

# Anime-fy Yourself

Using Core ML

Yono Mittlefehldt – @yonomitt – 2021

# Who is Yono?

Co-creator of [Gus on the Go](#)

# Who is Yono?

Co-creator of [Gus on the Go](#)

Freelance software developer  
specializing in Computer Vision and iOS



# Who is Yono?

Co-creator of [Gus on the Go](#)

Freelance software developer  
specializing in Computer Vision and iOS

Tutorial writer for [raywenderlich.com](#)



# Who is Yono?

Co-creator of [Gus on the Go](#)

Freelance software developer  
specializing in Computer Vision and iOS

Tutorial writer for [raywenderlich.com](#)

Currently rewatching all of Futurama  
while exercising on a stationary bike



# Converting a Model to Core ML

# Model

# Model

A function

$$f(x, y, z) = w_1x + w_2y + w_3z + b$$

# Model

$$f(\mathbf{x}, \mathbf{y}, \mathbf{z}) = w_1\mathbf{x} + w_2\mathbf{y} + w_3\mathbf{z} + b$$

A function

One or more inputs

# Model

$$\mathbf{f(x, y, z)} = w_1x + w_2y + w_3z + b$$

A function

One or more inputs

One or more outputs

# Model

$$f(x, y, z) = w_1x + w_2y + w_3z + b$$

A function

One or more inputs

One or more outputs

Parameters

# Model

$$f(x, y, z) = \mathbf{w}_1x + \mathbf{w}_2y + \mathbf{w}_3z + b$$

A function

One or more inputs

One or more outputs

Parameters

Weights (multiplied)

# Model

$$f(x, y, z) = w_1x + w_2y + w_3z + \mathbf{b}$$

A function

One or more inputs

One or more outputs

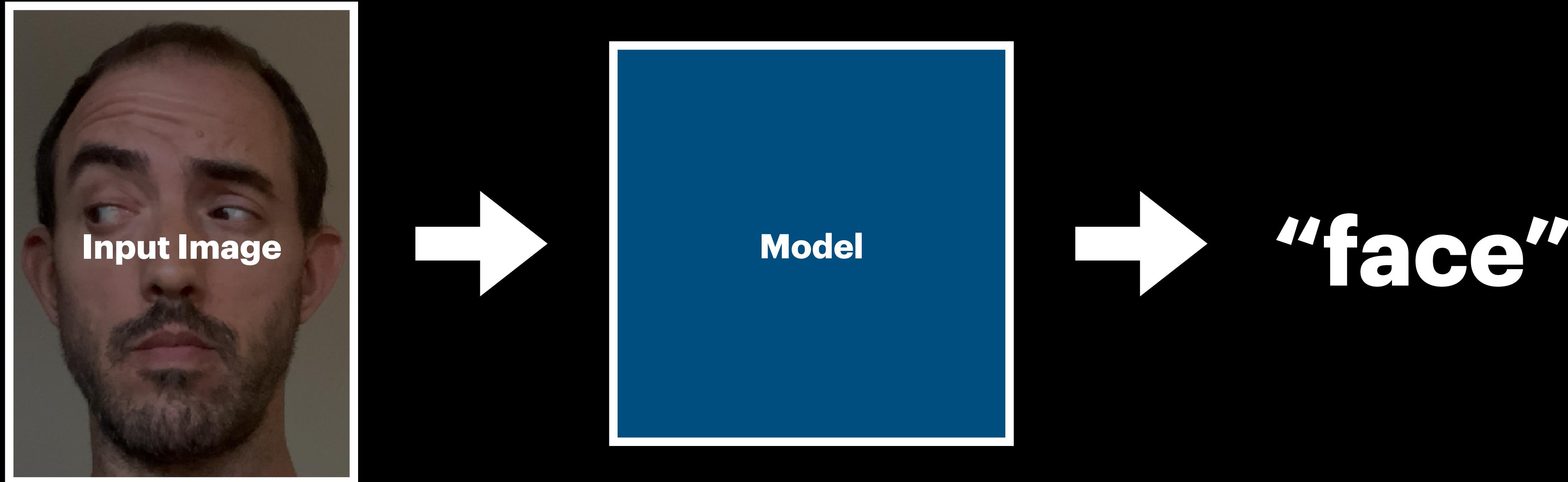
Parameters

Weights (multiplied)

Biases (added)

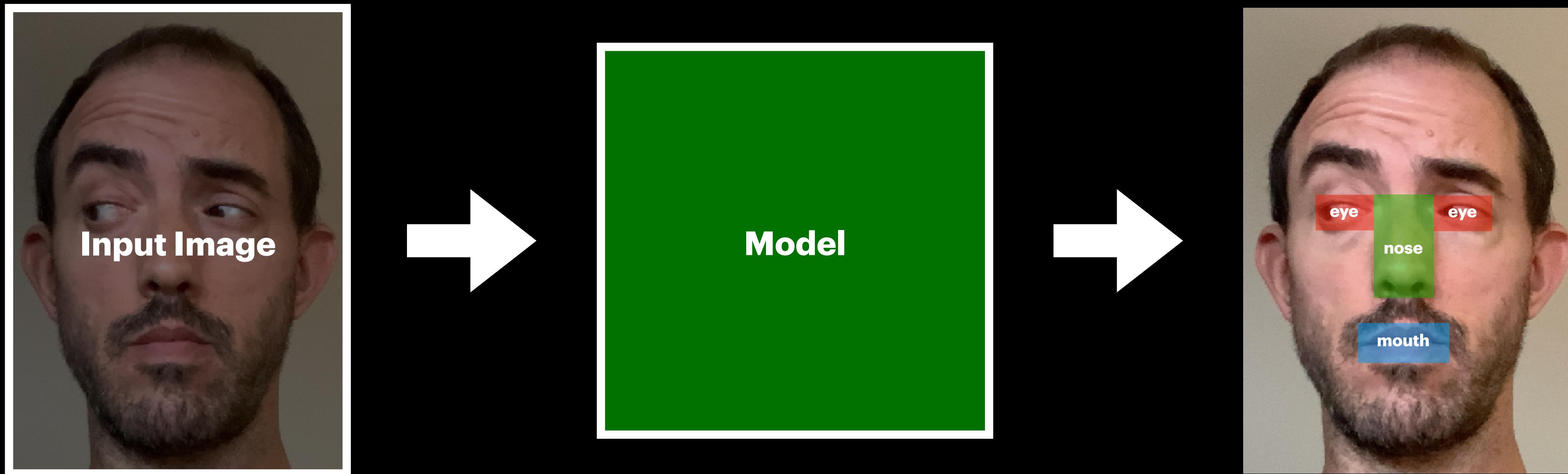
# Model Types

## Image Classification



# Model Types

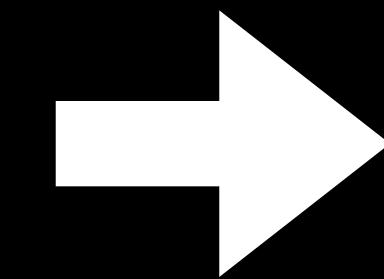
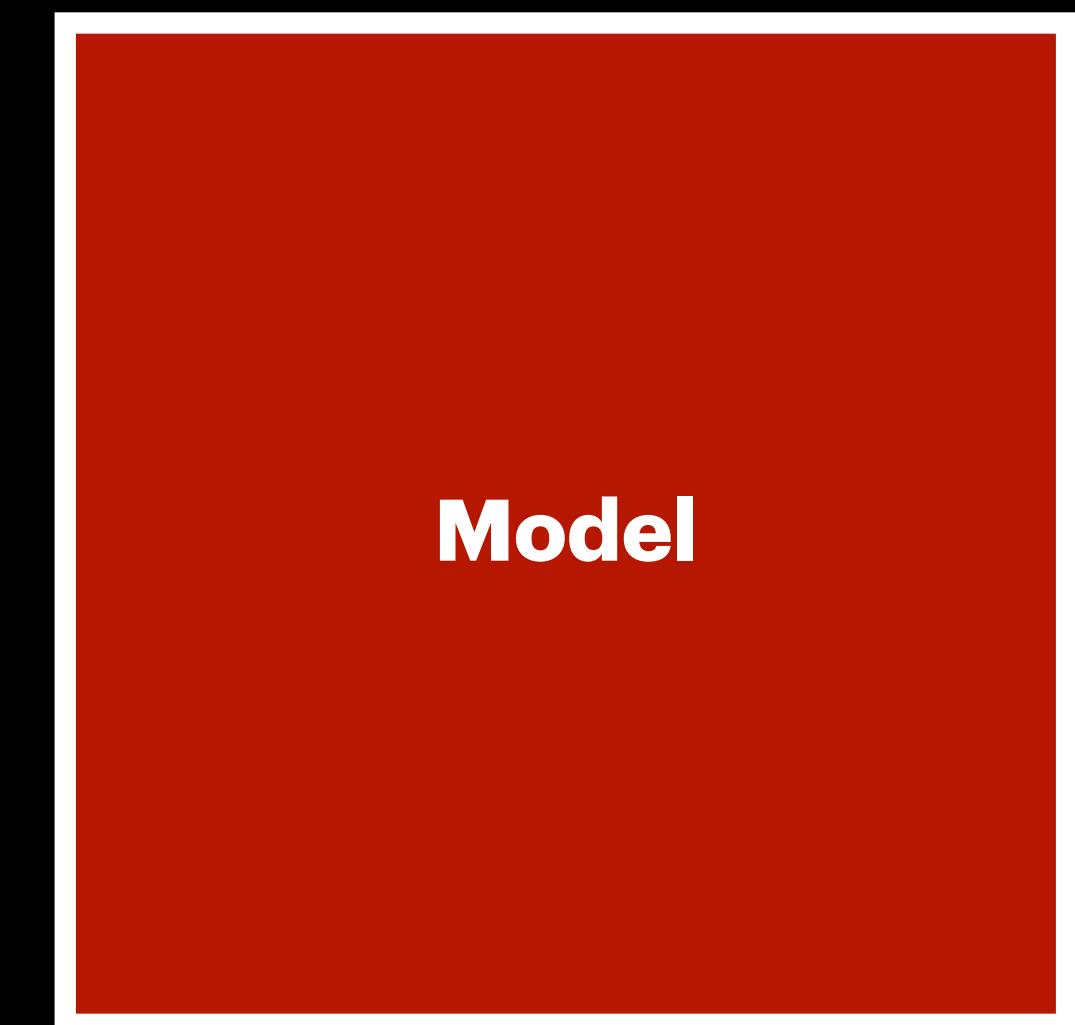
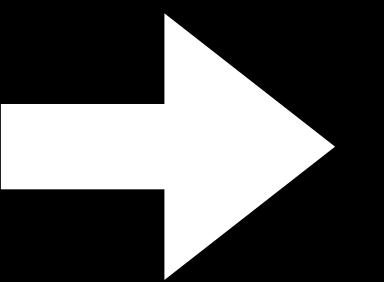
## Object Detection



# Model Types

## Text Generation

**“This talk is”**



**“This talk is  
wonderfully  
informative and will  
be remembered for  
many minutes  
afterward.”**

# Models Need Numbers

Encode Inputs and Decode Outputs

# Models Need Numbers

Encode Inputs and Decode Outputs

Images can be expressed as multidimensional arrays of pixel values

# Models Need Numbers

Encode Inputs and Decode Outputs

Images can be expressed as multidimensional arrays of pixel values

Separate values for each RGB channel

# Models Need Numbers

Encode Inputs and Decode Outputs

Images can be expressed as multidimensional arrays of pixel values

Separate values for each RGB channel

Scale is usually important – [0, 255] vs [0.0, 1.0] vs [-1.0, 1.0]

# Models Need Numbers

## Encode Inputs and Decode Outputs

Images can be expressed as multidimensional arrays of pixel values

Separate values for each RGB channel

Scale is usually important – [0, 255] vs [0.0, 1.0] vs [-1.0, 1.0]

Text can be encoded as integers

# Models Need Numbers

## Encode Inputs and Decode Outputs

Images can be expressed as multidimensional arrays of pixel values

Separate values for each RGB channel

Scale is usually important – [0, 255] vs [0.0, 1.0] vs [-1.0, 1.0]

Text can be encoded as integers

“This”, “this” -> 55

“talk” -> 743

# Models Need Numbers

## Encode Inputs and Decode Outputs

Images can be expressed as multidimensional arrays of pixel values

Separate values for each RGB channel

Scale is usually important – [0, 255] vs [0.0, 1.0] vs [-1.0, 1.0]

Text can be encoded as integers

“This”, “this” -> 55

“talk” -> 743

These values may be then encoded as one-hot vectors

# Frameworks to Create Models

PyTorch

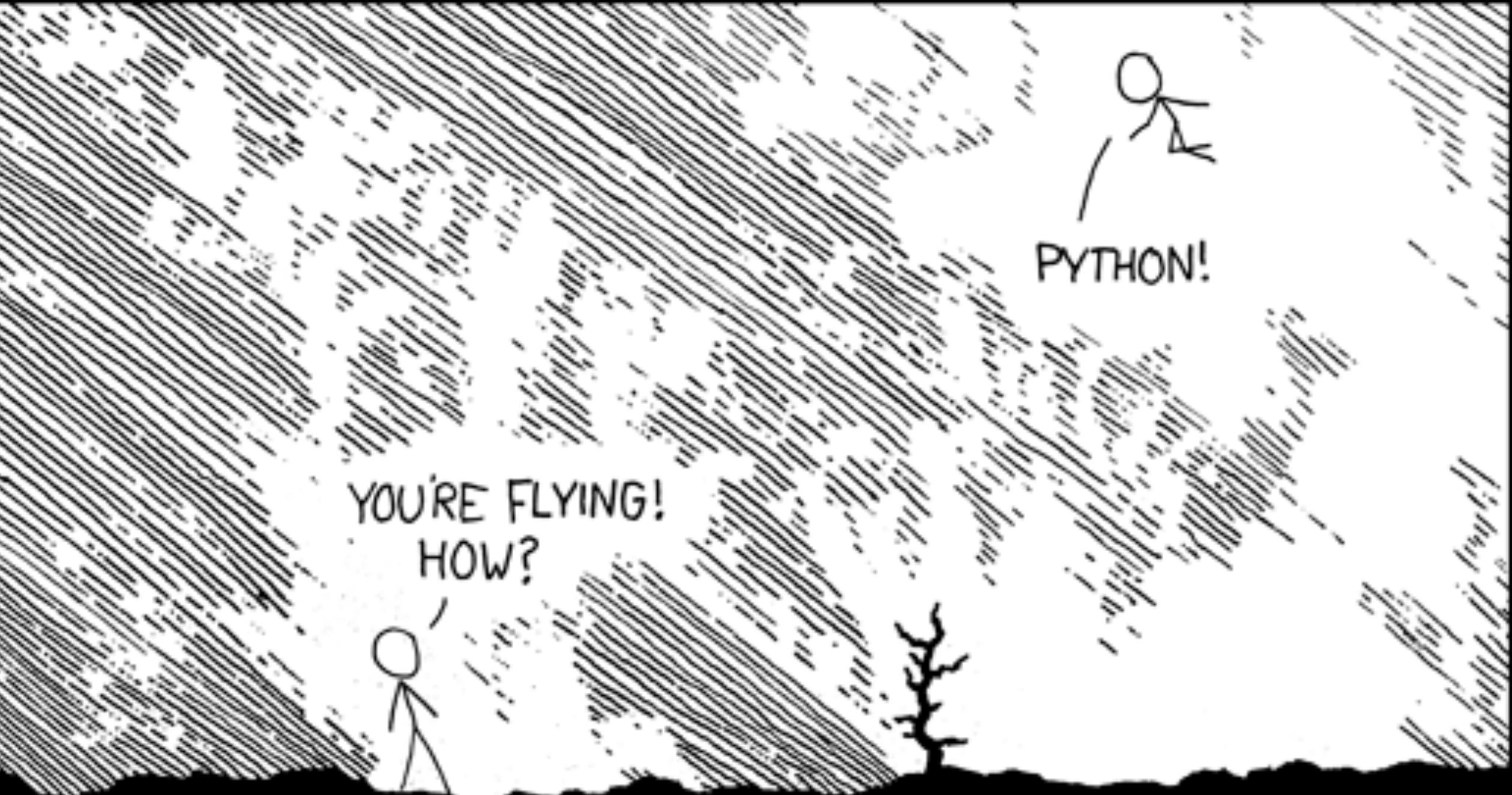
Tensorflow

Turi Create / Create ML

...

scikit-learn

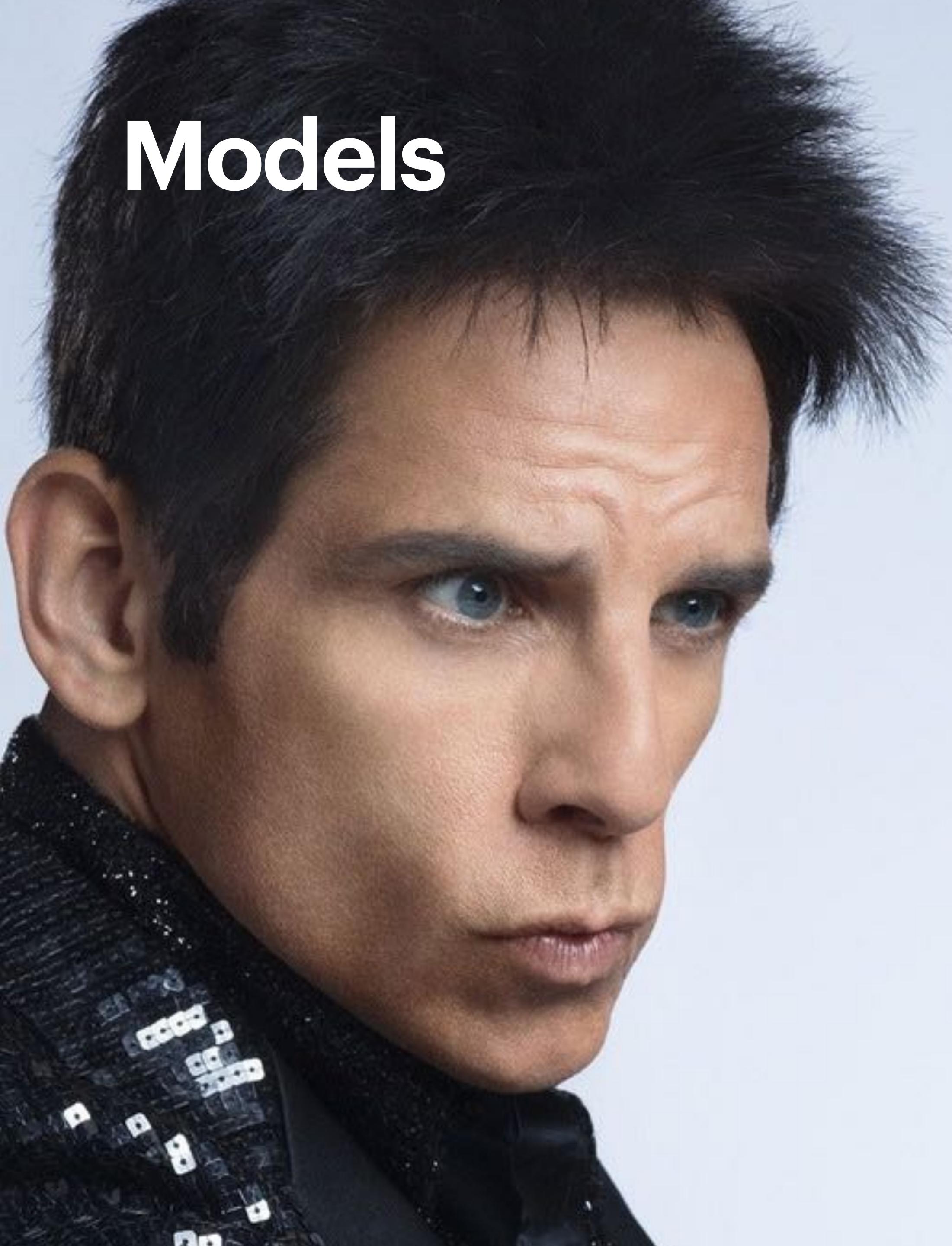
# Py in PyTorch



**Hey! What about the alt-text?**

I wrote 20 short programs in Python  
yesterday. It was wonderful. Perl, I'm leaving  
you.

# Models



# Models - Built In

The screenshot shows a web browser displaying the PyTorch documentation at [pytorch.org/vision/stable/models.html](https://pytorch.org/vision/stable/models.html). The page is titled "TORCHVISION.MODELS". The navigation bar includes links for "Get Started", "Ecosystem", "Mobile", "Blog", "Tutorials", "Docs", "Resources", and "GitHub". The "Docs" menu is currently selected. A sidebar on the left provides navigation through various PyTorch components like "Package Reference", "torchvision.datasets", "torchvision.io", and "torchvision.models". The main content area discusses the "models" subpackage, which contains definitions for tasks like image classification, semantic segmentation, object detection, instance segmentation, person keypoint detection, and video classification. It includes a note about backward compatibility and a section on "Classification" which lists several pre-trained models: AlexNet, VGG, ResNet, SqueezeNet, DenseNet, Inception v3, GoogLeNet, ShuffleNet v2, MobileNet v2, MobileNet v3, ResNext, Wide ResNet, MNASNet, EfficientNet, RegNet, Quantized Models, Semantic Segmentation, Fully Convolutional Networks, and DeepLabV3.

PyTorch

Get Started Ecosystem Mobile Blog Tutorials Docs Resources GitHub

0.11.0 ▾

Docs > torchvision.models

Shortcuts

## TORCHVISION.MODELS

The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

• NOTE

Backward compatibility is guaranteed for loading a serialized `state_dict` to the model created using old PyTorch version. On the contrary, loading entire saved models or serialized `ScriptModules` (serialized using older versions of PyTorch) may not preserve the historic behaviour. Refer to the following documentation

### Classification

The models subpackage contains definitions for the following model architectures for image classification:

- [AlexNet](#)
- [VGG](#)
- [ResNet](#)
- [SqueezeNet](#)
- [DenseNet](#)
- [Inception v3](#)
- [GoogLeNet](#)
- [ShuffleNet v2](#)
- [MobileNet v2](#)
- [MobileNet v3](#)
- [ResNext](#)
- [Wide ResNet](#)
- [MNASNet](#)
- [EfficientNet](#)
- [RegNet](#)
- [Quantized Models](#)

– Semantic Segmentation

- [Fully Convolutional Networks](#)
- [DeepLabV3](#)

torchvision.models

– Classification

- Alexnet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- MobileNet v3
- ResNext
- Wide ResNet
- MNASNet
- EfficientNet
- RegNet
- Quantized Models

– Semantic Segmentation

- Fully Convolutional Networks
- DeepLabV3

# Models - Built In

The screenshot shows a web browser displaying the TensorFlow API documentation for the `tf.keras.applications` module. The URL in the address bar is `tensorflow.org`. The page title is "Module: `tf.keras.applications`". On the left, there is a navigation sidebar with links to various TensorFlow modules like `tf.graph_util`, `tf.image`, etc., and specific models such as `MobileNetV3Large`, `densenet`, `efficientnet`, etc. The `tf.keras.applications` section is currently selected. The main content area contains a brief description of the `tf.keras.applications` module, followed by a list of its sub-modules: `densenet`, `efficientnet`, `imagenet_utils`, `inception_resnet_v2`, `inception_v3`, `mobilenet`, `mobilenet_v2`, `mobilenet_v3`, `nasnet`, `resnet`, and `resnet50`. There is also a note about TensorFlow 1 version compatibility.

# Models - Built In

```
import torchvision.models as models  
  
model = models.efficientnet_b5(pretrained=True)
```

```
import tensorflow as tf  
  
model = tf.keras.applications.efficientnet.EfficientNetB5(weights='imagenet')
```

# Models - Hubs

The screenshot shows the PyTorch Models - Hubs page on the official website. The header includes the PyTorch logo, navigation links for Get Started, Ecosystem, Mobile, Blog, Tutorials, Docs, Resources (highlighted in red), GitHub, and a search icon. Below the header, a large section titled "FOR RESEARCHERS — EXPLORE AND EXTEND MODELS FROM THE LATEST CUTTING EDGE RESEARCH" is displayed. It features five model cards: 3D ResNet, SlowFast, X3D, YOLOP, and another SlowFast entry. Each card includes a brief description, a "Sort" dropdown, and a detailed diagram of the network architecture.

**3D ResNet** 2.1k

Resnet Style Video classification networks pretrained on the Kinetics 400 dataset

**SlowFast** 2.1k

SlowFast networks pretrained on the Kinetics 400 dataset

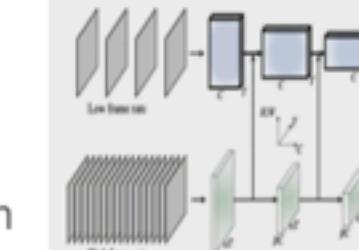
**X3D** 2.1k

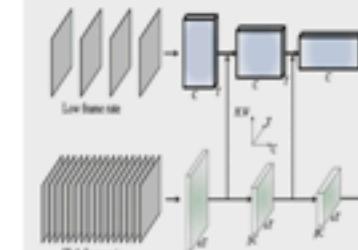
X3D networks pretrained on the Kinetics 400 dataset

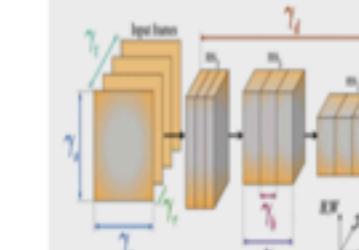
**YOLOP** 768

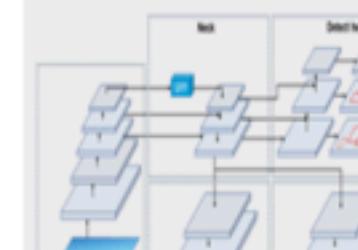
YOLOP pretrained on the BDD100K dataset

**Sort ▾**

**3D ResNet**   
Figure 1. A 3D ResNet has a low frame rate, low temporal resolution Slow pathway and a high frame rate,  $\alpha \times$  higher temporal resolution Fast pathway. The Fast pathway is lightweight by using a fraction ( $\beta$ , e.g., 1/8) of channels. Lateral connections fuse them.

**SlowFast**   
Figure 1. A SlowFast network has a low frame rate, low temporal resolution Slow pathway and a high frame rate,  $\alpha \times$  higher temporal resolution Fast pathway. The Fast pathway is lightweight by using a fraction ( $\beta$ , e.g., 1/8) of channels. Lateral connections fuse them.

**X3D**   
Figure 1. X3D networks progressively expand a 2D network across the following axes: Temporal duration  $\gamma_t$ , frame rate  $\gamma_f$ , spatial  $\gamma_s$ , and depth  $\gamma_d$ .

**YOLOP** 

# Models - Hubs

The screenshot shows the TensorFlow Hub website interface. On the left, there's a sidebar with navigation links like Home, All collections, All models, and All publishers. Below that are sections for Problem domains (Image, Text, Video, Audio), Model format (TF.js, TFLite, Coral), and a search bar at the top.

**Text Problem Domains**

- Embedding (196)
- Language model (98)
- Preprocessing (14)
- Generation (8)

**Image Problem Domains**

- Classification (274)
- Feature vector (217)
- Object detection (67)
- Segmentation (42)

**Video Problem Domains**

# Models - Hubs

```
import torch  
  
model = torch.hub.load('ultralytics/yolov5', 'yolov5s', pretrained=True)
```

```
import tensorflow_hub as tfhub  
  
model = tfhub.load('https://tfhub.dev/tensorflow/efficientnet/b5/classification/1')
```

# Models - Model Zoos

The screenshot shows a web browser window displaying the ModelZoo website ([modelzoo.co](https://modelzoo.co)). The page has a dark theme with a large central header "Model Zoo" and a subtitle "Discover open source deep learning code and pretrained models." Below the header are two purple buttons: "Browse Frameworks" and "Browse Categories". At the top of the page, there is a navigation bar with links for "ModelZoo", "Frameworks", "Categories", "Collections", "Suggest a Model", "Buy Me a Coffee", "Blog", and "About". A search bar with a magnifying glass icon and the placeholder "Filter models..." is located below the header. Three model cards are displayed at the bottom:

- OpenPose** (14800 stars): OpenPose represents the first real-time multi-person system to jointly detect human body, hand, and facial keypoints (in total 130 keypoints) on single images.
- Mask R-CNN** (14504 stars): This is an implementation of Mask R-CNN on Python 3, Keras, and TensorFlow. The model generates bounding boxes and segmentation masks for each instance of an object in the image. It's based on Feature
- pytorch-CycleGAN-and-pix2pix** (9980 stars): PyTorch implementation for both unpaired and paired image-to-image translation.

# Models - Model Zoos

The screenshot shows a web browser window for developer.apple.com with the URL <https://developer.apple.com/machine-learning/models/>. The page is titled "Machine Learning" and has tabs for "Overview", "Core ML", "Create ML", "APIs", "Models", and "Resources". Below the tabs, there are two tabs: "Images" (which is selected) and "Text". The main content area is titled "Images" and displays three examples:

- FCRN-DepthPrediction**: Depth Estimation. It shows a bedroom interior with a bed and two orange stools at the foot. A blue button labeled "View Models" is present.
- MNIST**: Drawing Classification. It shows a handwritten digit "7" with a confidence score of 7 : 99.96%. A blue button labeled "View Model" is present.
- UpdatableDrawingClassifier**: Drawing Classification. It shows a sun and a cloud with rays. A blue button labeled "View Model and Code Sample" is present.

**YOU GET A MODEL**



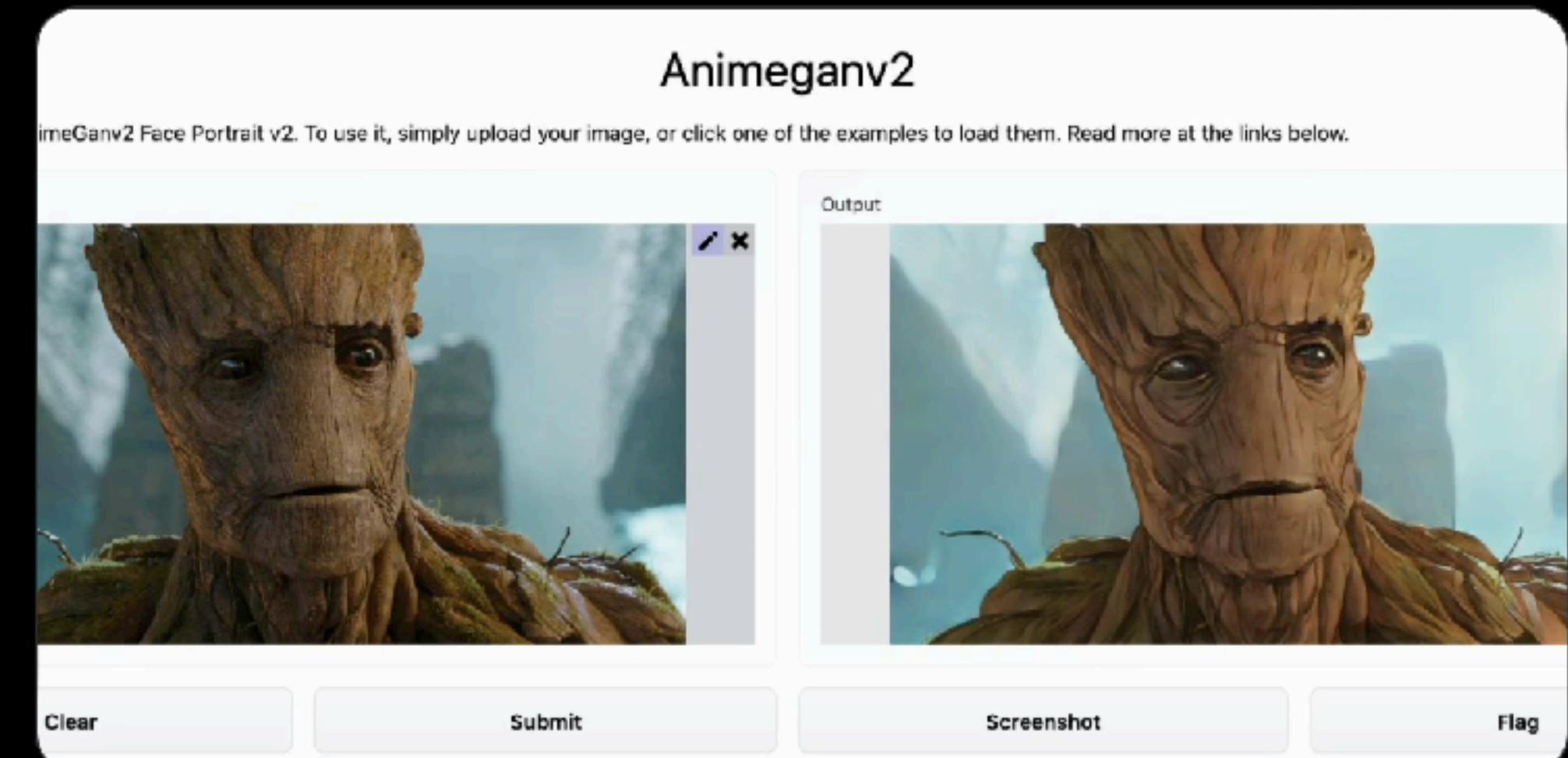
**AND YOU GET A MODEL, AND YOU...**

# Models



AK  
@ak92501

@Gradio Demo for AnimeGANv2 Face Portrait v2 now on @huggingface Spaces  
demo: [huggingface.co/spaces/akhaliq...](https://huggingface.co/spaces/akhaliq...)  
github: [github.com/bryandlee/anim...](https://github.com/bryandlee/anim...)



6:13 PM · Nov 6, 2021 · Twitter Web App

349 Retweets 351 Quote Tweets 2,022 Likes



Tweet your reply

Reply



AK @ak92501 · Nov 6  
Replies to @ak92501

...



AK  
@ak92501

...

.@Gradio Demo for AnimeGANv2 Face Portrait v2 now  
on @huggingface Spaces  
demo: [huggingface.co/spaces/akhaliq/AnimeGANv2](https://huggingface.co/spaces/akhaliq/AnimeGANv2)  
github: [github.com/bryandlee/animganv2](https://github.com/bryandlee/animganv2)



### Animeganv2

AnimeGANv2 Face Portrait v2. To use it, simply upload your image, or click one of the examples to load them. Read more at the links below.



**Output**

**Clear** **Submit** **Screenshot** **Flag**

A screenshot of a GitHub repository page for `bryandlee/animegan2-pytorch`. The repository is public and has 40 stars, 2.3k forks, and 272 issues. The `Code` tab is selected. A red arrow points to the commit history for the file `hubconf.py`.

The commit history shows:

- bryandlee Docs: Update sample results** (25d7b01, 8 days ago) - 72 commits
- `samples` (Docs: Remove face samples, 8 days ago)
- `weights` (Add weights from gdrive, last month)
- `.gitignore` (Initial commit, 10 months ago)
- `LICENSE` (Create LICENSE, 4 months ago)
- `README.md` (Docs: Update sample result, 8 days ago)
- `colab_demo.ipynb` (Fix uploader, 20 days ago)
- `convert_weights.py` (typo fixed, 10 months ago)
- `demo.ipynb` (Feat: Update notebook demo with hub model, 21 days ago)
- hubconf.py** (Refactor: Minor formatting, 8 days ago)
- `model.py` (additional cli, 9 months ago)
- `requirements.txt` (Docs: Add requirements, 8 days ago)

**About**: PyTorch implementation of AnimeGANv2. Tags: gan, style-transfer, image2image.

**Contributors**: bryandlee, xhlulu Xing Han Lu, AK391, Sxela Alex.

**Languages**: (not explicitly listed)

# Examples



**But we don't want a PyTorch model!**

We want a Core ML model...

# Core ML Tools

Convert models from popular frameworks to Core ML using Python

PyTorch

Tensorflow

scikit-learn

(to appease the machine learning purists)

Read, write, and optimize Core ML models

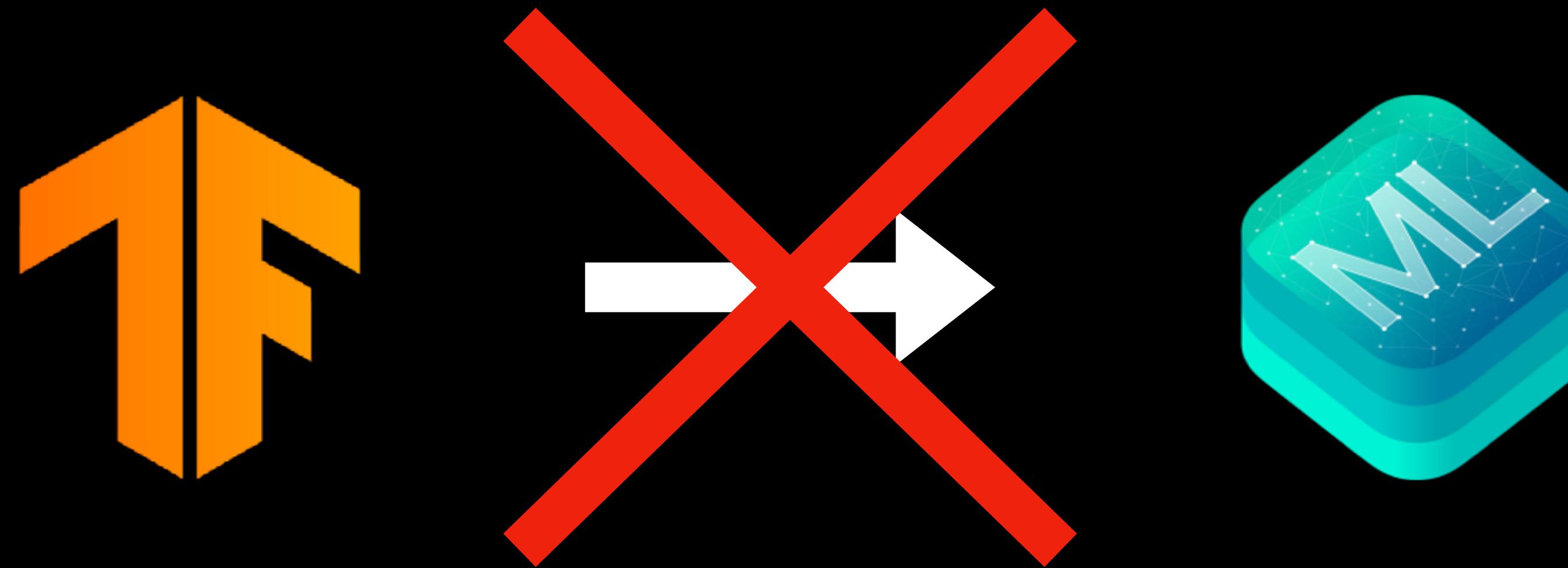


# Core ML Tools

Long, long ago

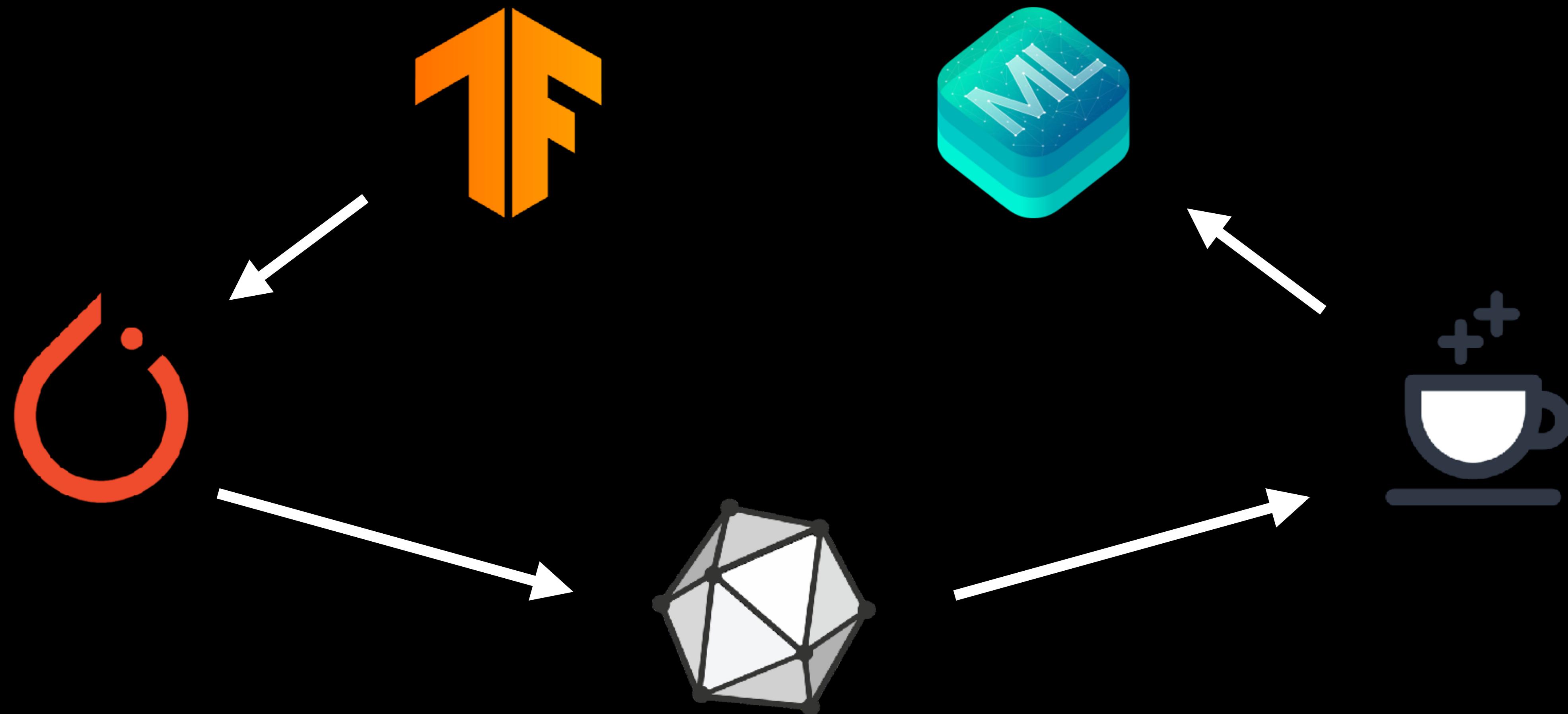
# Core ML Tools

Long, long ago (2019)



# Core ML Tools

Long, long ago (2019)



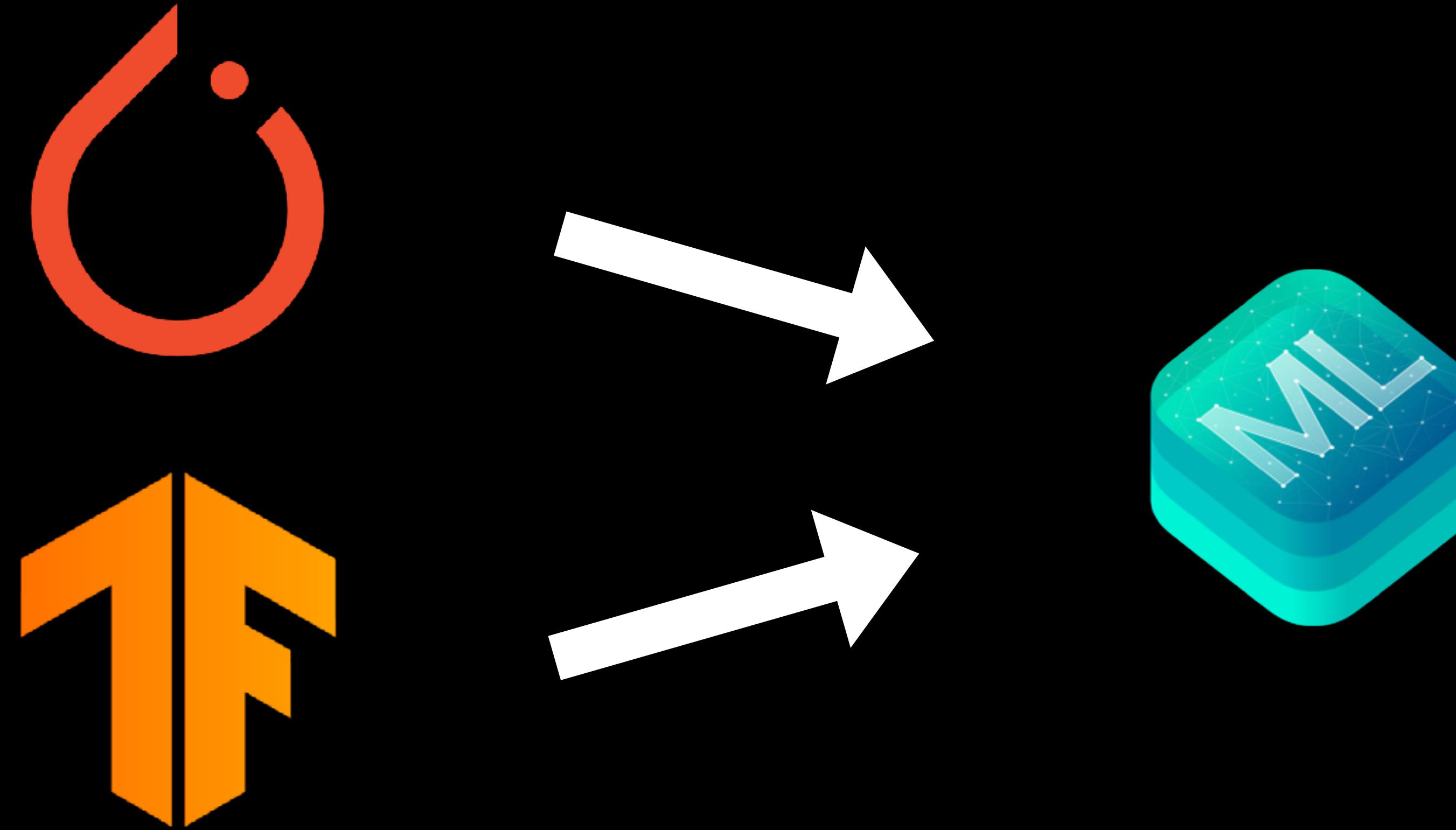
# Core ML Tools

## Today



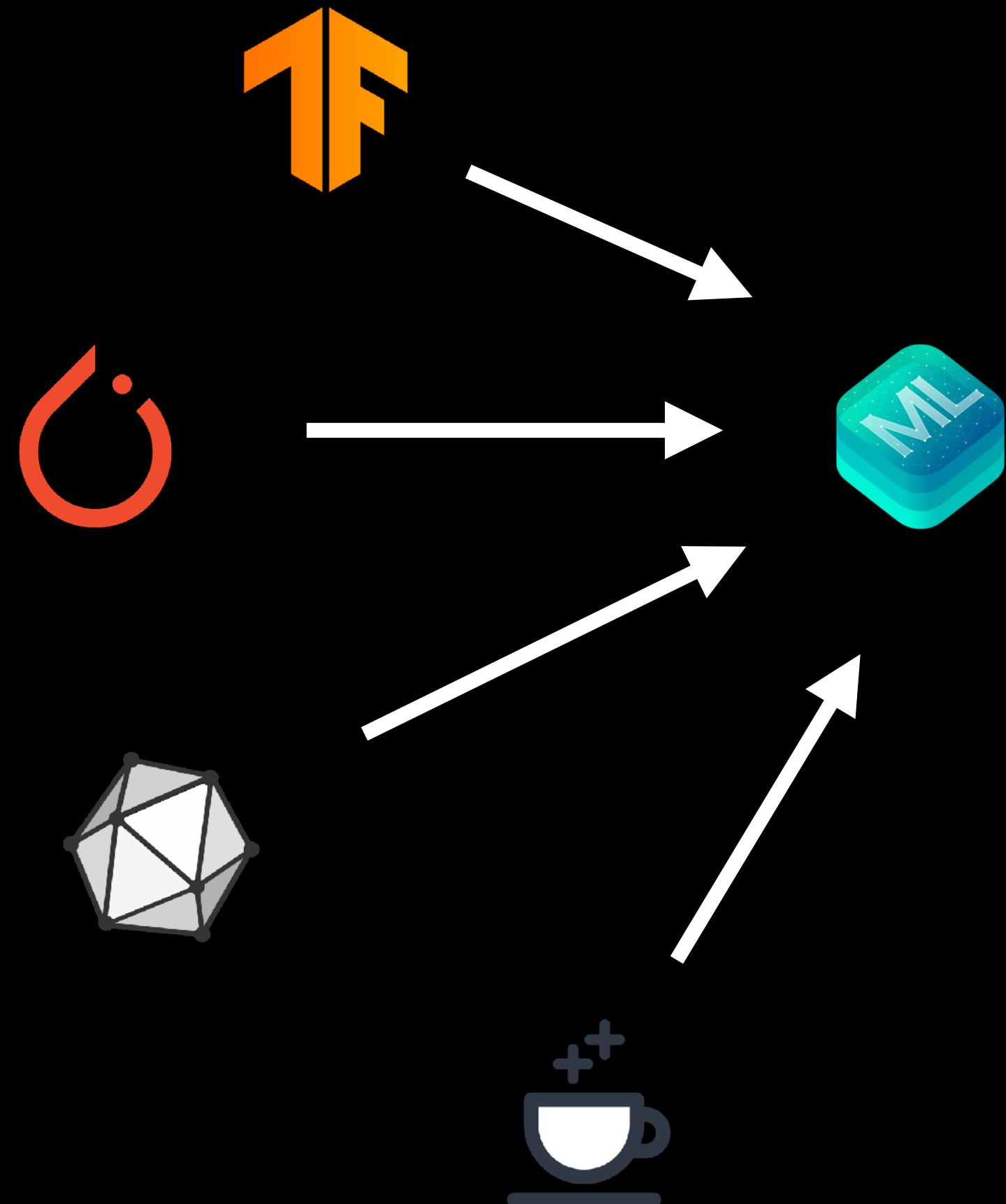
# Core ML Tools

## Today

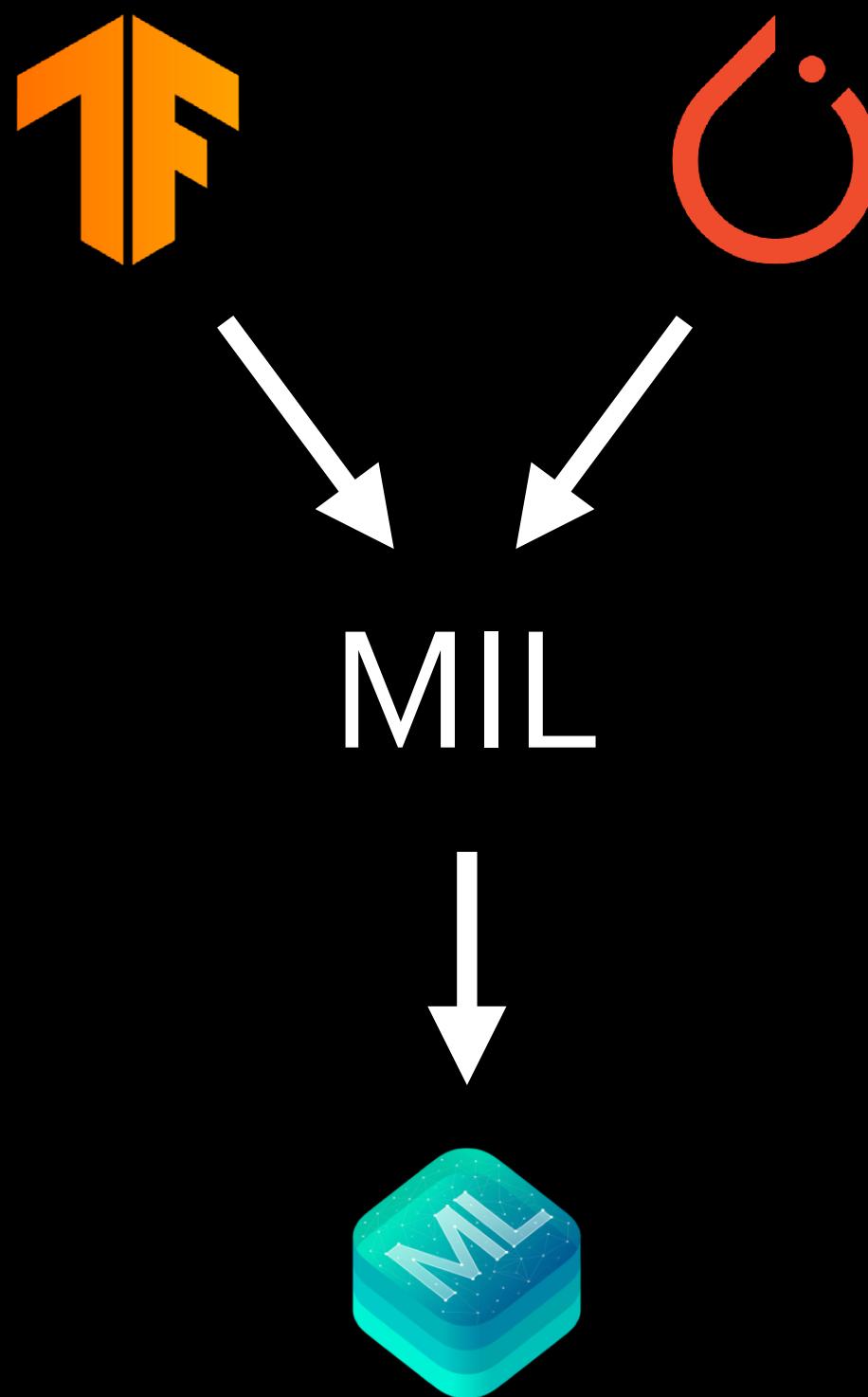


# What changed?

2019



2021

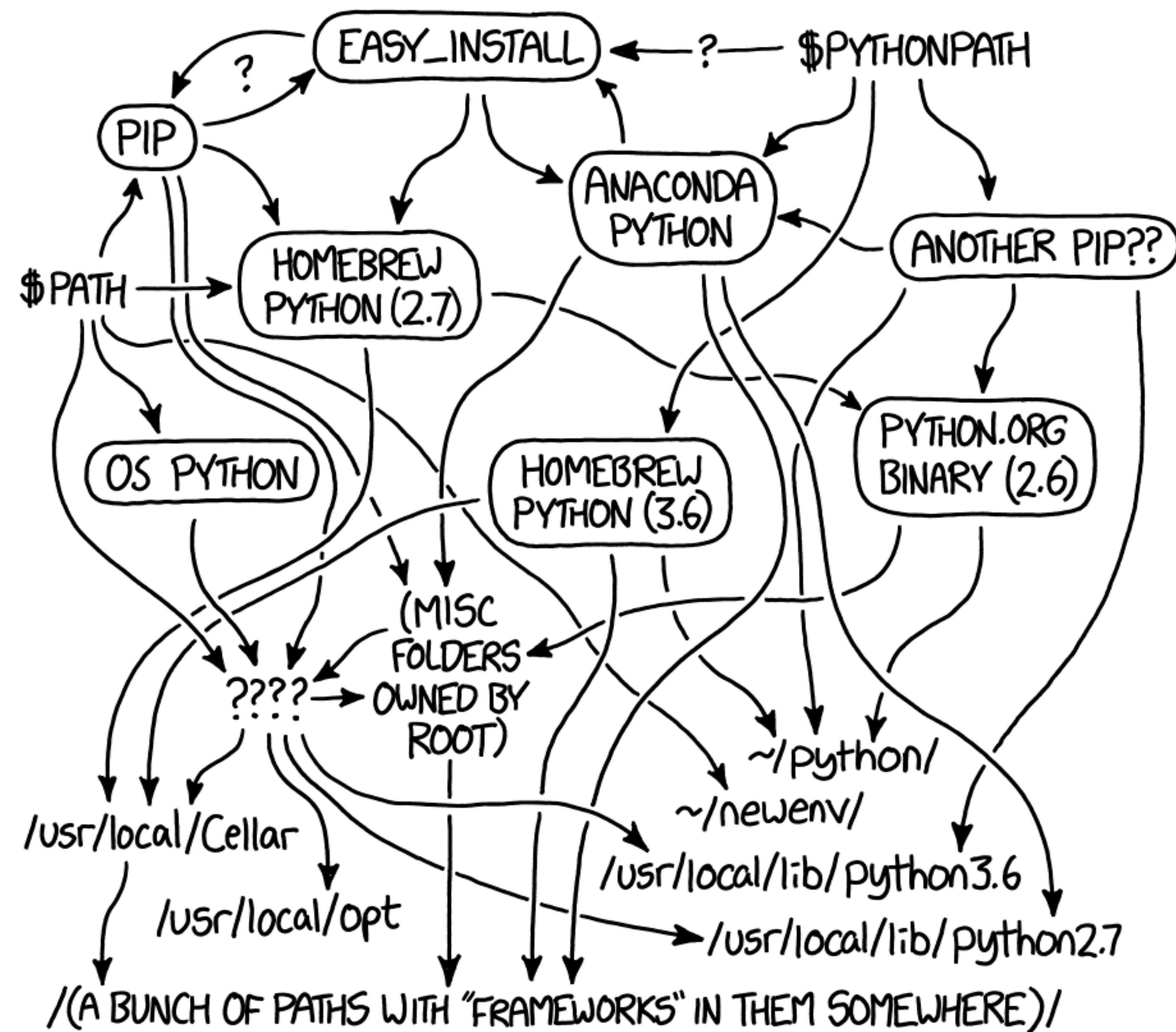


# New-ish Layer Used in AnimeGANv2

Group Norm

Implemented in MIL ops

# Python Environment



MY PYTHON ENVIRONMENT HAS BECOME SO DEGRADED  
THAT MY LAPTOP HAS BEEN DECLARED A SUPERFUND SITE.

The Python environmental protection agency wants to seal it in a cement chamber, with pictorial messages to future civilizations warning them about the danger of using sudo to install random Python packages.

Conda

# Conda Setup

```
> brew install miniforge
```

```
taojuice@M1-MBP ~ % b
```

# Conda Setup

```
> brew install miniforge  
> conda init zsh
```

```
taojuice@M1-MBP ~ % b
```

# Conda Setup

restart shell or Terminal.app

# Conda Env Setup

```
> conda create --name coreml python=3.9
```

```
Last login: Thu Dec  9 11:32:37 on ttys001  
(base) toojuice@M1-MBP ~ % con
```

# Conda Env Setup

```
> conda create --name coreml python=3.9  
> conda activate coreml
```

```
Last login: Thu Dec  9 11:32:37 on ttys001  
(base) toojuice@M1-MBP ~ % con
```

# Conda Env Setup

```
> conda create --name coreml python=3.9  
> conda activate coreml  
> conda install pytorch torchvision
```

```
Last login: Thu Dec  9 11:32:37 on ttys001  
(base) toojuice@M1-MBP ~ % con
```

# Conda Env Setup

```
> conda create --name coreml python=3.9  
> conda activate coreml  
> conda install pytorch torchvision  
> pip install coremltools
```

```
Last login: Thu Dec  9 11:32:37 on ttys001  
(base) toojuice@M1-MBP ~ % con
```

# Conda Env Setup

```
> conda create --name coreml python=3.9  
> conda activate coreml  
> conda install pytorch torchvision  
> pip install coremltools  
> conda install jupyterlab
```

\* optional but cool!

# Conda Deactivate

When you're done using the environment

```
> conda deactivate
```

# Other Handy Conda Commands

```
> conda list
```

# Other Handy Conda Commands

> conda update <package>

# Other Handy Conda Commands

> conda remove <package(s)>

# Other Handy Conda Commands

> conda env list

# Other Handy Conda Commands

```
> conda env remove --name <env_name>
```

# Conda Cheat Sheet

[https://docs.conda.io/projects/conda/en/4.6.0/\\_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf](https://docs.conda.io/projects/conda/en/4.6.0/_downloads/52a95608c49671267e40c689e0bc00ca/conda-cheatsheet.pdf)

# Check Python Path and Version

>

# Check Python Path and Version

```
> which python
```

# Check Python Path and Version

```
> which python  
/opt/homebrew/Caskroom/miniforge/base/envs/coreml/bin/  
python  
>
```

# Check Python Path and Version

```
> which python
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/coreml/bin/  
python
```

```
> python --version
```

# Check Python Path and Version

```
> which python
```

```
/opt/homebrew/Caskroom/miniforge/base/envs/coreml/bin/  
python
```

```
> python --version
```

```
Python 3.9.7
```

# Check Library Versions

```
>>> import torch
```

# Check Library Versions

```
>>> import torch  
>>> torch.__version__
```

# Check Library Versions

```
>>> import torch  
>>> torch.__version__  
'1.10.0'  
>>> import coremltools
```

# Check Library Versions

```
>>> import torch  
>>> torch.__version__  
'1.10.0'  
  
>>> import coremltools  
>>> coremltools.__version__
```

# Check Library Versions

```
>>> import torch  
>>> torch.__version__  
'1.10.0'  
  
>>> import coremltools  
>>> coremltools.__version__  
'5.1.0'
```

# Conversion Script

<https://github.com/yonomitt/Anime-fy-Yourself>



```
import torch  
import coremltools as ct
```

Python 3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:24:02)

[Clang 11.1.0 ] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>>

Python 3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:24:02)

[Clang 11.1.0 ] on darwin

Type "help", "copyright", "credits" or "license" for more information.

>>> import torch

Python 3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:24:02)

[Clang 11.1.0 ] on darwin

Type "help", "copyright", "credits" or "license" for more information.

```
>>> import torch
```

```
>>> import coremltools as ct
```

Python 3.9.7 | packaged by conda-forge | (default, Sep 29 2021, 19:24:02)

[Clang 11.1.0 ] on darwin

Type "help", "copyright", "credits" or "license" for more information.

```
>>> import torch
```

```
>>> import coremltools as ct
```

WARNING:root:Torch version 1.10.0 has not been tested with coremltools. You may run into unexpected errors. Torch 1.9.1 is the most recent version that has been tested.

```
>>>
```

```
import torch  
import coremltools as ct
```

```
import torch  
import coremltools as ct  
import sys
```

```
import torch
import coremltools as ct
import sys

# check for command line arguments
if len(sys.argv) < 2:
    print(f'USAGE: {sys.argv[0]} <image size>')
    sys.exit(-1)
```

```
import torch
import coremltools as ct
import sys

# check for command line arguments
if len(sys.argv) < 2:
    print(f'USAGE: {sys.argv[0]} <image size>')
    sys.exit(-1)

# store the desired image size
SIZE = int(sys.argv[1])
```





```
# load the AnimeGAN v2 model using the pretrained Face Portrait v2 weights
model = torch.hub.load("bryandlee/animegan2-pytorch:main", "generator",
                        pretrained="face_paint_512_v2")
model.eval()
```

```
# load the AnimeGAN v2 model using the pretrained Face Portrait v2 weights
model = torch.hub.load("bryandlee/animegan2-pytorch:main", "generator",
                        pretrained="face_paint_512_v2")
model.eval()
```

```
# create a dummy input for the model trace
# dimensions are (batch size, channels (RGB), width, height)
dummy_in = torch.rand(1, 3, SIZE, SIZE)
```

```
# load the AnimeGAN v2 model using the pretrained Face Portrait v2 weights
model = torch.hub.load("bryandlee/animegan2-pytorch:main", "generator",
                        pretrained="face_paint_512_v2")
model.eval()
```

```
# create a dummy input for the model trace
# dimensions are (batch size, channels (RGB), width, height)
dummy_in = torch.rand(1, 3, SIZE, SIZE)
```

```
# run the model's forward method and record the tensor operations
traced_model = torch.jit.trace(model, dummy_in)
```

```
# load the AnimeGAN v2 model using the pretrained Face Portrait v2 weights
model = torch.hub.load("bryandlee/animegan2-pytorch:main", "generator",
                        pretrained="face_paint_512_v2")
model.eval()
```

```
# create a dummy input for the model trace
# dimensions are (batch size, channels (RGB), width, height)
dummy_in = torch.rand(1, 3, SIZE, SIZE)
```

```
# run the model's forward method and record the tensor operations
traced_model = torch.jit.trace(model, dummy_in)
```

```
# save the traced model (TorchScript) to a file
traced_model.save("animegan_v2.pt")
```



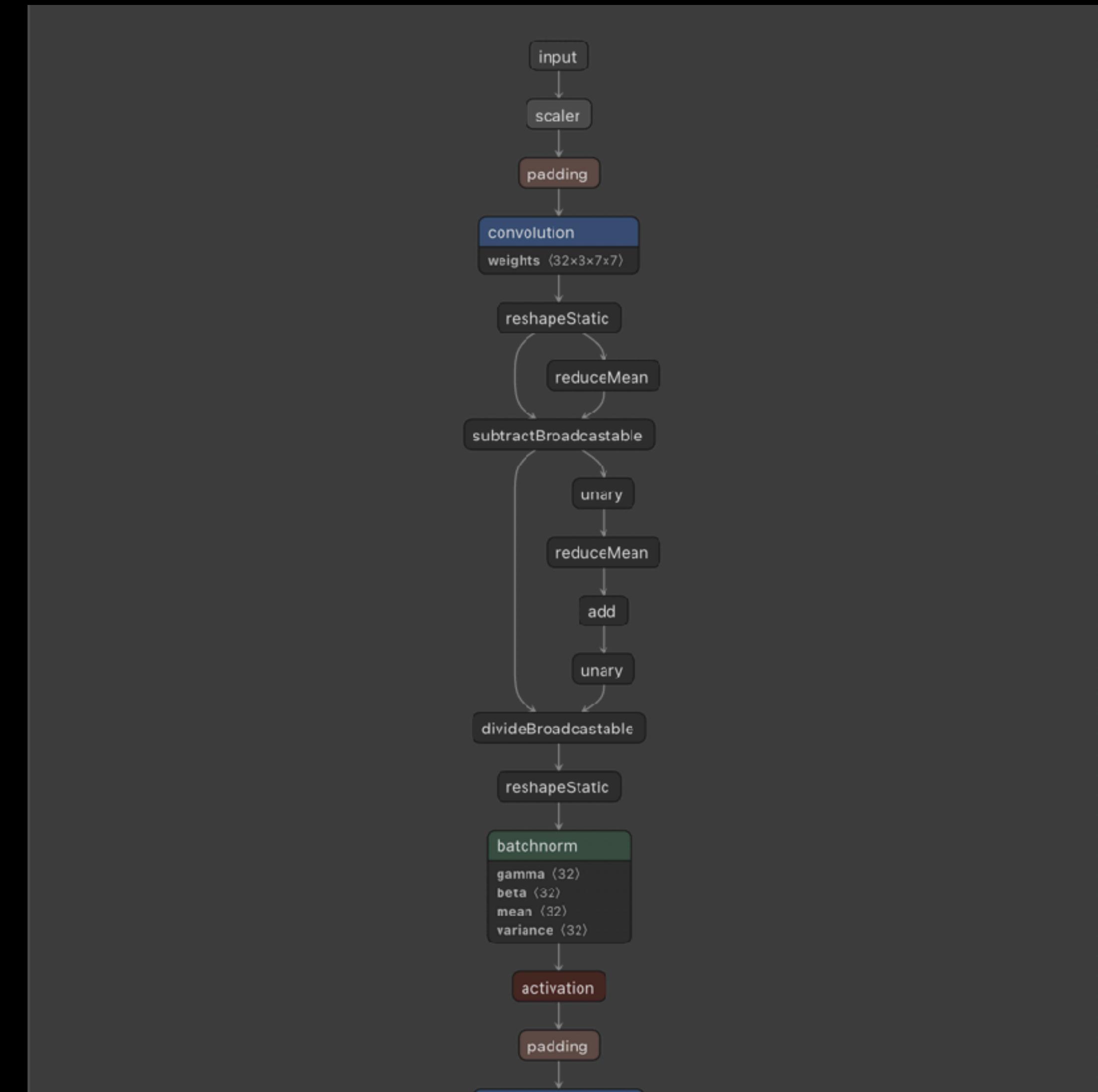
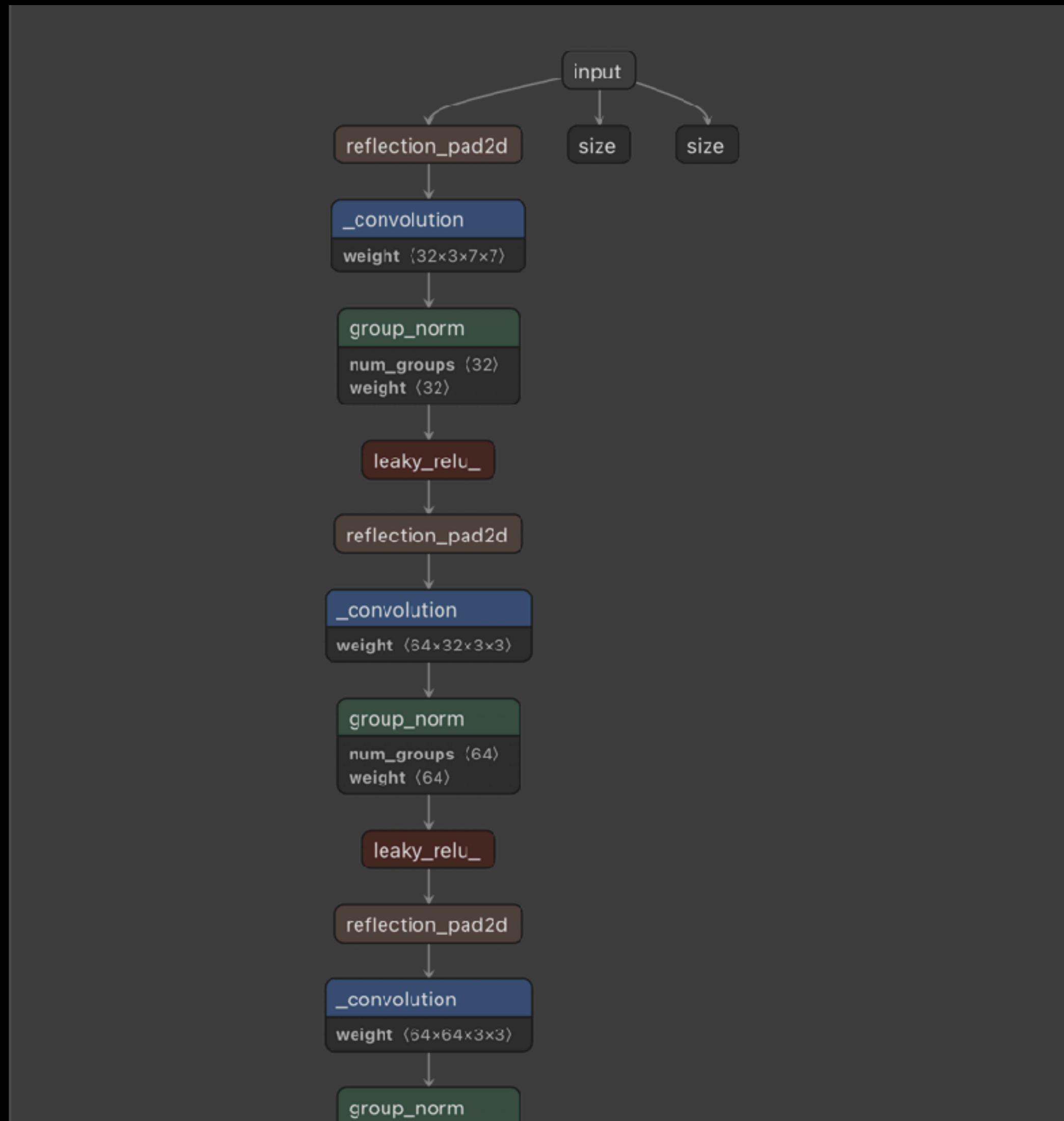
```
# define the CoreML inputs  
input = ct.TensorType(name="input", shape=dummy_in.shape)
```

```
# define the CoreML inputs  
  
input = ct.TensorType(name="input", shape=dummy_in.shape)  
  
  
# convert the TorchScript to an MLModel  
  
mlmodel = ct.convert("animegan_v2.pt", inputs=[input])
```

```
# define the CoreML inputs  
  
input = ct.TensorType(name="input", shape=dummy_in.shape)  
  
# convert the TorchScript to an MLModel  
  
mlmodel = ct.convert("animegan_v2.pt", inputs=[input])  
  
# save the model  
  
mlmodel.save(f"AnimeGANv2_{SIZE}.mlmodel")
```

**THIS SLIDE INTENTIONALLY LEFT BLANK**

# Netron

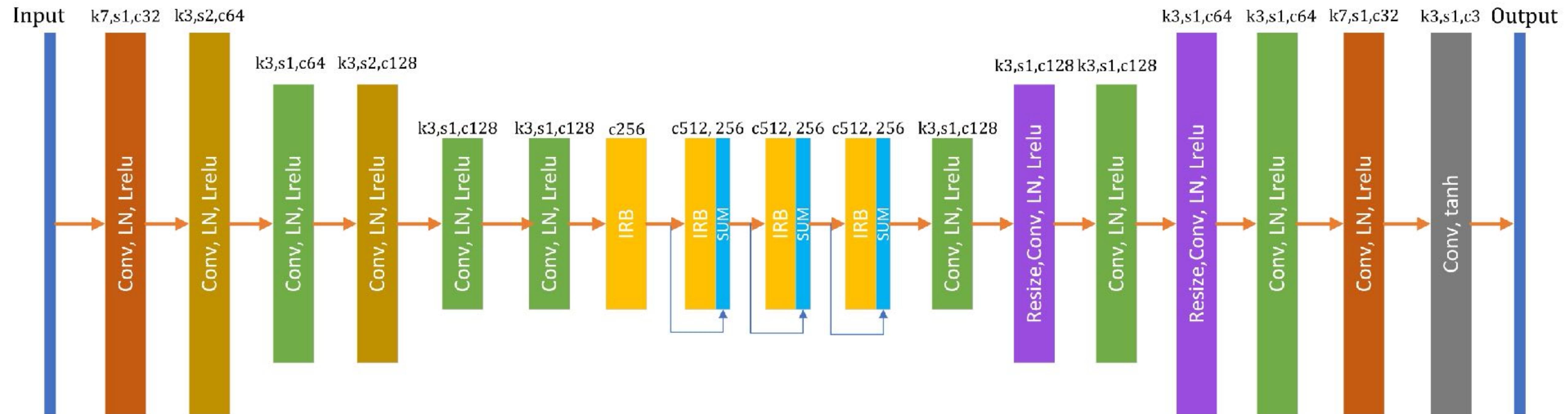


# Trace vs Script

Trace uses a sample input and traces which operations are performed on the data

Script will follow all branches of control and conditional operations

# AnimeGAN v2



# Fixing the Input Type

```
# define the CoreML inputs  
  
input = ct.TensorType(name="input", shape=dummy_in.shape)  
  
# convert the TorchScript to an MLModel  
  
mlmodel = ct.convert("animegan_v2.pt", inputs=[input])  
  
# save the model  
  
mlmodel.save(f"AnimeGANv2_{SIZE}.mlmodel")
```

```
# define the CoreML inputs

input = ct.TensorType(name="input", shape=dummy_in.shape)

# convert the TorchScript to an MLModel

mlmodel = ct.convert("animagan_v2.pt", inputs=[input])

# save the model

mlmodel.save(f"AnimeGANv2_{SIZE}.mlmodel")
```

```
# define the CoreML inputs

input = ct.ImageType(name="input", shape=dummy_in.shape, scale=1./127.5,
                     bias=[-1.0, -1.0, -1.0], color_layout='RGB', channel_first=True)

# convert the TorchScript to an MLModel

mlmodel = ct.convert("animegan_v2.pt", inputs=[input])

# save the model

mlmodel.save(f"AnimeGANv2_{SIZE}.mlmodel")
```

**THIS SLIDE INTENTIONALLY LEFT BLANK**

**Can we do better?**

# Model Editing

# Changing the Output Type



```
import coremltools.proto.FeatureTypes_pb2 as ft  
from coremltools import utils
```

```
import coremltools.proto.FeatureTypes_pb2 as ft
from coremltools import utils

# get a deep copy of the model's protobuf specification
spec = mlmodel.get_spec()
```

```
import coremltools.proto.FeatureTypes_pb2 as ft
from coremltools import utils

# get a deep copy of the model's protobuf specification
spec = mlmodel.get_spec()

# make the output an image type
output = spec.description.output[0]
```

```
import coremltools.proto.FeatureTypes_pb2 as ft
from coremltools import utils

# get a deep copy of the model's protobuf specification
spec = mlmodel.get_spec()

# make the output an image type
output = spec.description.output[0]
output.type.imageType.colorSpace = ft.ImageFeatureType.RGB
```

```
import coremltools.proto.FeatureTypes_pb2 as ft
from coremltools import utils

# get a deep copy of the model's protobuf specification
spec = mlmodel.get_spec()

# make the output an image type
output = spec.description.output[0]
output.type.imageType.colorSpace = ft.ImageFeatureType.RGB
output.type.imageType.width = SIZE
```

```
import coremltools.proto.FeatureTypes_pb2 as ft
from coremltools import utils

# get a deep copy of the model's protobuf specification
spec = mlmodel.get_spec()

# make the output an image type
output = spec.description.output[0]
output.type.imageType.colorSpace = ft.ImageFeatureType.RGB
output.type.imageType.width = SIZE
output.type.imageType.height = SIZE
```

```
import coremltools.proto.FeatureTypes_pb2 as ft
from coremltools import utils

# get a deep copy of the model's protobuf specification
spec = mlmodel.get_spec()

# make the output an image type
output = spec.description.output[0]
output.type.imageType.colorSpace = ft.ImageFeatureType.RGB
output.type.imageType.width = SIZE
output.type.imageType.height = SIZE

# save the CoreML model
utils.save_spec(spec, f"AnimeGANv2_{SIZE}.mlmodel")
```

**THIS SLIDE INTENTIONALLY LEFT BLANK**

# Scaling the Output



```
# get the neural network
```

```
nn = get_nn(spec)
```

```
# get the neural network
nn = get_nn(spec)

# add a new layer to the end of the neural network
unscaled_output_layer = nn.layers[-1]
```

```
# get the neural network
nn = get_nn(spec)

# add a new layer to the end of the neural network
unscaled_output_layer = nn.layers[-1]
scale_layer = nn.layers.add()
```

```
# get the neural network
nn = get_nn(spec)

# add a new layer to the end of the neural network
unscaled_output_layer = nn.layers[-1]
scale_layer = nn.layers.add()

# set it's name
scale_layer.name = "scale_output"
```



```
# make the new layer a linear activation function,  
# which scales the output from [-1, 1] to [0, 255]  
#  $y = \alpha * x + \beta$   
scale_layer.activation.linear.alpha = 127.5  
scale_layer.activation.linear.beta = 127.5
```

```
# make the new layer a linear activation function,  
# which scales the output from [-1, 1] to [0, 255]  
#  $y = \alpha * x + \beta$   
scale_layer.activation.linear.alpha = 127.5  
scale_layer.activation.linear.beta = 127.5  
  
# connect the new layer to the output  
scale_layer.output.append(unscaled_output_layer.output[0])
```

```
# make the new layer a linear activation function,  
# which scales the output from [-1, 1] to [0, 255]  
#  $y = \alpha * x + \beta$   
scale_layer.activation.linear.alpha = 127.5  
scale_layer.activation.linear.beta = 127.5  
  
# connect the new layer to the output  
scale_layer.output.append(unscaled_output_layer.output[0])  
  
# connect the old last layer's output to the new scale layer's input  
unscaled_output_layer.output[0] = "scale_layer_input"  
scale_layer.input.append(unscaled_output_layer.output[0])
```



```
# save the CoreML model  
utils.save_spec(spec, f"AnimeGANv2_{SIZE}.mlmodel")
```

**THIS SLIDE INTENTIONALLY LEFT BLANK**

# Renaming the Output



```
# rename the output  
output.name = "output"
```

```
# rename the output  
output.name = "output"  
output_layer = nn.layers[-1]
```

```
# rename the output  
output.name = "output"  
output_layer = nn.layers[-1]  
output_layer.output[0] = output.name
```

```
# rename the output  
output.name = "output"  
output_layer = nn.layers[-1]  
output_layer.output[0] = output.name  
  
# save the CoreML model  
utils.save_spec(spec, f"AnimeGANv2_{SIZE}.mlmodel")
```

**THIS SLIDE INTENTIONALLY LEFT BLANK**

# Quantizing the Parameters



```
from coremltools.models.neural_network import quantization_utils
```

```
from coremltools.models.neural_network import quantization_utils
```

```
...
```

```
from coremltools.models.neural_network import quantization_utils  
...  
  
# convert the TorchScript to an MLModel  
mlmodel = ct.convert("animegan_v2.pt", inputs=[input])
```

```
from coremltools.models.neural_network import quantization_utils  
...  
  
# convert the TorchScript to an MLModel  
mlmodel = ct.convert("animegan_v2.pt", inputs=[input])  
  
  
# convert the weights from fp32 -> fp16  
mlmodel = quantization_utils.quantize_weights(mlmodel, nbits=16)
```

**THIS SLIDE INTENTIONALLY LEFT BLANK**

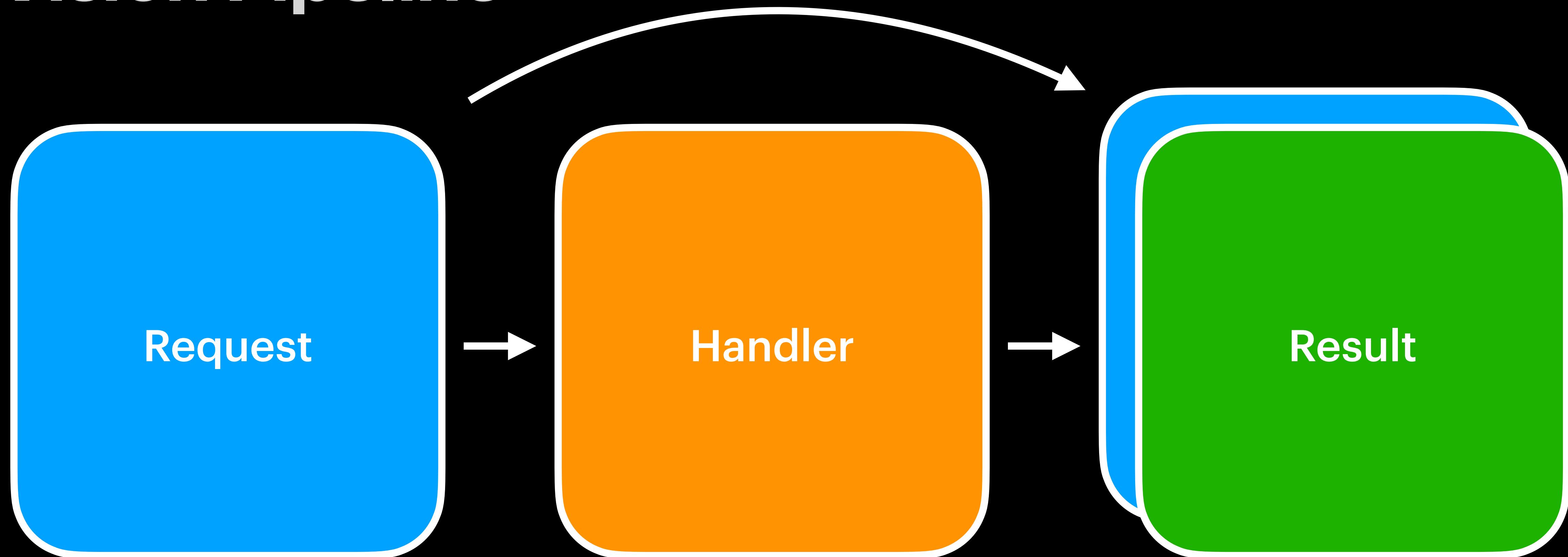
# Comparing Outputs



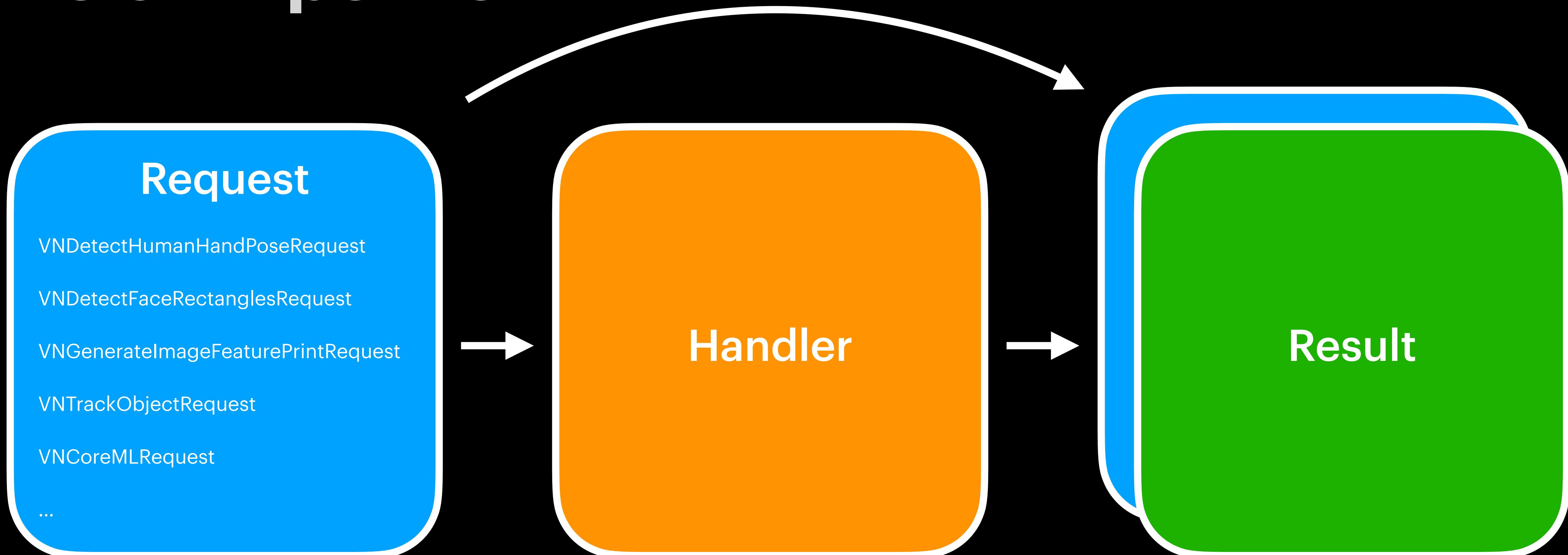
# iOS Integration

# Vision Pipeline

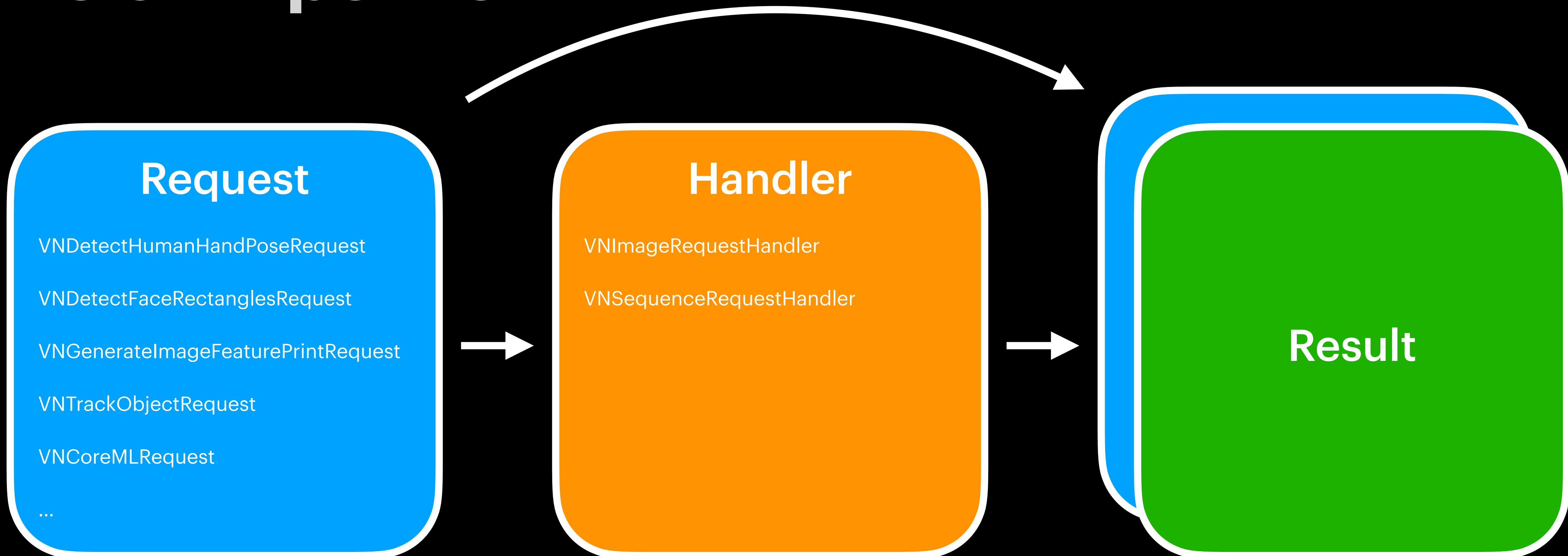
# Vision Pipeline



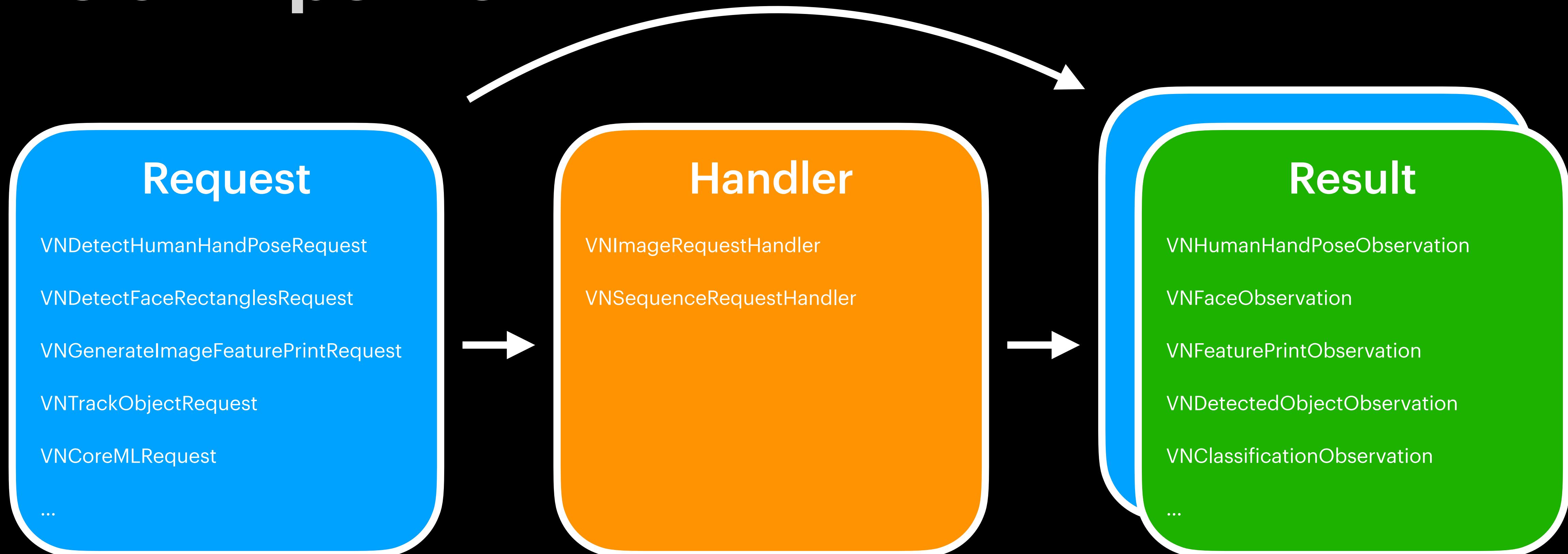
# Vision Pipeline



# Vision Pipeline



# Vision Pipeline





```
private let request: VNRequest = {
```

```
}()
```

```
private let request: VNRequest = {  
    let configuration = MLModelConfiguration()  
  
    }○
```

```
private let request: VNRequest = {  
    let configuration = MLModelConfiguration()  
    configuration.computeUnits = .all  
    configuration.allowLowPrecisionAccumulationOnGPU = true  
}  
○
```

```
private let request: VNRequest = {  
    let configuration = MLModelConfiguration()  
    configuration.computeUnits = .all  
    configuration.allowLowPrecisionAccumulationOnGPU = true  
  
    let model = try! AnimeGANv2_512(configuration: configuration).model  
}
```

```
private let request: VNRequest = {  
    let configuration = MLModelConfiguration()  
    configuration.computeUnits = .all  
    configuration.allowLowPrecisionAccumulationOnGPU = true  
  
    let model = try! AnimeGANv2_512(configuration: configuration).model  
  
    let req = VNCoreMLRequest(model: try! VNCoreMLModel(for: model))  
}
```

```
private let request: VNRequest = {  
    let configuration = MLModelConfiguration()  
    configuration.computeUnits = .all  
    configuration.allowLowPrecisionAccumulationOnGPU = true  
  
    let model = try! AnimeGANv2_512(configuration: configuration).model  
  
    let req = VNCoreMLRequest(model: try! VNCoreMLModel(for: model))  
    req.imageCropAndScaleOption = .centerCrop  
}
```

```
private let request: VNRequest = {  
    let configuration = MLModelConfiguration()  
    configuration.computeUnits = .all  
    configuration.allowLowPrecisionAccumulationOnGPU = true  
  
    let model = try! AnimeGANv2_512(configuration: configuration).model  
  
    let req = VNCoreMLRequest(model: try! VNCoreMLModel(for: model))  
    req.imageCropAndScaleOption = .centerCrop  
  
    return req  
}
```



```
let handler = VNImageRequestHandler(cvPixelBuffer: image, options: [:])
```

```
let handler = VNImageRequestHandler(cvPixelBuffer: image, options: [:])
```

```
let animeRequest = try! handler.perform([request])
```

```
let handler = VNImageRequestHandler(cvPixelBuffer: image, options: [:])
```

```
let animeRequest = try! handler.perform([request])
```

```
guard let results = animeRequest.results as? [VNPixelBufferObservation],
```

```
let result = results.first else {
```

```
    return nil
```

```
}
```

```
let handler = VNImageRequestHandler(cvPixelBuffer: image, options: [:])
```

```
let animeRequest = try! handler.perform([request])
```

```
guard let results = animeRequest.results as? [VNPixelBufferObservation],
```

```
let result = results.first else {
```

```
    return nil
```

```
}
```

```
return CIImage(cvPixelBuffer: result.pixelBuffer)
```

**THIS SLIDE INTENTIONALLY LEFT BLANK**

<https://github.com/yonomitt/Anime-fy-Yourself>

# Thank You

@yonomitt

<https://github.com/yonomitt/Anime-fy-Yourself>