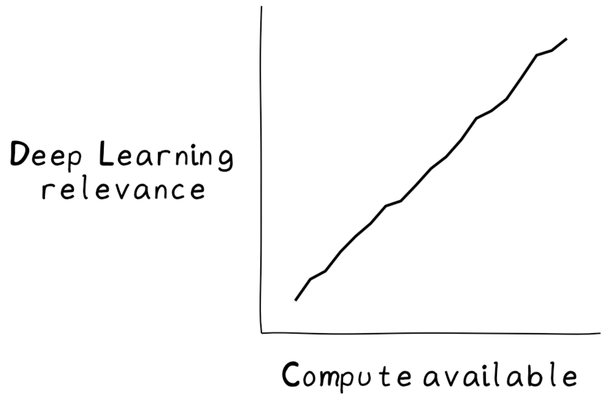
March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation. There were many examples of AI researchers' belated learning of this bitter lesson, and it is instructive to review some of the most prominent.

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

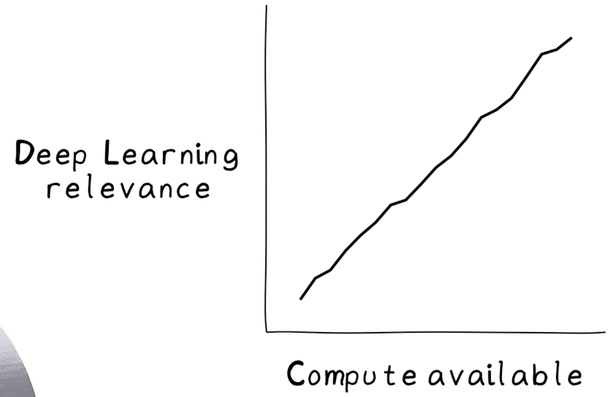
Deep Learning

- **Capabilities scale with compute**



Deep Learning

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



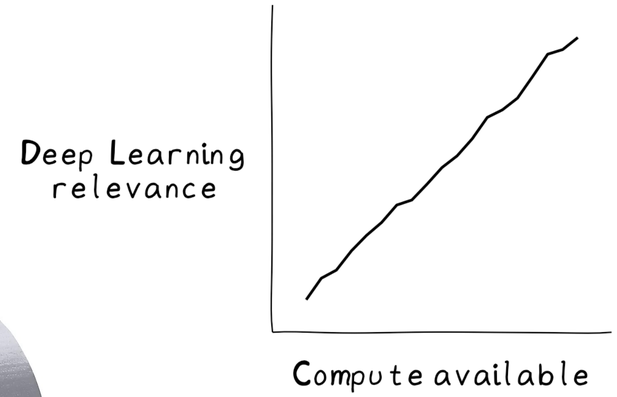
Deep Learning: General Purpose, Versatile

- 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.

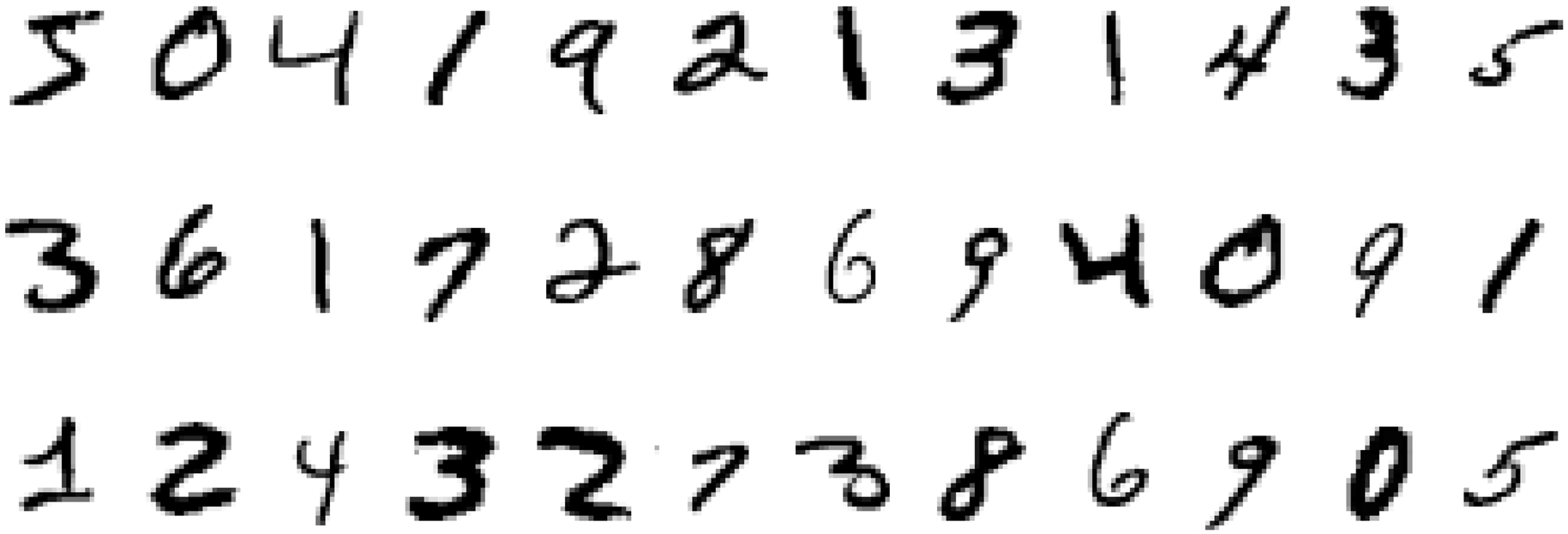
- Comes with many kinds of layers.
- Layers compose with a common, simple, interface.
- Create paths for data (marbles).
- Designing models is fun!
- Feeding models data is fun!

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.



- 60,000 training images
- 10,000 test images

>

Simple MNIST Image Classifier

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

<

Simple MNIST Image Classifier

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

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

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
```

Model Visualizations

```
> print(model)
```

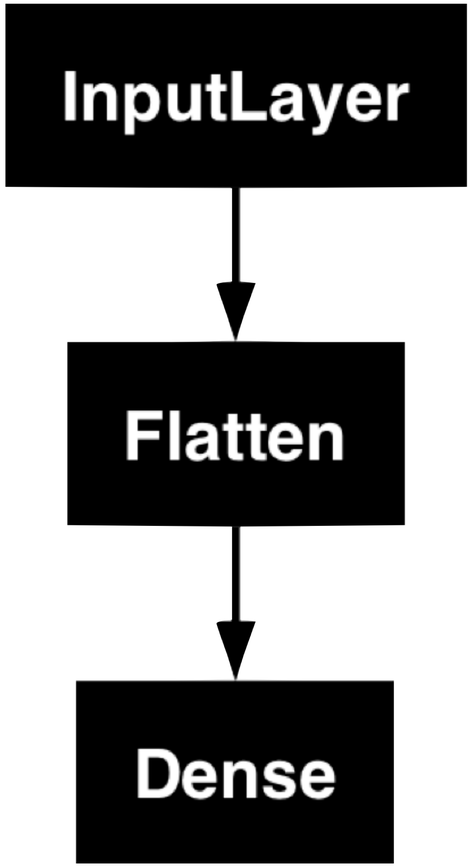
Model: "functional"

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)

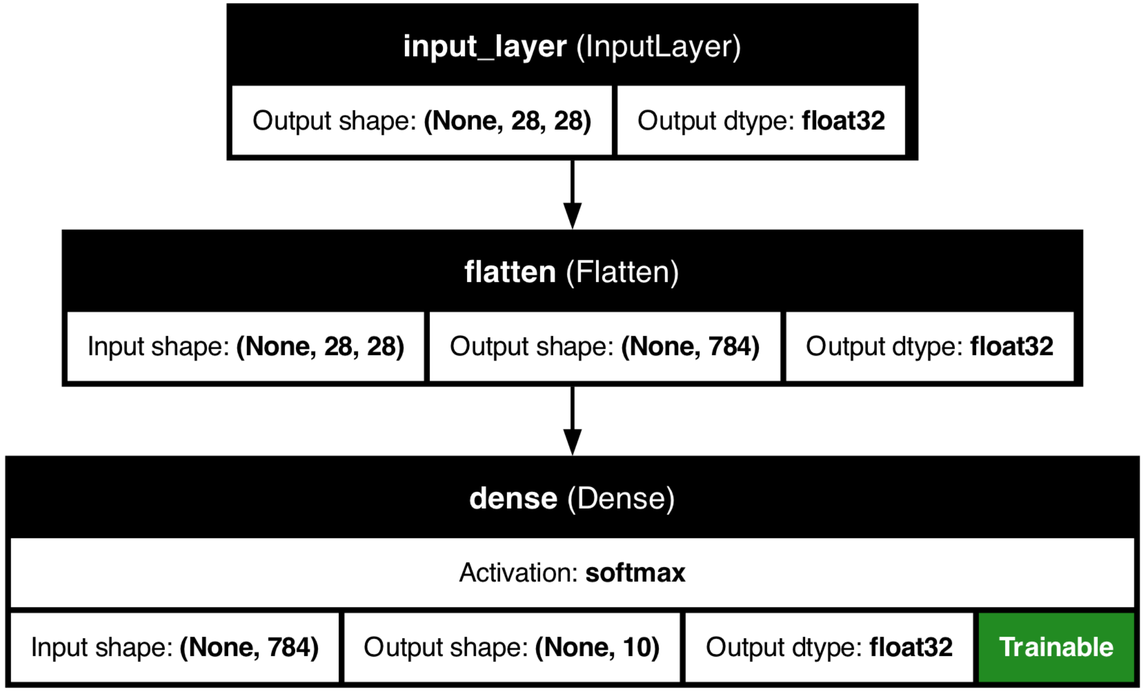
Model Visualizations

```
> plot(model)
```



Model Visualizations

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



High Level Training API

`compile()` `fit()`

High Level Training API

`compile()` `fit()`

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

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
)
```

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!

>

Simple MNIST Image Classifier

```
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
```

Trains to 97.5% accuracy!

<

Simple MNIST Image Classifier

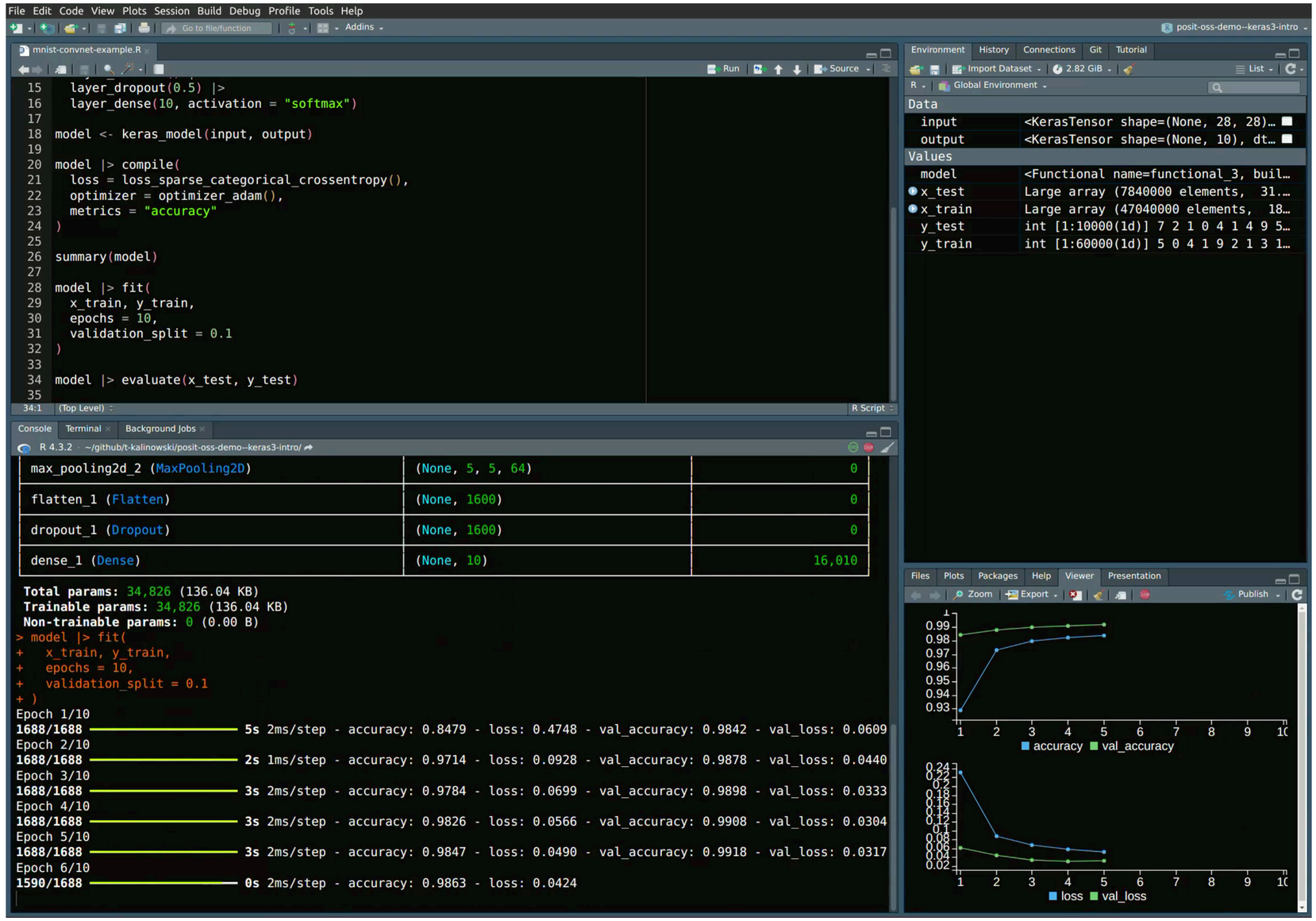
Trains to 98.7% accuracy!

```
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
```

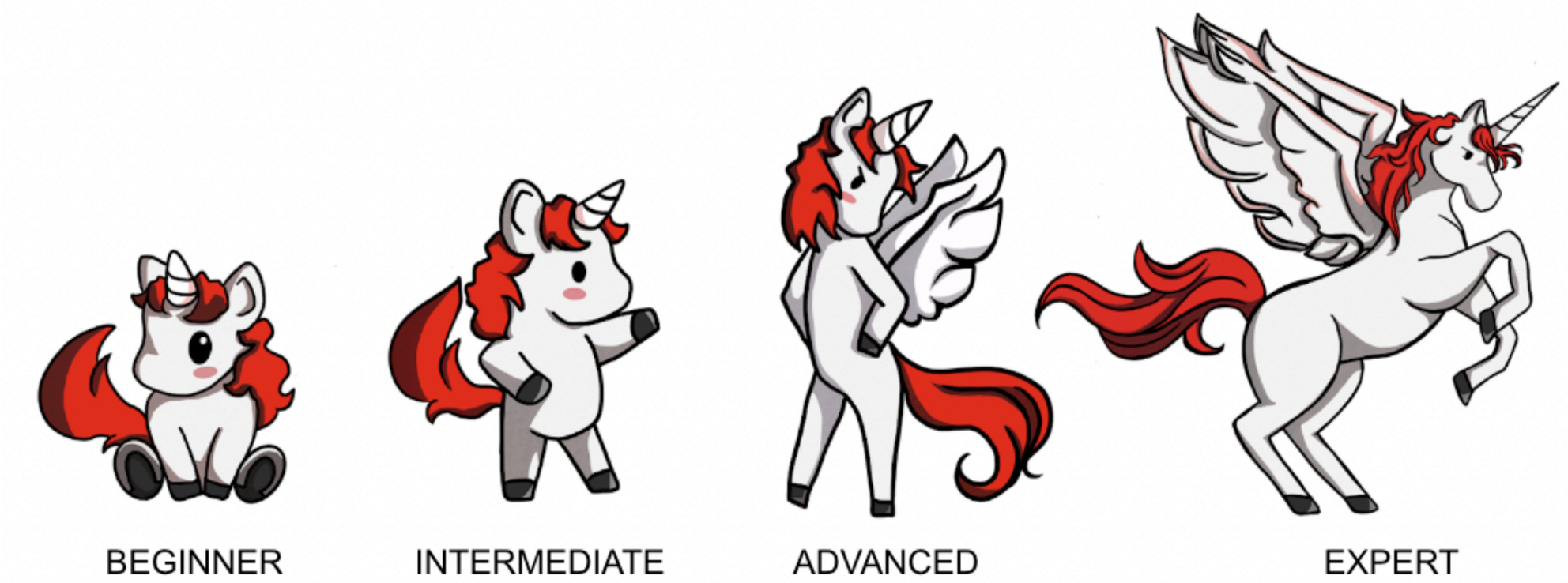
Simple MNIST Image Classifier

```
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!


File Edit Code View Plots Session Build Debug Profile Tools Help
mnist-convnet-example.R
layer_dropout(0.5) |>
layer_dense(10, activation = "softmax")
model <- keras_model(input, output)
model |> compile(
loss = loss_sparse_categorical_crossentropy(),
optimizer = optimizer_adam(),
metrics = "accuracy"
)
summary(model)
model |> fit(
x_train, y_train,
epochs = 10,
validation_split = 0.1
)
model |> evaluate(x_test, y_test)
Environment History Connections Git Tutorial
Data
input <KerasTensor shape=(None, 28, 28)…
output <KerasTensor shape=(None, 10), dt…
Values
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 Terminal Background Jobs
R 4.3.2 ~/github/t-kalinowski/posit-oss-demo--keras3-intro/
max_pooling2d_2 (MaxPooling2D) (None, 5, 5, 64) 0
flatten_1 (Flatten) (None, 1600) 0
dropout_1 (Dropout) (None, 1600) 0
dense_1 (Dense) (None, 10) 16,010
Total params: 34,826 (136.04 KB)
Trainable params: 34,826 (136.04 KB)
Non-trainable params: 0 (0.00 B)
> model |> fit(
+ x_train, y_train,
+ epochs = 10,
+ validation_split = 0.1
+)
Epoch 1/10
1688/1688 5s 2ms/step - accuracy: 0.8479 - loss: 0.4748 - val_accuracy: 0.9842 - val_loss: 0.0609
Epoch 2/10
1688/1688 2s 1ms/step - accuracy: 0.9714 - loss: 0.0928 - val_accuracy: 0.9878 - val_loss: 0.0440
Epoch 3/10
1688/1688 3s 2ms/step - accuracy: 0.9784 - loss: 0.0699 - val_accuracy: 0.9898 - val_loss: 0.0333
Epoch 4/10
1688/1688 3s 2ms/step - accuracy: 0.9826 - loss: 0.0566 - val_accuracy: 0.9908 - val_loss: 0.0304
Epoch 5/10
1688/1688 3s 2ms/step - accuracy: 0.9847 - loss: 0.0490 - val_accuracy: 0.9918 - val_loss: 0.0317
Epoch 6/10
1590/1688 0s 2ms/step - accuracy: 0.9863 - loss: 0.0424
Files Plots Packages Help Viewer Presentation
accuracy val_accuracy
loss val_loss

Keras: Next Steps After MNIST



https://keras.io/guides/keras_nlp/getting_started/

>

Keras: Next Steps After MNIST

Progressive Disclosure of Complexity

<

>

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_...
```

<

Defining Custom Layers
`keras_model()`

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

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

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_...
```

Defining Custom Layers

`Layer()`

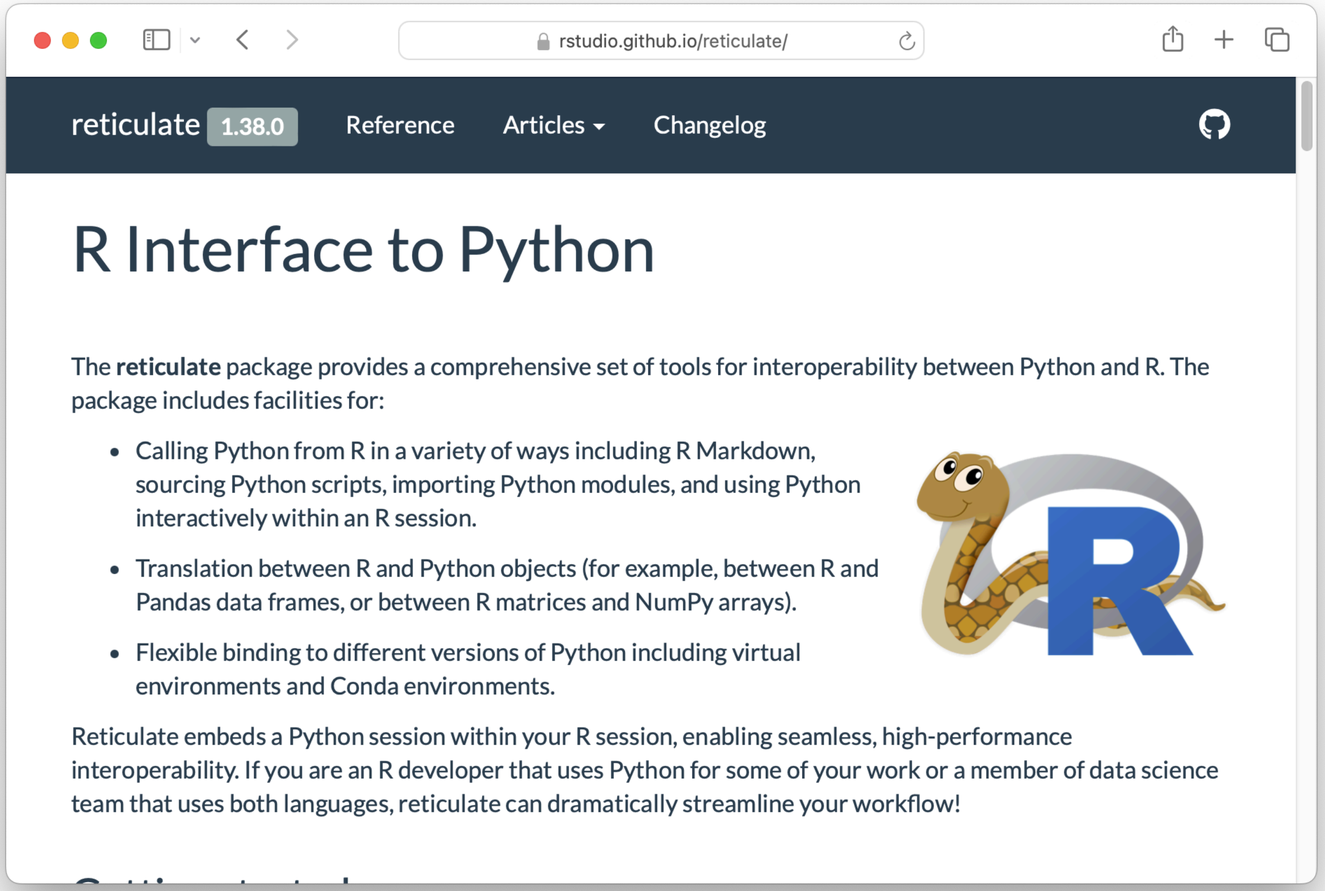
```
# Subclass base Layer class
Layer(...)

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

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

Subclassing API

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

```
Layer()                    LearningRateSchedule()
Callback()                 Constraint()
Metric()                   Loss()
Model()
```

Reticulate: Access everything in Python from R



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.

> Keras 3: What's New

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

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.

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")
```

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)

>

Saving & Loading
New backend-agnostic file format.

Convert models between different backends

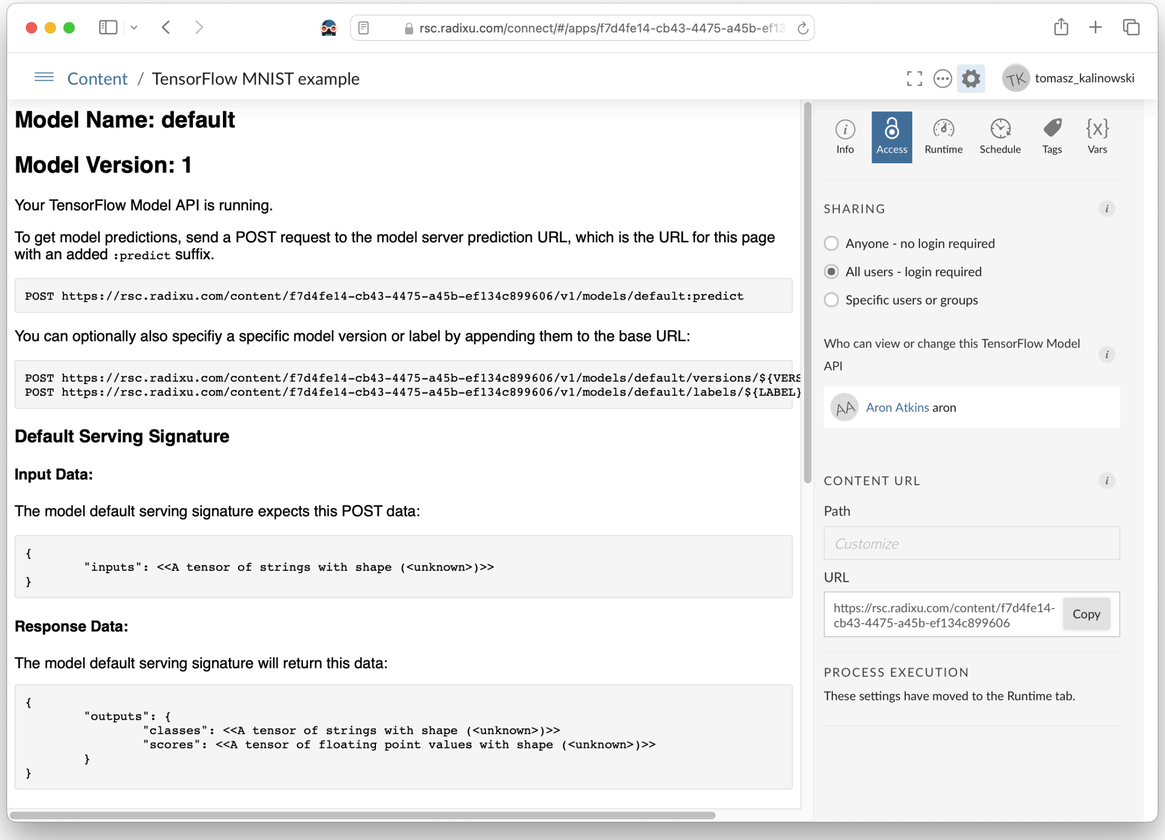
```r
library(keras3)
use_backend("jax")
model <- keras_model(...)
model |> fit(...)
model |> save_model("my_model.keras")

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

<

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



Keras 3 Multi-Backend Support: Ops

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

```
op_sum()                    op_roll()                   op_identity()
op_abs()                    op_ctc_decode()             op_imag()
op_slice_update()           op_ravel()                  op_square()
op_average_pool()           op_normalize()              op_elu()
op_isfinite()               op_qr()                     op_add()
op_isnan()                  op_softmax()                op_binary_crossentropy()
op_relu6()                  op_one_hot()                op_lu_factor()
op_maximum()                op_softplus()               op_gelu()
op_eig()                    op_zeros()                  op_custom_gradient()
op_arctan()                 op_argsort()                op_diagonal()
...                         ...                         ...
```

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.

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)
```

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!

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.

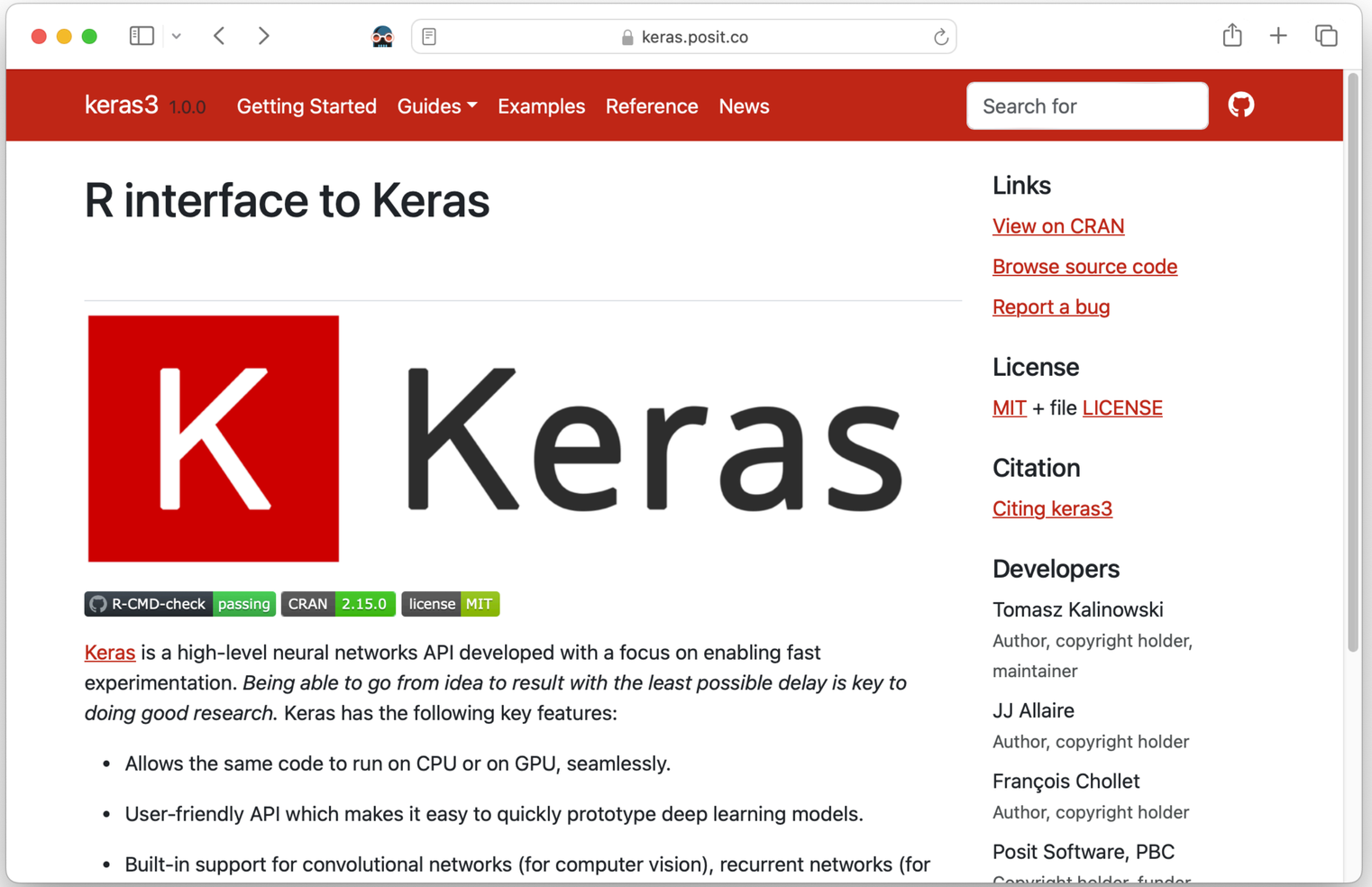
- All functions are pure (easy to reason about)

> 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

<

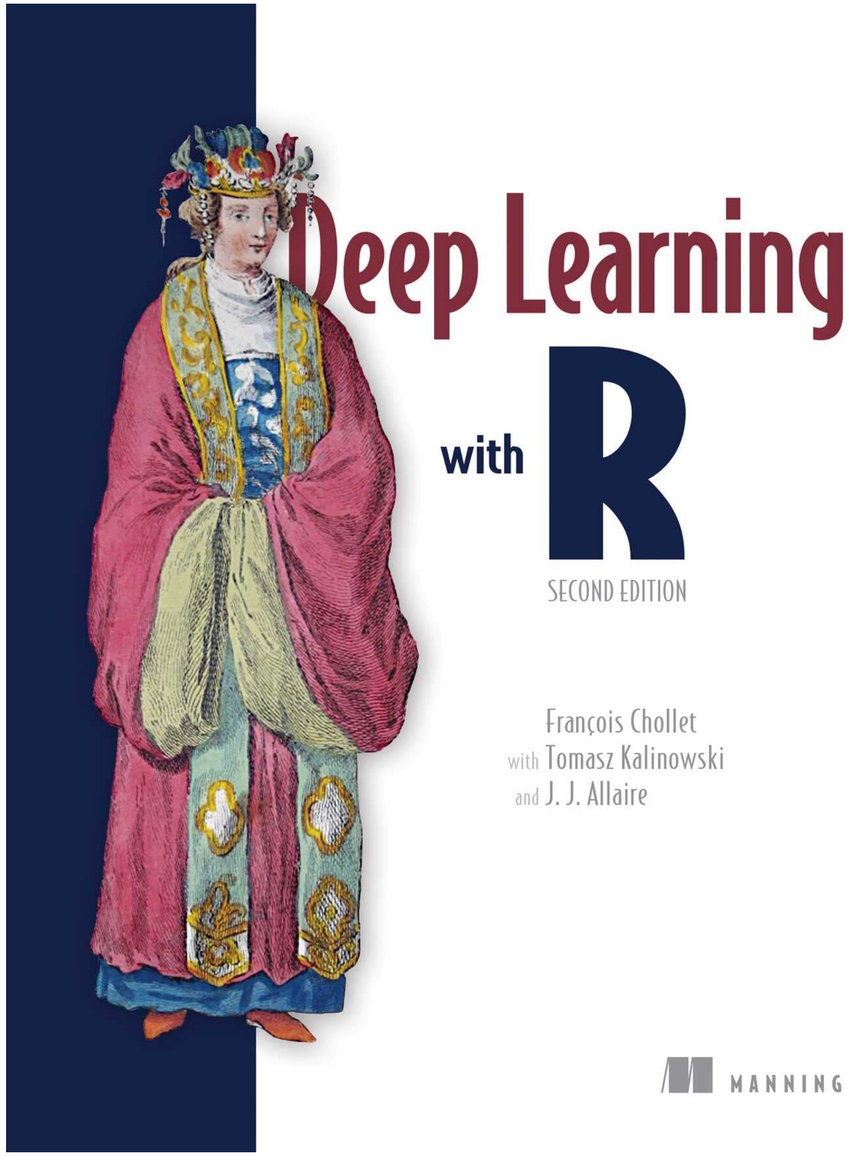
Keras 3 Package website: https://keras3.posit.co



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



posit
conf(2024)

Thank you.