

# Learning Systems 2018: Lecture 3 – Keras Overview



Ian Foster and Rick Stevens  
Argonne National Laboratory  
The University of Chicago

Crescat scientia; vita excolatur

Many of these slides are adapted from  
talks from François Chollet and others

imperative  
symbolic

theano

Caffe

Microsoft  
CNTK



++ Caffe2



before

2012

2013

2014

2015

2016

2017

imperative  
symbolic

theano

Caffe

Microsoft  
CNTK



↑  
Caffe2

before

2012

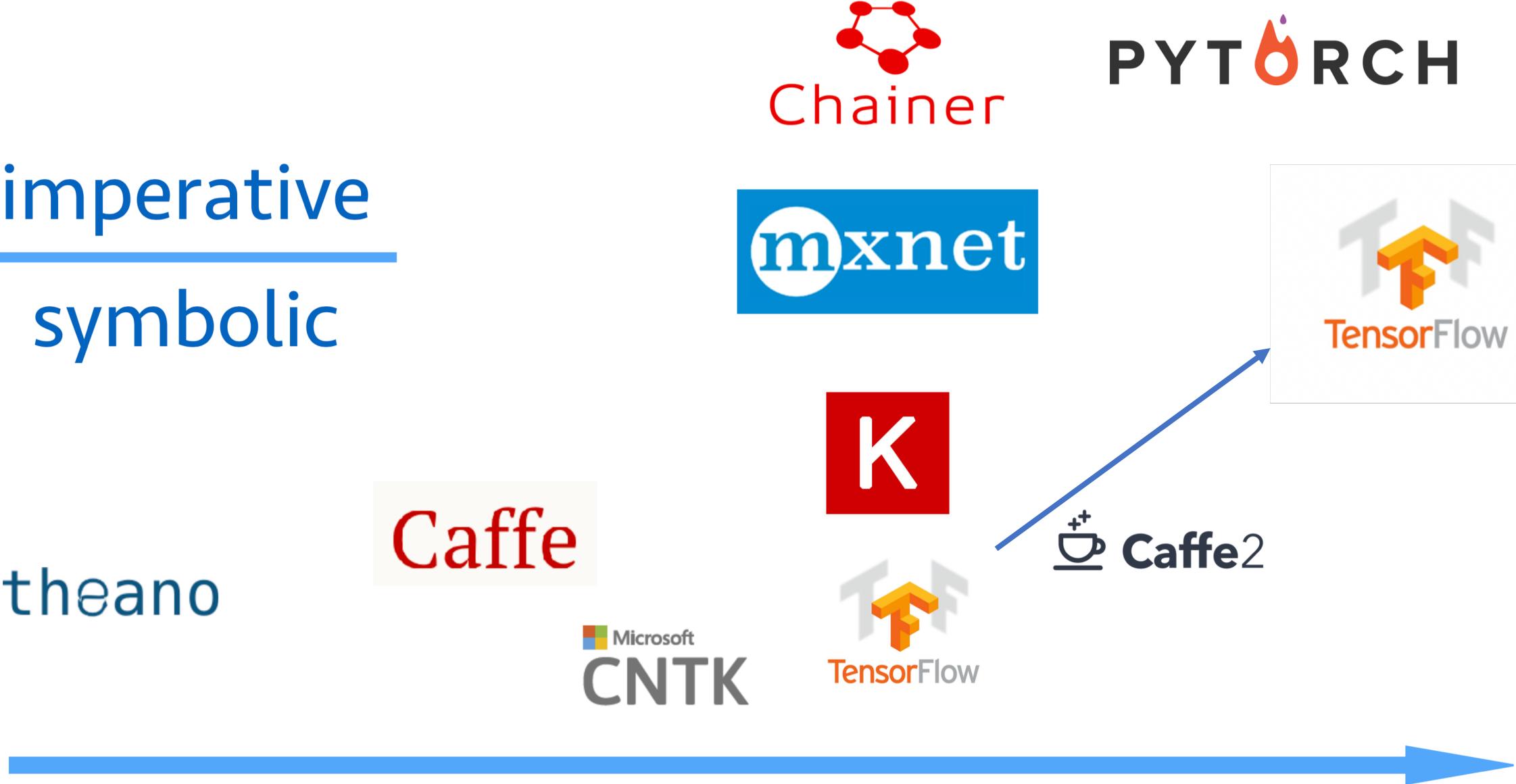
2013

2014

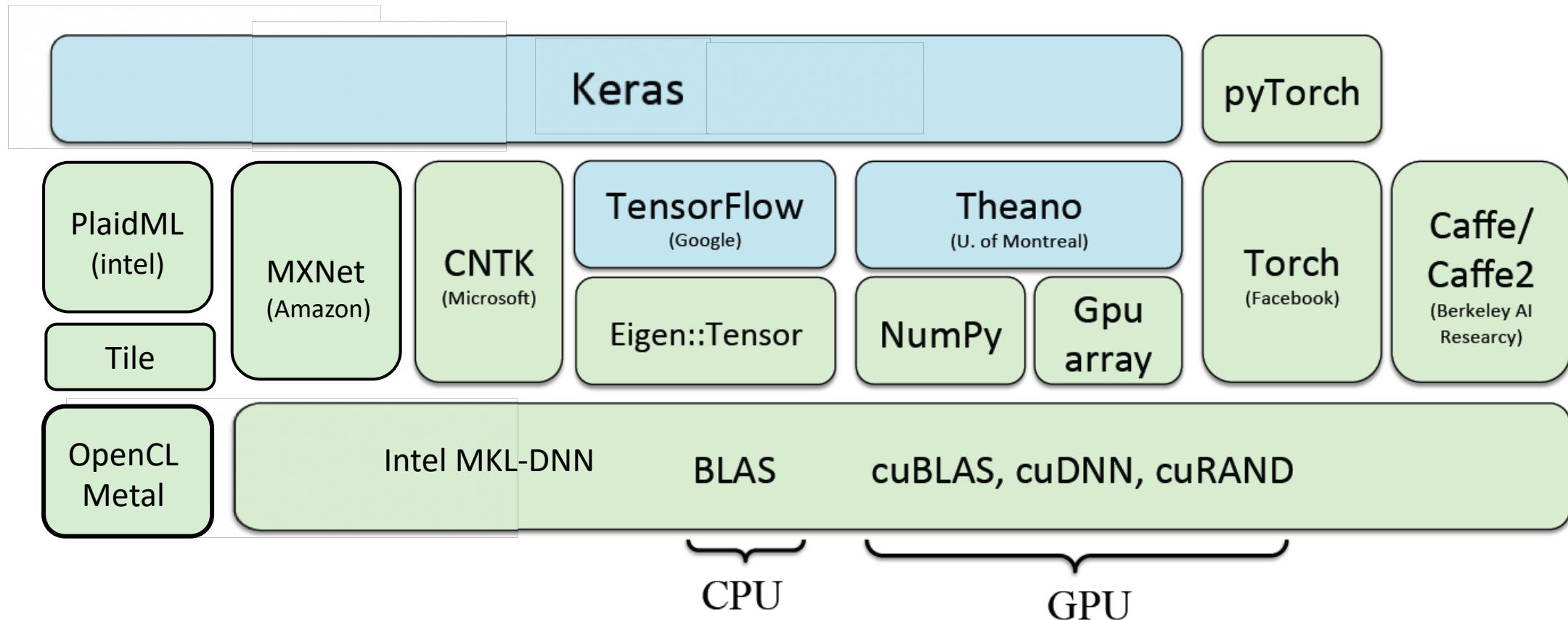
2015

2016

2017



# Deep Learning Software Stacks



# Open Source Framework Comparison

	Languages	Tutorials and training materials	CNN modeling capability	RNN modeling capability	Architecture: easy-to-use and modular front end	Speed	Multiple GPU support	Keras compatible
Theano	Python, C++	++	++	++	+	++	+	+
Tensor-Flow	Python	+++	+++	++	+++	++	++	+
Torch	Lua, Python (new)	+	+++	++	++	+++	++	
Caffe	C++	+	++		+	+	+	
MXNet	R, Python, Julia, Scala	++	++	+	++	++	+++	+
Neon	Python	+	++	+	+	++	+	
CNTK	C++	+	+	+++	+	++	+	+

# Introduction to Keras

- What is Keras?
  - What are the target use cases
  - TensorFlow Integration
  - Backends for other frameworks
- How to use Keras
  - 3 API Styles
- Distributed, Multi-GPU and TPU Training
- Eager execution (define-by-run, aka dynamic graphs)

# Keras: an API for specifying and training differentiable programs

Keras API

TensorFlow / CNTK / MXNet / Theano / ...

GPU

CPU

TPU

# Keras is the “Official” high-level API of TensorFlow (but not the only one)

- tensorflow.keras (tf.keras) module
- Part of core TensorFlow since v1.4
- Full Keras API
- Better optimized for TF
- Better integration with TF-specific features
  - Estimator API
  - Eager execution
  - etc.

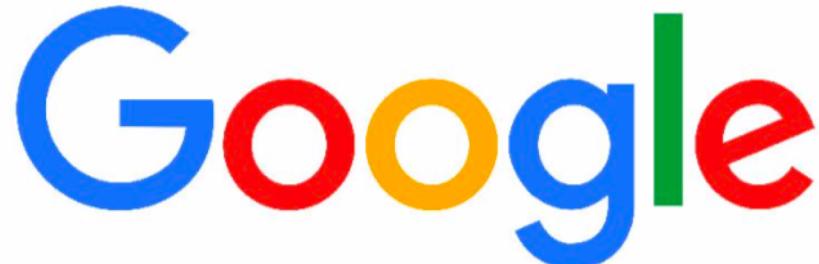
**tf.keras**

**TensorFlow**

**GPU**    **CPU**    **TPU**

# Who makes Keras? Contributors and Backers

 633 contributors

 Google



# How is Keras different?

- A focus on user experience
- Large adoption in industry and research community > 250,000 devs
- Multi-backend, multi-platform
- Easy path to productization of models
- Easy learning curve
- Well documented
- Good examples library
- Good pre-trained model library

# Adoption by Industry

**NETFLIX**

**UBER**

**Google**

 **instacart**

 **HUAWEI**

 **nVIDIA.**

 **Square**

 **Expedia®**

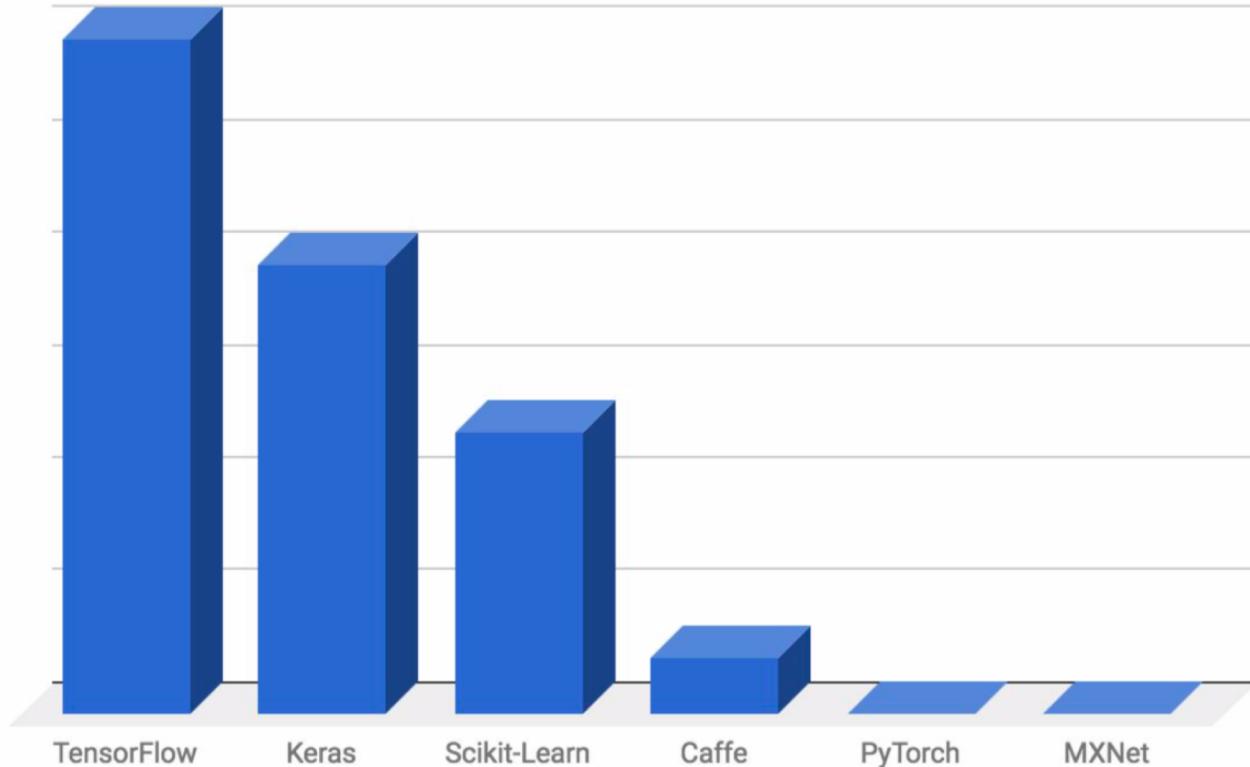
 **Zocdoc**

 **yelp.**

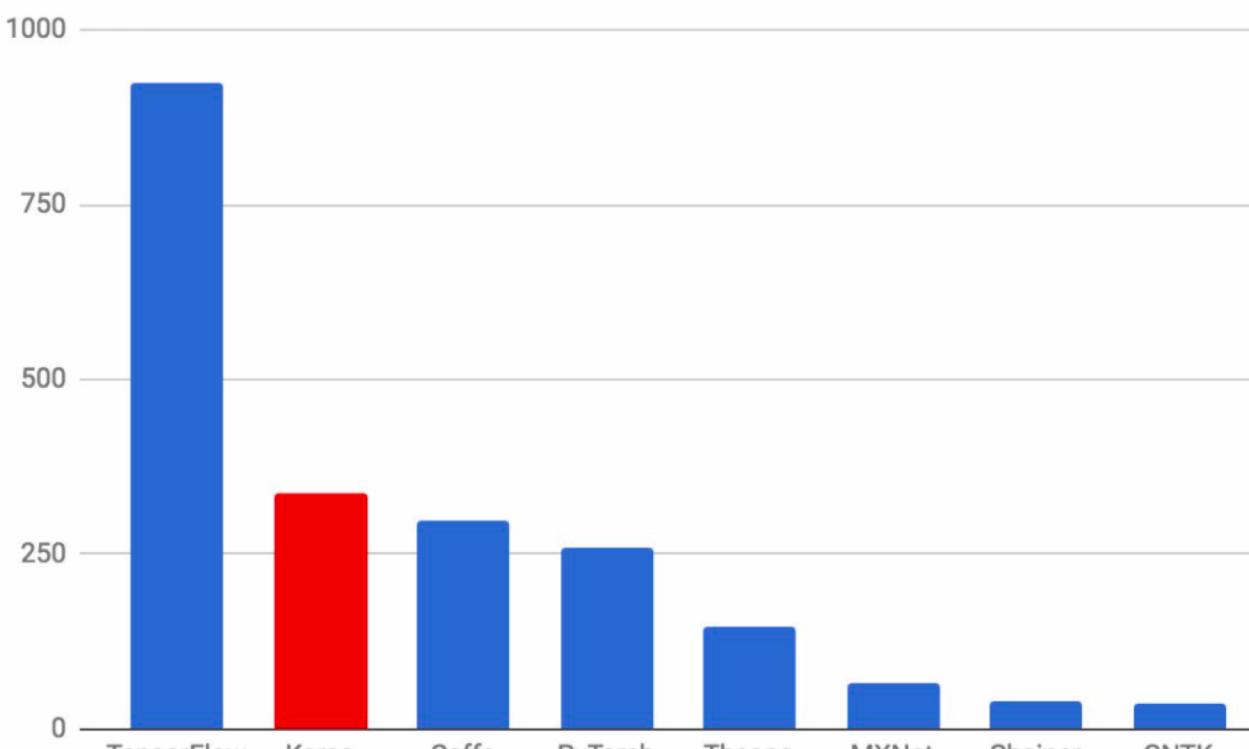
**etc...**

# High-Adoption in Startup Land

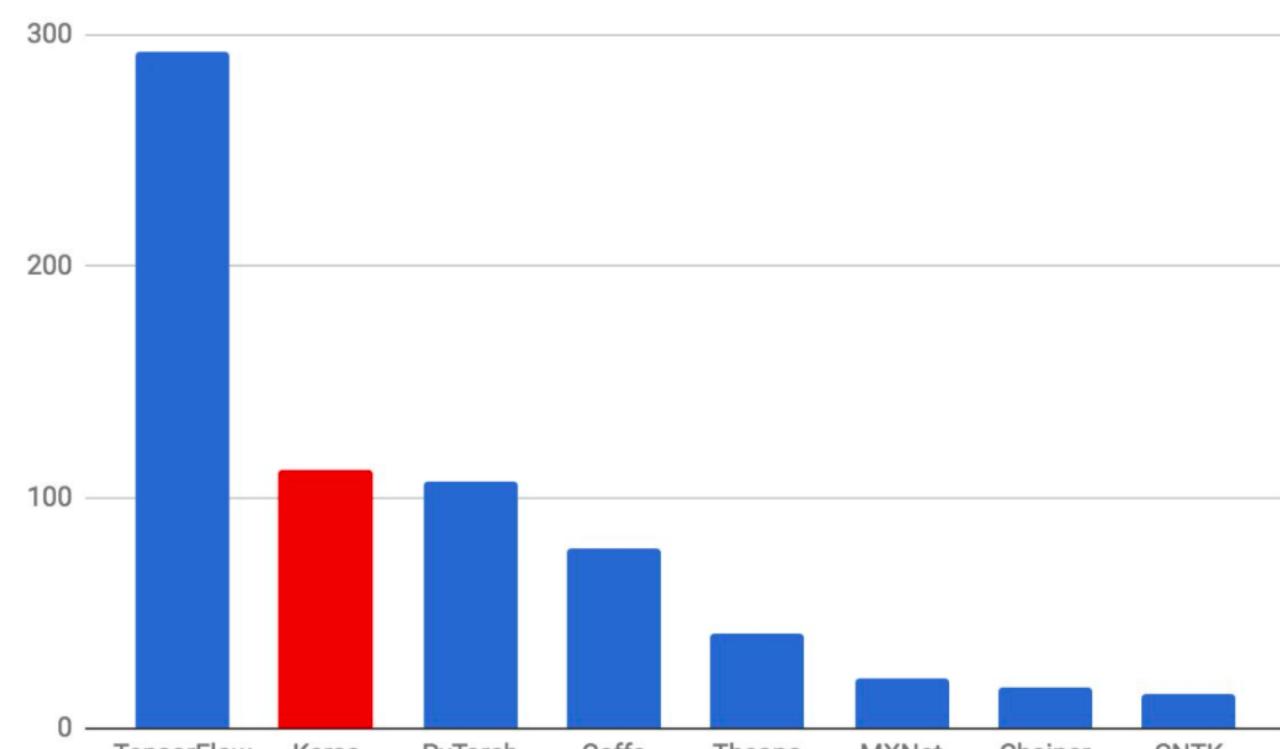
Hacker News jobs board mentions - out of 964 job postings



# Adoption in Research Community



arXiv mentions as of 2018/03/07 (past 3 months)



arXiv mentions as of 2018/03/07 (past 1 month)

# User Experience is the Primary Goal

**Keras is an API designed for human beings, not machines.** Keras follows best practices for reducing cognitive load: it offers consistent & simple APIs, it minimizes the number of user actions required for common use cases, and it provides clear and actionable feedback upon user error.

**This makes Keras easy to learn and easy to use.** As a Keras user, you are more productive, allowing you to try more ideas than your competition, faster -- which in turn helps you win machine learning competitions.

**This ease of use does not come at the cost of reduced flexibility:** because Keras integrates with lower-level deep learning languages (in particular TensorFlow), it enables you to implement anything you could have built in the base language. In particular, as `tf.keras`, the Keras API integrates seamlessly with your TensorFlow workflows.

# Keras is multi-backend, multi-platform

- Develop in Python or R
  - Unix, OSX and Windows
- Run the same code (nearly true) on
  - TensorFlow
  - Theano
  - MXNet
  - CNTK
  - PlaidML (☺) (Vertex.ai → Intel Machine Learning Group now)
- Backends run on nearly everything
  - X86/ARM CPU, NVIDIA GPUs, AMD GPUs, TPU, (custom accelerators...)



## Keras

---

### R interface to Keras

**Keras** is a high-level neural networks API developed with a focus on enabling fast experimentation. *Being able to go from idea to result with the least possible delay is key to doing good research.* Keras has the following key features:

- Allows the same code to run on CPU or on GPU, seamlessly.
- User-friendly API which makes it easy to quickly prototype deep learning models.
- Built-in support for convolutional networks (for computer vision), recurrent networks (for sequence processing), and any combination of both.
- Supports arbitrary network architectures: multi-input or multi-output models, layer sharing, model sharing, etc. This means that Keras is appropriate for building essentially any deep learning model, from a memory network to a neural Turing machine.
- Is capable of running on top of multiple back-ends including [TensorFlow](#), [CNTK](#), or [Theano](#).

For additional details on why you might consider using Keras for your deep learning projects, see the [Why Use Keras?](#) article.

This website provides documentation for the R interface to Keras. See the main Keras website at <https://keras.io> for additional information on the project.

# Large number of options for moving models to production

- TF-serving
- In-Browser with GPU acceleration (WebKeras, Keras.js, WebDNN, ...)
- Android (TF, TF Lite), iPhone (native CoreML support)
- Raspberry Pi
- JVM
- via ONNX to other systems  
<https://github.com/onnx/onnxmлltools>

Ex, AR apps with Keras + TF + CoreML + ARKit

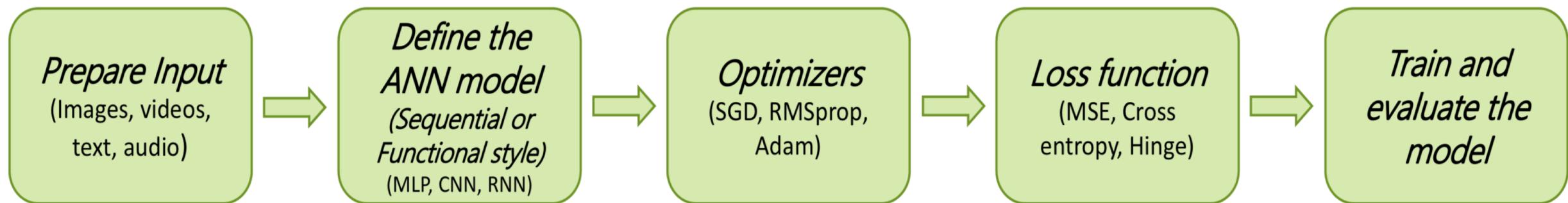
How to use  
Keras:  
An introduction

# Implementing a neural network in Keras

- Five major steps

- Preparing the input and specify the input dimension (size)
- Define the model architecture and build the computational graph
- Specify the optimizer and configure the learning process
- Specify the Inputs, Outputs of the computational graph (model) and the Loss function
- Train and test the model on the dataset

Note: Gradient calculations are taken care by Auto – Differentiation and parameter updates are done automatically in the backend



# Three API styles

- The Sequential Model
  - Super simple
  - Only for single-input, single-output, sequential layer stacks
  - Good for 70% of use cases
- The functional API
  - Like Lego bricks
  - Multi-input, multi-output, arbitrary static graph topologies
  - Good for 95% of use cases
- Model subclassing (for OO folks that want to drop down)
  - Maximum flexibility
  - Large potential error surface

# The Sequential API

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

# The Sequential API

You need data in X and Y for training

Typically X\_train, Y\_train and X\_test, Y\_test

```
import keras
from keras import layers

model = keras.Sequential()
model.add(layers.Dense(20, activation='relu', input_shape=(10,)))
model.add(layers.Dense(20, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))

model.fit(x, y, epochs=10, batch_size=32)
```

Compile call, and choice of Optimizer and Loss are omitted here

# The functional API

```
import keras
from keras import layers

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)
model.fit(x, y, epochs=10, batch_size=32)
```

# Model subclassing

```
import keras
from keras import layers

class MyModel(keras.Model):

    def __init__(self):
        super(MyModel, self).__init__()
        self.dense1 = layers.Dense(20, activation='relu')
        self.dense2 = layers.Dense(20, activation='relu')
        self.dense3 = layers.Dense(10, activation='softmax')

    def call(self, inputs):
        x = self.dense1(x)
        x = self.dense2(x)
        return self.dense3(x)

model = MyModel()
model.fit(x, y, epochs=10, batch_size=32)
```

# High-Level Keras Elements

- Models
- Layers
- Preprocessing Data
- Loss functions
- Metrics (for reporting)
- Optimizers
- Activations
- Callbacks
- Datasets
- Applications

## Layers

- About Keras layers
- Core Layers
- Convolutional Layers
- Pooling Layers
- Locally-connected Layers
- Recurrent Layers
- Embedding Layers
- Merge Layers
- Advanced Activations Layers
- Normalization Layers
- Noise layers
- Layer wrappers
- Writing your own Keras layers

## Losses

Usage of loss functions

Available loss functions

mean\_squared\_error

mean\_absolute\_error

mean\_absolute\_percentage\_error

mean\_squared\_logarithmic\_error

squared\_hinge

hinge

categorical\_hinge

logcosh

categorical\_crossentropy

sparse\_categorical\_crossentropy

binary\_crossentropy

kullback\_leibler\_divergence

poisson

cosine\_proximity

## Metrics

Usage of metrics

Arguments

Returns

Available metrics

binary\_accuracy

categorical\_accuracy

sparse\_categorical\_accuracy

top\_k\_categorical\_accuracy

sparse\_top\_k\_categorical\_accuracy

Custom metrics

## Utils

CustomObjectScope

HDF5Matrix

Sequence

to\_categorical

normalize

get\_file

print\_summary

plot\_model

multi\_gpu\_model

[Preprocessing](#)

[Sequence Preprocessing](#)

[Text Preprocessing](#)

[Image Preprocessing](#)

## Optimizers

[Usage of optimizers](#)

[Parameters common to all Keras optimizers](#)

[SGD](#)

[RMSprop](#)

[Adagrad](#)

[Adadelta](#)

[Adam](#)

[Adamax](#)

[Nadam](#)

## Activations

[Usage of activations](#)

[Available activations](#)

[softmax](#)

[elu](#)

[selu](#)

[softplus](#)

[softsign](#)

[relu](#)

[tanh](#)

[sigmoid](#)

[hard\\_sigmoid](#)

[exponential](#)

[linear](#)

[On "Advanced Activations"](#)

## Callbacks

[Usage of callbacks](#)

[Callback](#)

[BaseLogger](#)

[TerminateOnNaN](#)

[ProgbarLogger](#)

[History](#)

[ModelCheckpoint](#)

[EarlyStopping](#)

[RemoteMonitor](#)

[LearningRateScheduler](#)

[TensorBoard](#)

[ReduceLROnPlateau](#)

[CSVLogger](#)

[LambdaCallback](#)

[Create a callback](#)

[Example: recording loss history](#)

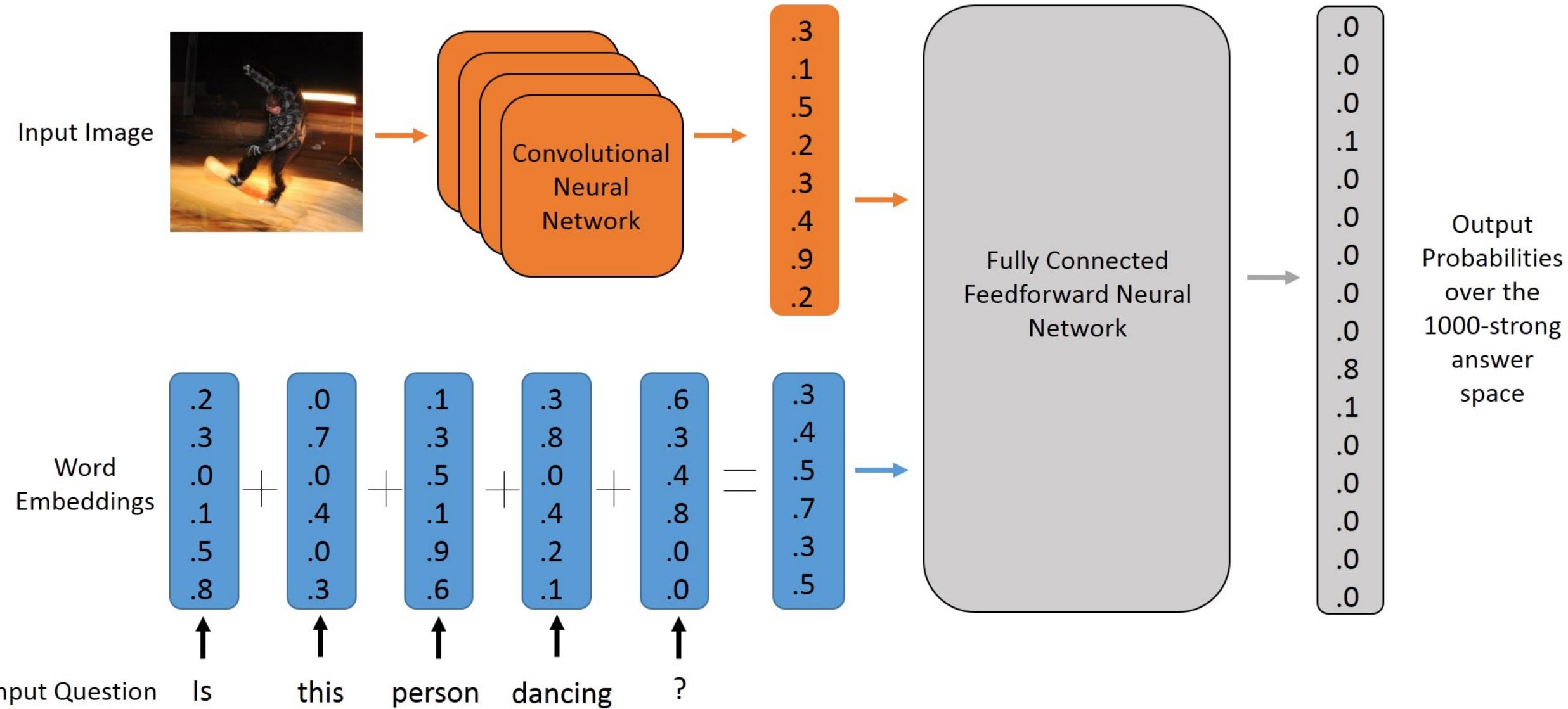
[Example: model checkpoints](#)

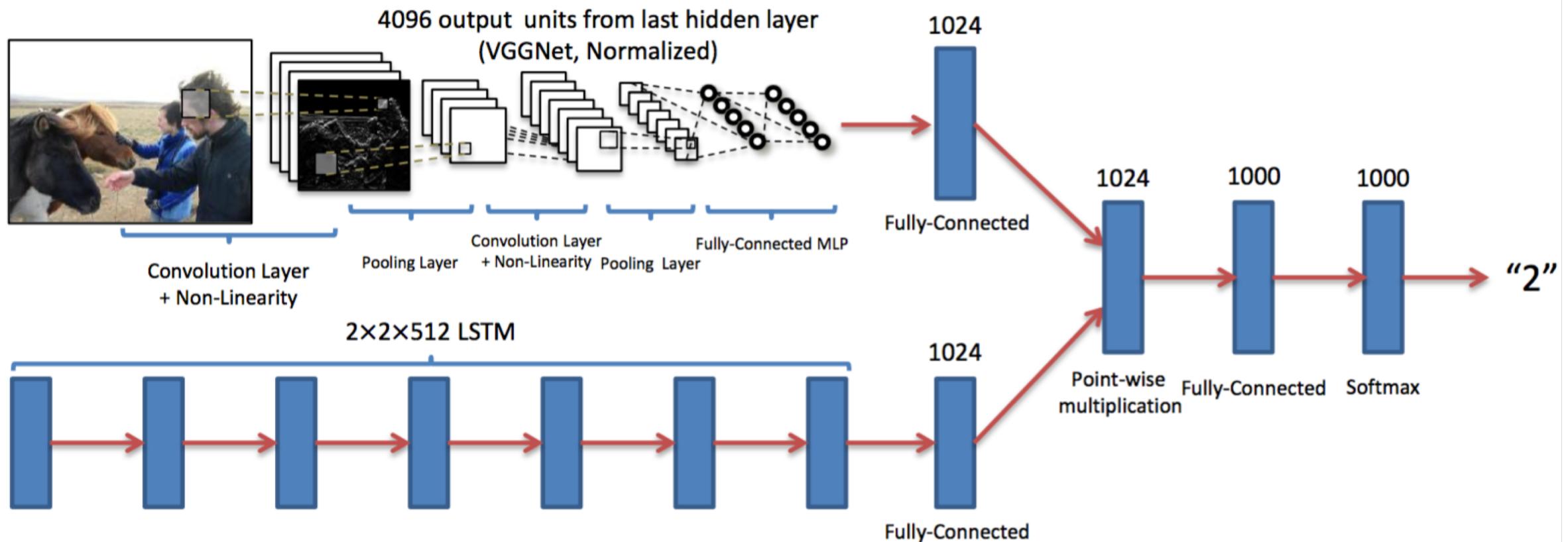
Datasets
Datasets
CIFAR10 small image classification
CIFAR100 small image classification
IMDB Movie reviews sentiment classification
Reuters newswire topics classification
MNIST database of handwritten digits
Fashion-MNIST database of fashion articles
Boston housing price regression dataset

Applications
Available models
Usage examples for image classification models
Documentation for individual models
Xception
VGG16
VGG19
ResNet50
InceptionV3
InceptionResNetV2
MobileNet
DenseNet
NASNet
MobileNetV2

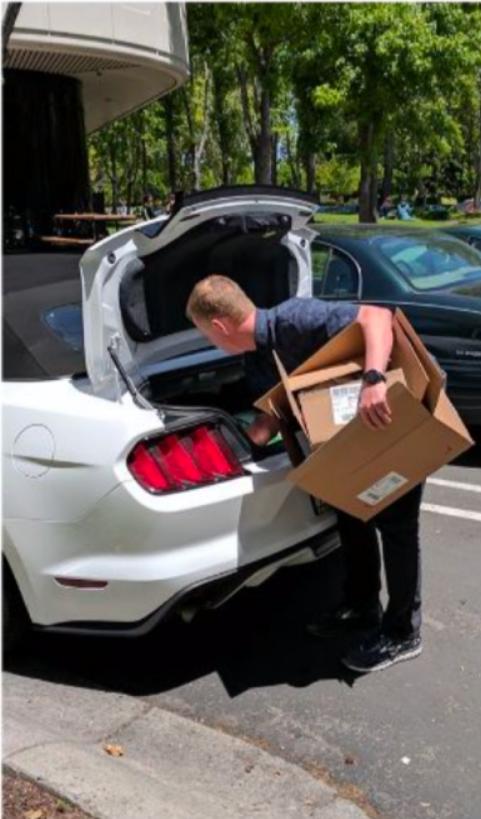
Initializers
Usage of initializers
Available initializers
Initializer
Zeros
Ones
Constant
RandomNormal
RandomUniform
TruncatedNormal
VarianceScaling
Orthogonal
Identity
lecun_uniform
glorot_normal
glorot_uniform
he_normal
lecun_normal
he_uniform
Using custom initializers

Example: building a  
video captioning model



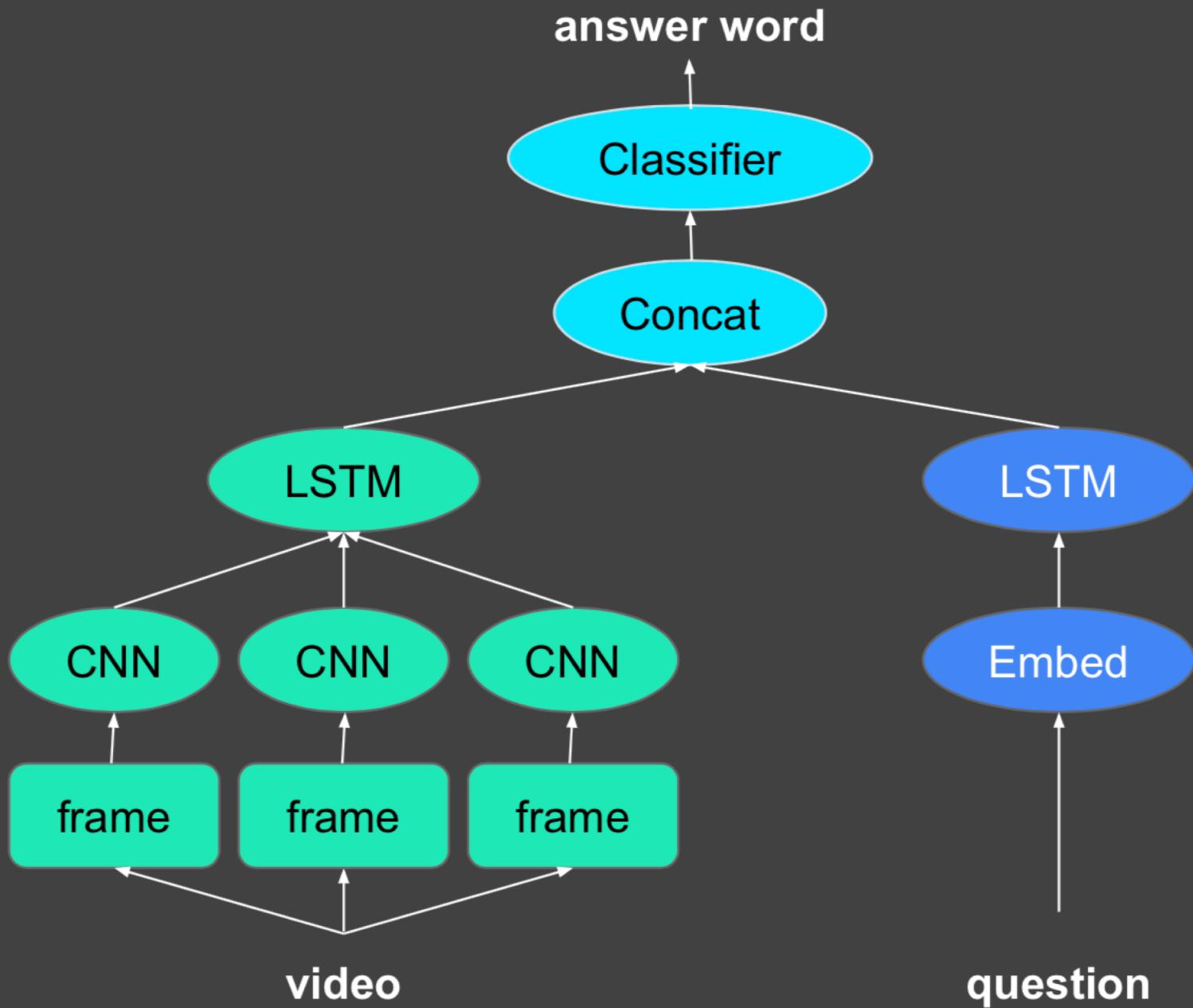


# Toy video-QA problem

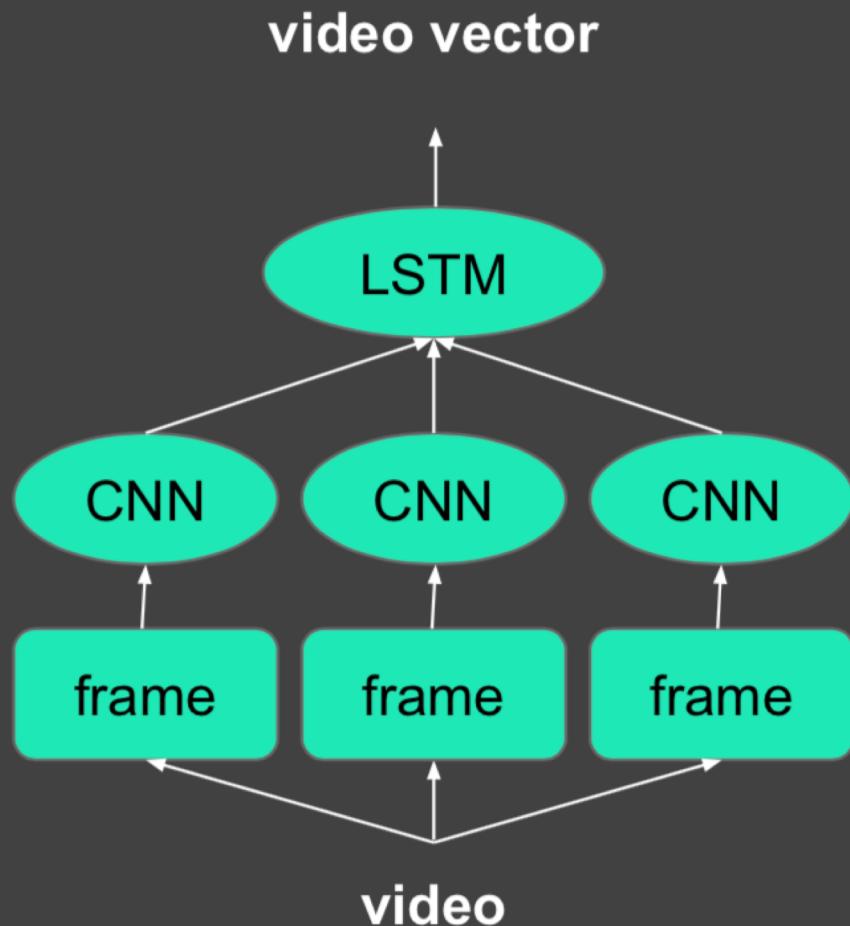


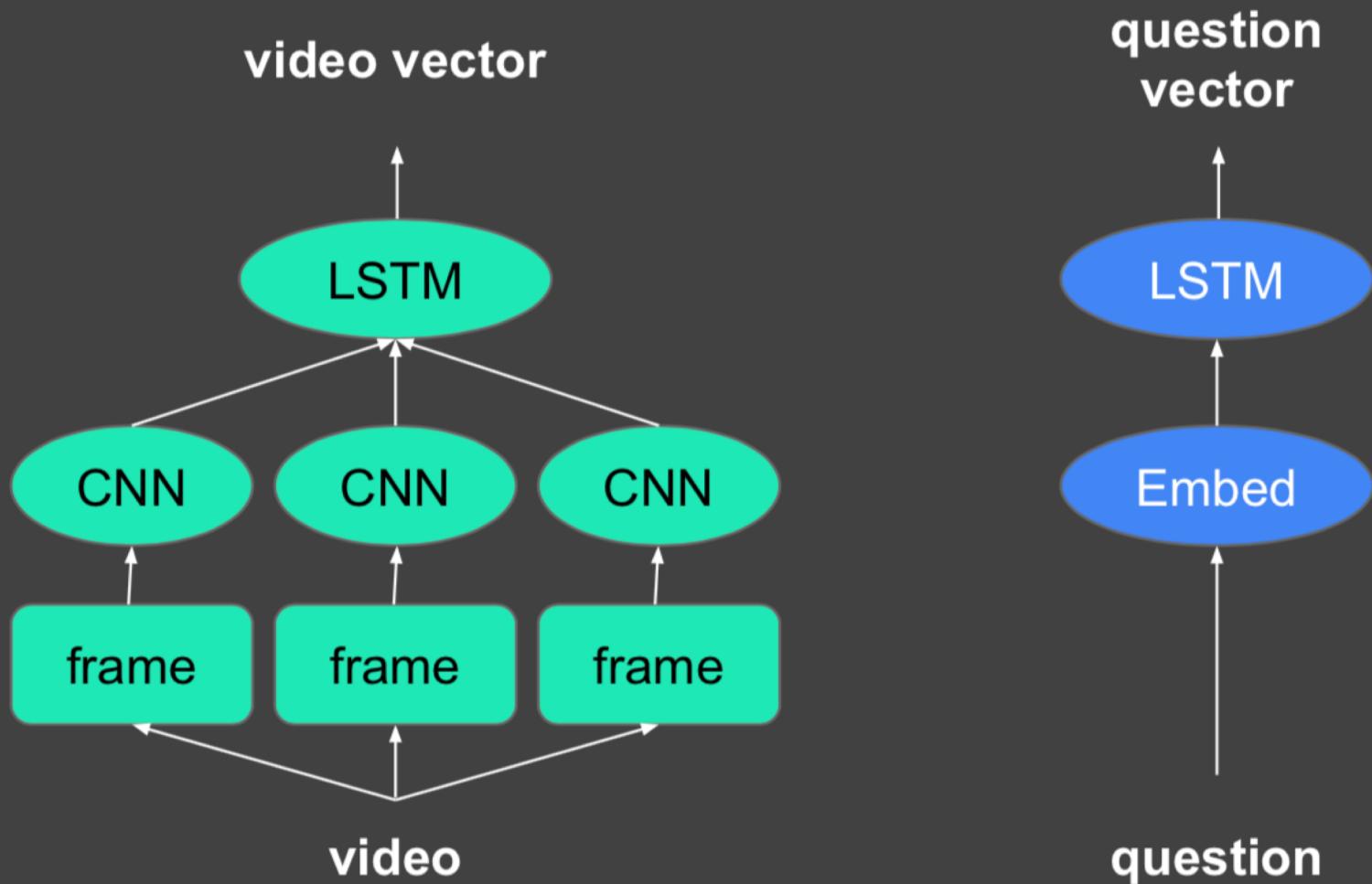
> "**What is the man doing?**"  
> packing

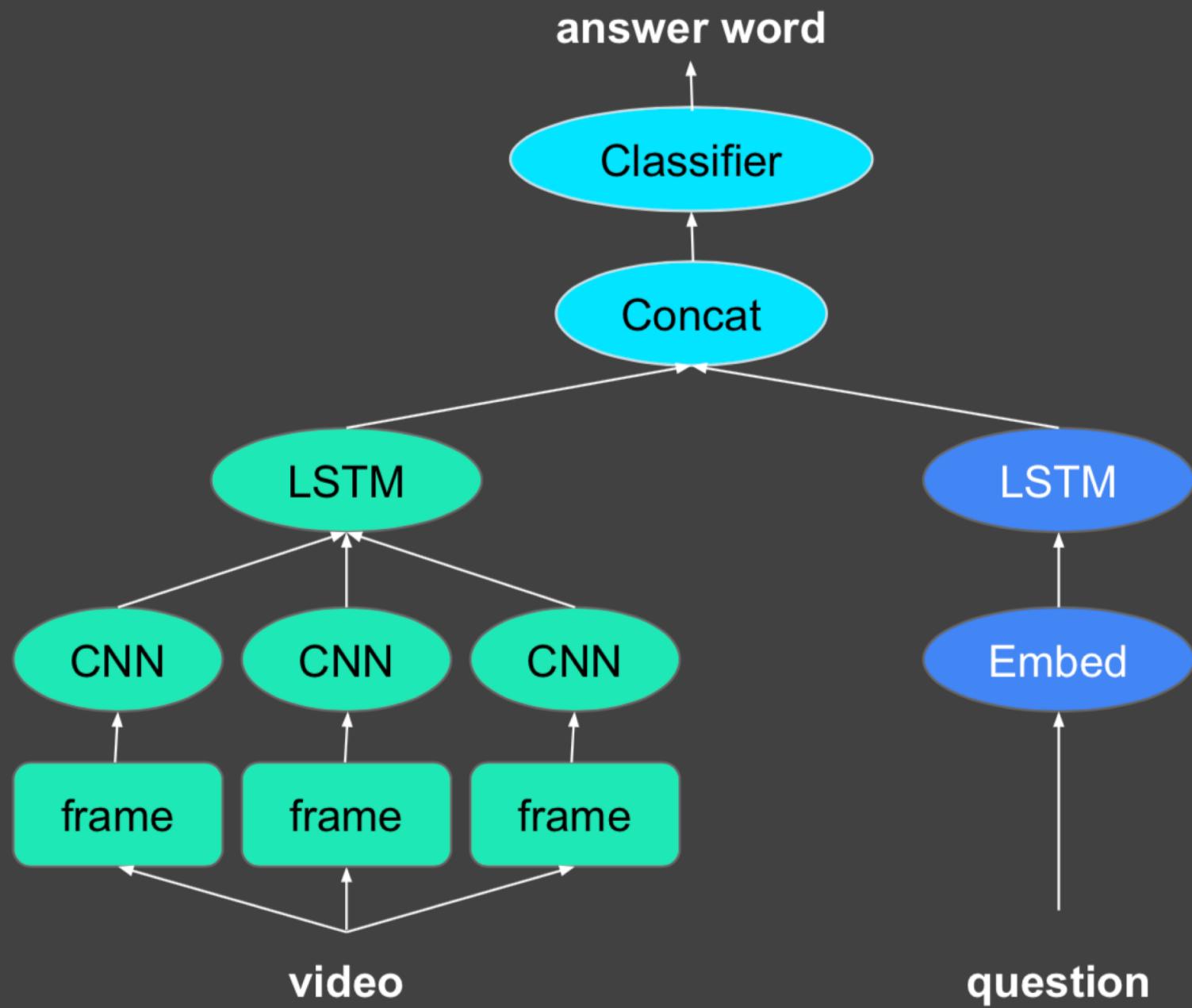
> "**What color is his shirt?**"  
> blue

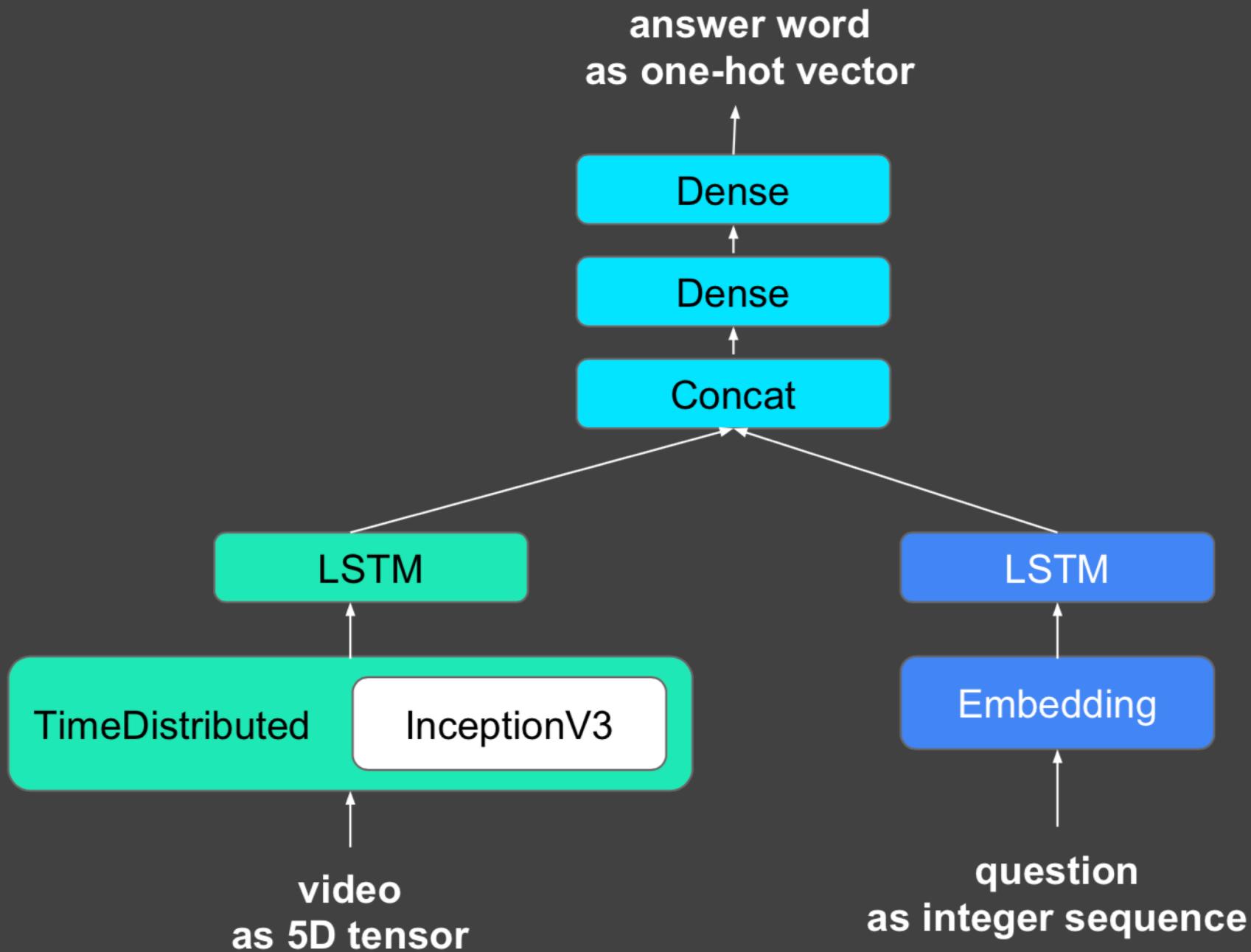


# from frames to a vector









# Turning frames into a vector, with pre-trained representations

Keras Functional API

```
import keras  
from keras import layers  
from keras.applications import InceptionV3
```

Pretrained “Application”

```
video = keras.Input(shape=(None, 150, 150, 3), name='video')  
cnn = InceptionV3(weights='imagenet',  
                    include_top=False,  
                    pooling='avg')
```

cnn.trainable = False “Application” read only

```
frame_features = layers.TimeDistributed(cnn)(video)
```

```
video_vector = layers.LSTM(256)(frame_features) Built in layer types
```

# Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                    include_top=False,
                    pooling='avg')  
"Application" read only
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)  
Built in layer types
```

# Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                    include_top=False,
                    pooling='avg')
cnn.trainable = False                                     "Application" read only
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)           Built in layer types
```

# Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                    include_top=False,
                    pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

“Application” read only

Built in layer types

# Turning frames into a vector, with pre-trained representations

```
import keras
from keras import layers
from keras.applications import InceptionV3

video = keras.Input(shape=(None, 150, 150, 3), name='video')
cnn = InceptionV3(weights='imagenet',
                    include_top=False,
                    pooling='avg')
cnn.trainable = False
frame_features = layers.TimeDistributed(cnn)(video)
video_vector = layers.LSTM(256)(frame_features)
```

## Turning a sequence of words into a vector

```
question = keras.Input(shape=(None, ), dtype='int32', name='question')
embedded_words = layers.Embedding(input_voc_size, 256)(question)
question_vector = layers.LSTM(128)(embedded_words)
```

## Predicting an answer word

```
x = layers.concatenate([video_vector, question_vector])
x = layers.Dense(128, activation=tf.nn.relu)(x)      Tensorflow call
predictions = layers.Dense(output_voc_size,
                           activation='softmax',
                           name='predictions')(x)
```

# Setting up the training configuration

```
model = keras.models.Model([video, question], predictions)
model.compile(optimizer=tf.AdamOptimizer(),           Tensorflow call
               loss=keras.losses.categorical_crossentropy)
```

```
model.fit_generator(data_generator,
                     steps_per_epoch=1000,
                     epochs=100)
```

Data\_generator streams data too large  
to fit into memory

<https://stanford.edu/~shervine/blog/keras-how-to-generate-data-on-the-fly>

Distributed,  
multi-GPU,  
& TPU training

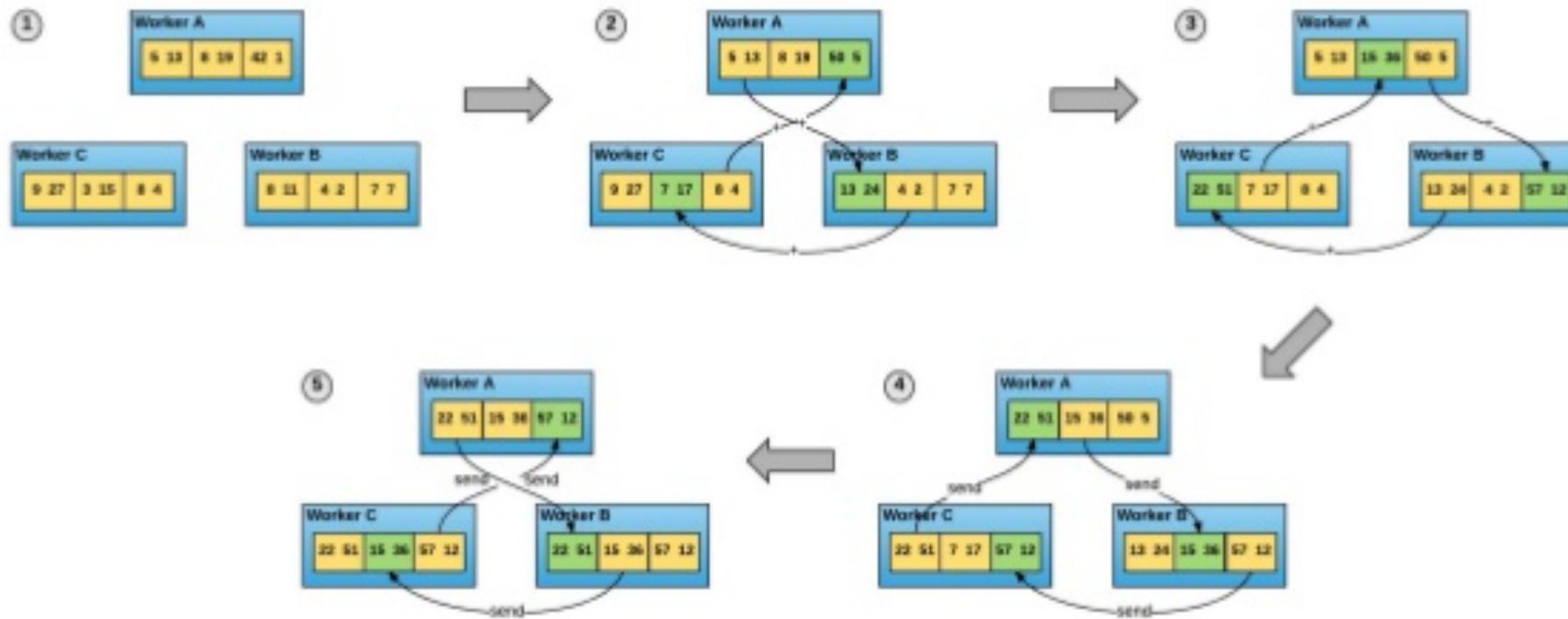
# Data Parallel (Distributed Training)

---

- Horovod (from Uber)
- Estimator API (TF built-in option)
- Dist-Keras (Spark)
  - Elephas (Spark)
- Built-in Multi-GPU support (requires multiple GPUs in one address space, such as on a DGX workstation)



# Horovod Technique



Patarasuk, P., & Yuan, X. (2009). Bandwidth optimal all-reduce algorithms for clusters of workstations. *Journal of Parallel and Distributed Computing*, 69(2), 117-124. doi:10.1016/j.jpdc.2008.09.002

UBER



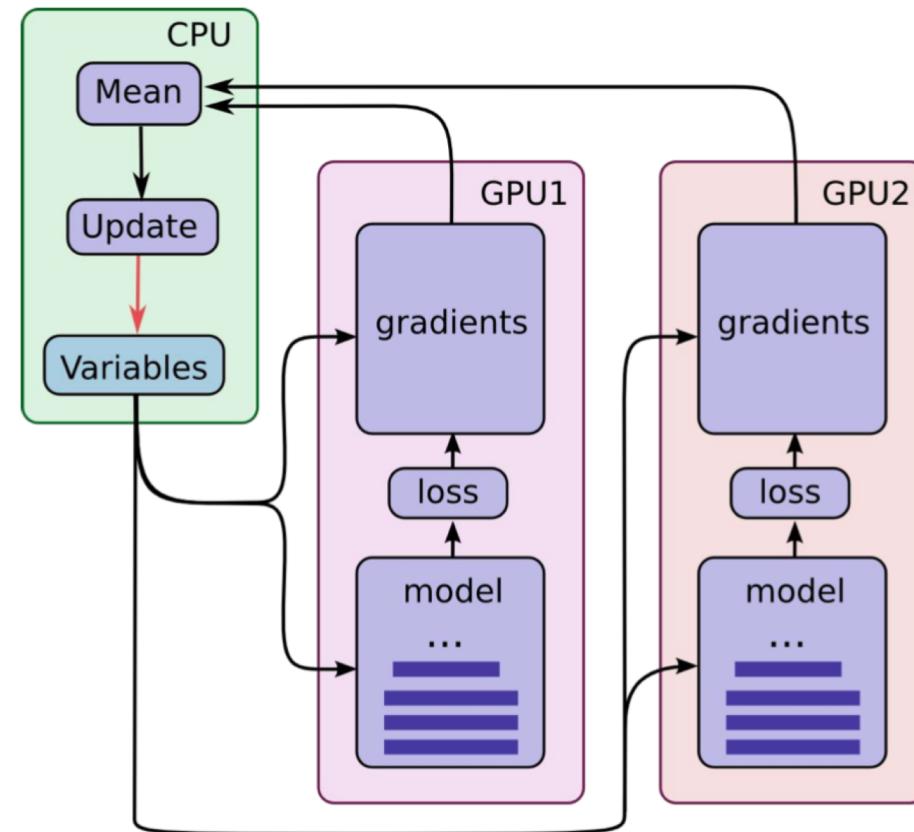
# Built-in Multi-GPU support

```
import tensorflow as tf
from keras.applications import Xception
from keras.utils import multi_gpu_model

# Instantiate the base model
# (here, we do it on CPU, which is optional).
with tf.device('/cpu:0'):
    model = Xception(weights=None,
                        input_shape=(height, width, 3),
                        classes=num_classes)

# Replicates the model on 8 GPUs.
# This assumes that your machine has 8 available GPUs.
parallel_model = multi_gpu_model(model, gpus=8)
parallel_model.compile(loss='categorical_crossentropy',
                       optimizer='rmsprop')

# This `fit` call will be distributed on 8 GPUs.
# Since the batch size is 256, each GPU will process 32 samples.
parallel_model.fit(x, y, epochs=20, batch_size=256)
```





# TPU support

Training + inference

Via Estimator API





# Deferred (symbolic) vs eager (Imperative)

- **Deferred**: you use Python to build a computational graph that gets executed later
- **Eager**: the Python runtime is the execution runtime (like Numpy)
- In brief:
  - Symbolic tensors **don't have a value** in your Python code (yet)
  - Eager tensors **have a value** in your Python code
  - With eager execution, you can use value-dependent dynamics topologies (tree-RNNs, etc.)

# The Keras functional API and Sequential API work with eager execution

```
import keras
from keras import layers
import tensorflow.contrib.eager as tfe

tfe.enable_eager_execution()

inputs = keras.Input(shape=(10,))
x = layers.Dense(20, activation='relu')(x)
x = layers.Dense(20, activation='relu')(x)
outputs = layers.Dense(10, activation='softmax')(x)

model = keras.Model(inputs, outputs)

preds = model(x) # Works on *value arrays*
model.fit(x, y, epochs=10, batch_size=32) # The Model API works too.
```

# Eager Execution

- Eager execution is a flexible machine learning platform for research and experimentation, providing:
- *An intuitive interface*—Structure your code naturally and use Python data structures. Quickly iterate on small models and small data.
- *Easier debugging*—Call ops directly to inspect running models and test changes. Use standard Python debugging tools for immediate error reporting.
- *Natural control flow*—Use Python control flow instead of graph control flow, simplifying the specification of dynamic models.

---

# Dynamic Network Surgery for Efficient DNNs

---

**Yiwen Guo\***  
Intel Labs China  
[yiwen.guo@intel.com](mailto:yiwen.guo@intel.com)

**Anbang Yao**  
Intel Labs China  
[anbang.yao@intel.com](mailto:anbang.yao@intel.com)

**Yurong Chen**  
Intel Labs China  
[yurong.chen@intel.com](mailto:yurong.chen@intel.com)

## Abstract

Deep learning has become a ubiquitous technology to improve machine intelligence. However, most of the existing deep models are structurally very complex, making them difficult to be deployed on the mobile platforms with limited computational power. In this paper, we propose a novel network compression method called dynamic network surgery, which can remarkably reduce the network complexity by making on-the-fly connection pruning. Unlike the previous methods which accomplish this task in a greedy way, we properly incorporate connection splicing into the whole process to avoid incorrect pruning and make it as a continual network maintenance. The effectiveness of our method is proved with experiments. Without any accuracy loss, our method can efficiently compress the number of parameters in LeNet-5 and AlexNet by a factor of  $108\times$  and  $17.7\times$  respectively, proving that it outperforms the recent pruning method by considerable margins. Code and some models are available at <https://github.com/yiwenguo/Dynamic-Network-Surgery>.

## 1 Introduction

As a family of brain inspired models, deep neural networks (DNNs) have substantially advanced a variety of artificial intelligence tasks including image classification [13, 19, 11], natural language processing, speech recognition and face recognition.

Despite these tremendous successes, recently designed networks tend to have more stacked layers, and thus more learnable parameters. For instance, AlexNet [13] designed by Krizhevsky et al. has 61 million parameters to win the ILSVRC 2012 classification competition, which is over 100 times more than that of LeCun’s conventional model [15] (e.g., LeNet-5), let alone the much more complex models like VGGNet [19]. Since more parameters means more storage requirement and more floating-point operations (FLOPs), it increases the difficulty of applying DNNs on mobile platforms with limited memory and processing units. Moreover, the battery capacity can be another bottleneck [9].

Although DNN models normally require a vast number of parameters to guarantee their superior performance, significant redundancies have been reported in their parameterizations [4]. Therefore, with a proper strategy, it is possible to compress these models without significantly losing their prediction accuracies. Among existing methods, network pruning appears to be an outstanding one due to its surprising ability of accuracy loss prevention. For instance, Han et al. [9] recently propose to