AXIS COMMUNICATIONS

TensorFlow 2.0

Training - Overview

> Introduction
> Tensorflow Basics
> Transfer Learning

Part 1

Introduction

Introduction - overview

> Prerequisites (HW/SW)

> Colab: A jupyter notebook for exercises

> Brief recap of neural networks

> Why Tensorflow 2.0?

Prerequisites

Prerequisites – in general

- PC with Nvidia GPU
- Ubuntu 16.04/18.04 LTS (or Debian)
- Using pre-built Tensorflow binary
- CUDA X.X
- CuDNN X.X
- Python 3.X
- Numpy, matplotlib etc.

Prerequisites – this course

- **A browser** (tested browser: Google Chrome)
- **Colab runtime** (via exercise links)
- **A Google account**
  - dummy account available if needed
  - Code saved in Google Drive

Colab

Working with the colab exercises

- > Click on the provided link to the current colab exercise
- > Select ”File -> Save a copy in Drive…”
  - **This is VERY important, so we do not all edit the same file!**
- > Select ”Runtime -> Select runtime type”
  - Make sure ”Hardware accelerator” is set to ”GPU”
- > To run the exercise, select each code cell and either:
  - press ”Shift + Return”
  - Or click on the play button to the left on the cell.
- > Edit cells and re-run cells as needed.
- > Sometimes, after too many code changes, you might want to
  - Select ”Runtime -> Reset all runtimes…” and re-run the whole exercise.

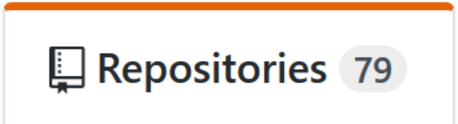
Why Tensorflow 2.0?

Why do we need a remake of Tensorflow at all?

- Too much boilerplate code (environment specific)
- Graph concept unnecessarily complicated
- Hard to debug when things go wrong
- Not enough flexibility

- Tensorflow 2.0 much more ”Pythonic” and intuitive than 1.X
- Size became prohibitively large: Now modularized into Core TF and other repos
- Tensorflow Extended (TFX): End-to-end platform for deploying production ML pipelines
- …and much more!

https://github.com/tensorflow



model-optimization

compression

model-analysis

examples

tfx

agents

datasets

graphics

deepmath

tensorflow

models

federated

docs

networking

addons

privacy

gan

probability

autograph

hub

benchmarks

serving

community

tensorboard

swift-apis

io

estimator

swift-bindings

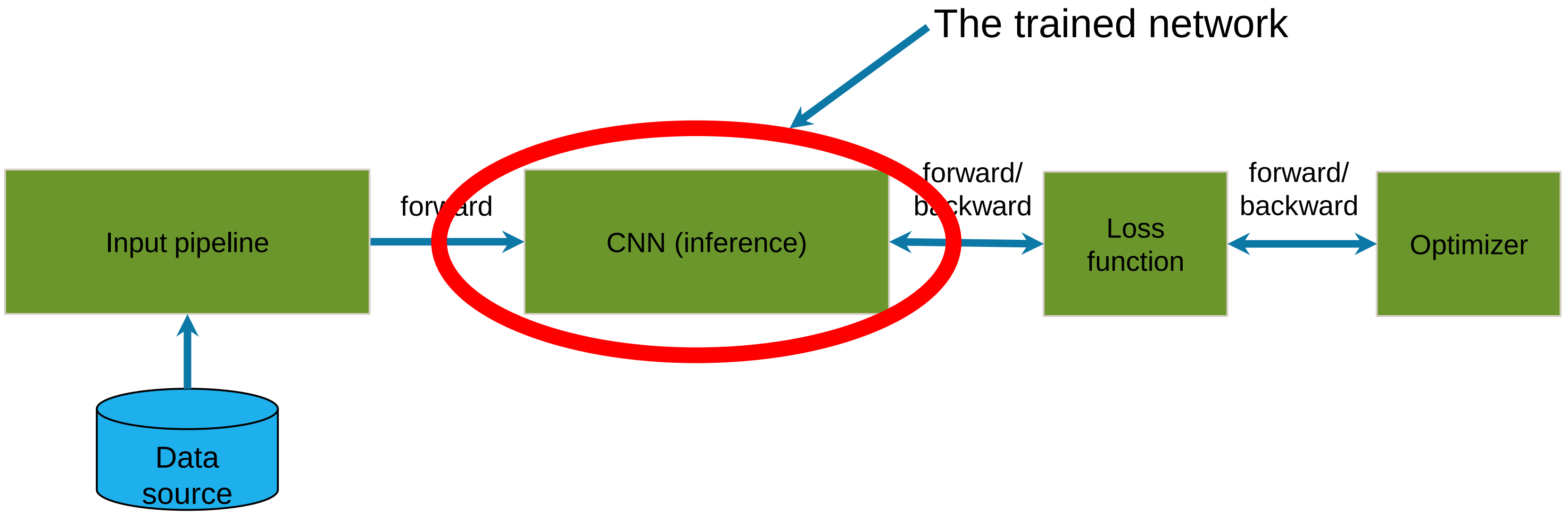
ecosystem

transform

tensorrt

metadata

Overview: Training and deployment



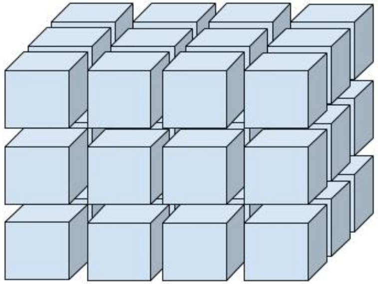
Part 2

Tensorflow basics

Tensorflow basics - overview

> Tensors in Tensorflow
> Input pipeline (Datasets)
> Dataset transformations
> CNN model definition (Keras)
> Training loop
> Learning rate schemes
> Tensorboard

Tensors in Tensorflow



Input/output tensors (activations) have the format

`[batch, height, width, channels]`

Batching – Synergy from processing several images

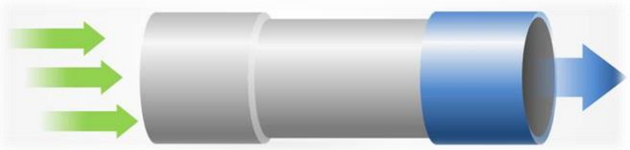
Tensors and numpy arrays can be used interchangeably*

- NumPy operations accept tf.Tensor arguments.
- TensorFlow math operations convert NumPy arrays to tf.Tensor objects.
- The tf.Tensor.numpy() method returns the object's value as a NumPy ndarray.

*For more details, see https://www.tensorflow.org/beta/guide/eager

Creating an input pipeline

Source
(e.g. .jpeg's on file)



Transformed formatted source
(e.g. tf.Tensor object)

The tf.Data.Dataset provides the needed input pipeline functionality

Define input source

tf.data.Dataset.from_generator()
tf.data.Dataset.from_tensors()
tf.data.Dataset.from_tensor_slices()
tf.data.Dataset.zip()
tf.data.experimental.CsvDataset()
tf.data.TFRecordDataset

Configure pipeline and transform data

tf.data.Dataset.repeat()
tf.data.Dataset.shuffle()
tf.data.Dataset.batch()
tf.data.Dataset.map()
tf.data.Dataset.cache()

All these return new Dataset objects,
And thus supports stacking of operations

Further reading: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/TFRecordDataset

Pre-packaged datasets from Tensorflow

Install (and import) the tensorflow datasets repository

```
pip3 install -U tensorflow_datasets
```

List available datasets

```
tfds.list_builders()
```

Load the desired dataset into dataset objects and metadata dicts

```
dataset, metadata = tfds.load(‘<dataset_name>', as_supervised=True, with_info=True)
```

Metadata object has easily accessible info about the training dataset

```
metadata.splits['train'].num_examples
metadata.splits['test'].num_examples
```

Access the respective datasets by name

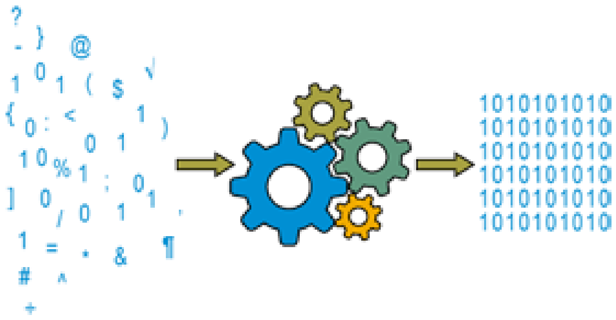
```
train_dataset = dataset['train']
```

For the high-level APIs for the first example this is the minimum needed to train the CNN…
…in practice one also wants some additional preprocessing options…

https://www.tensorflow.org/datasets

Input data transformations – the map function

- On-the-fly processing of input data
  - Input format requirements
  - Data augmentation
    - Extending the dataset variation
    - Domain adaption
    - Dataset regularization



Example: Normalize input images for dataset object `my_dataset` from range [0, 255] to [0, 1]

```
def normalize(images, labels):
    images = tf.cast(images, tf.float32)
    images /= 255
    return images, labels
```

```
my_dataset = my_dataset.map(normalize)
```

New iterable dataset object
from old dataset object

Configure the input pipeline

Choose a batch size (usually as large as your GPU memory allows)

```
BATCH_SIZE = 32
```

Configure the training pipeline

```
train_dataset = train_dataset.repeat()
                             .shuffle(num_train_examples)
                             .batch(BATCH_SIZE)
```

No need to shuffle the test dataset, or run more than one full iteration (epoch)

```
test_dataset = test_dataset.batch(BATCH_SIZE)
```

Note that the order of operations matter

Typically: repeat before shuffle, and batch at the end

Keras - a High Level API for Defining Neural Networks

- Keras is an **API standard** for defining and training machine learning models.
- Keras is not tied to a specific implementation*
- A reference implementation of Keras is maintained as an independent open source project, which you can find at www.keras.io.
- This project is independent of TensorFlow, and has an active community of contributors and users.
- TensorFlow **includes an implementation of the Keras API** (in the tf.keras module) with TensorFlow-specific enhancements.

Enhancements:

tf.data
Distribution strategies
Exporting models in SavedModel format
Deployment on Tensorflow Lite
etc.
Blogpost: Standardizing on Keras

*The Keras API has implementations for TensorFlow, MXNet, TypeScript, JavaScript, CNTK, Theano, PlaidML, Scala, CoreML and more

Keras - a High Level API for Defining Neural Networks

- 3 ways to specify an architecture
  - Sequential
  - Functional
  - Subclassing
- Ok to mix and match…

We will use the simplest sequential API for the first CNN example

The API documentation – Obs: 2.0!

https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras
https://keras.io/ (this doc is often more complete than the Tensorflow version)

tf.keras.layers.conv2D as conv2D

Only two mandatory parameters

```
layer_instance = Conv2D(64, (3, 3))
```

Constructor returns a layer instance

Output filters

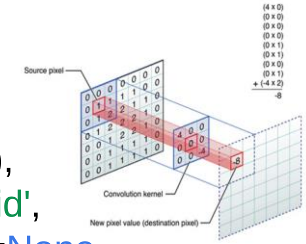
Filter kernel size

Typically you want to specify (at least) a few more

```
activation='relu'
padding='same' or 'valid'
name='block1_conv2'
```

Params for base classes (e.g. input_shape)

```
__init__(
    filters,
    kernel_size,
    strides=(1, 1),
    padding='valid',
    data_format=None,
    dilation_rate=(1, 1),
    activation=None,
    use_bias=True,
    kernel_initializer='glorot_uniform',
    bias_initializer='zeros',
    kernel_regularizer=None,
    bias_regularizer=None,
    activity_regularizer=None,
    kernel_constraint=None,
    bias_constraint=None,
    **kwargs
)
```



Complete list of parameters for each layer is found at:
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers

Many alternative names for each layer

Example: Conv2D

Aliases:

- Class `tf.compat.v1.keras.layers.Conv2D`
- Class `tf.compat.v1.keras.layers.Convolution2D`
- Class `tf.compat.v2.keras.layers.Conv2D`
- Class `tf.compat.v2.keras.layers.Convolution2D`
- Class `tf.keras.layers.Conv2D`
- Class `tf.keras.layers.Convolution2D`

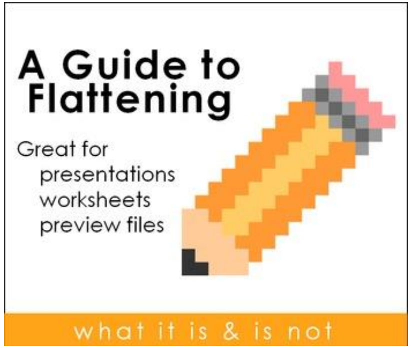
Usually, both long and abbreviated names exist

tf.keras.layers.Flatten as Flatten

No mandatory parameters
`Flatten()`

Still good practice to name the layer: **`name='flatten'`**

Transforms an image from a **`[B,H,W,C]`** array of pixels to a **`[B,L]`** array of pixels.



- Pure transformation layer
- No learnable parameters - only reformats data
- Batch size is not specified, but is inferred from the input image

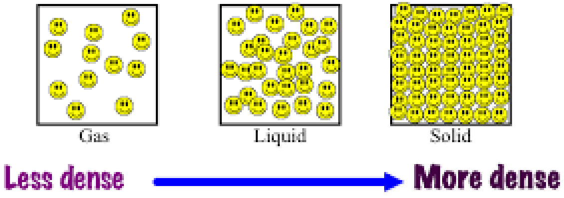
Actually, 2 tensor formats supported

`[B,H,W,C]` = channels last

Default

`[B,C,H,W]` = channels first

tf.keras.layers.Dense as Dense



Only one mandatory parameter

`Dense(4096)`

`Number of outputs`
`units = 4096`

Typically you also want to specify 2 more

`activation='relu'` or **`'softmax'`** (default is **`None`**)
`name='fc1'`

Typical usage assumes flattened input (using **`Flatten()`**)

- Input shape **`(batch_size, input_dim)`**
- Output shape: **`(batch_size, units)`**

tf.keras.layers.MaxPool as MaxPool

No mandatory parameters

`MaxPool2D()`

Default `pool_size` is (2, 2)

Typically you also want to specify some more

`pool_size=(vertical, horizontal)` or `value`
`strides=(vertical, horizontal)` or `value` or `None`
`name='block5_pool'`

Default is `None`, and then `strides = pool_size` will be used

Related layers

`AvgPool2D()/AveragePooling2D()`

`GlobalMaxPool2D()/GlobalMaxPooling2D()`
`GlobalAvgPool2D()/GlobalAveragePooling2D()`

Global poolings corresponds to a pool_size equal to the height and width of the entire input tensor (but do not require it to be known in advance)

tf.keras.layers.Dropout as Dropout

One mandatory parameter

```
Dropout(rate=0.5)
```

Float between 0 and 1.
Fraction of the input units to set to zero at training time

- First example of layer with different behaviour at training and evaluation time.
- At evaluation time the layer does nothing.

Module: tf.keras.models

Classes

`class Model` : `Model` groups layers into an object with training and inference features.

`class Sequential` : Linear stack of layers.

Model creation

```
model = tf.keras.Sequential([layer1, …, layerN])
```

First layer needs input shape specification (if internal layers need to know this)

A simple Neural network

```
model = tf.keras.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28, 1)),
    tf.keras.layers.Dense(128, activation=tf.nn.relu),
    tf.keras.layers.Dense(10,  activation=tf.nn.softmax)
])
```


Further reading at https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/models

Preparing the model for training

Compile the model - configures the model for training.

Only one mandatory parameter

```
model.compile(optimizer='adam’)
```

Set optimizer by string, or by class instance

However, a training objective (a *loss function*) is needed for any meaningful training to occur

```
loss='sparse_categorical_crossentropy'
```

Set loss function by string or by passing an explicit loss function (this one is the single class classifier loss)

Typically one also sets evaluation metric(s)

```
metrics=['accuracy']
```

For more choices of compile options, see the compile() function at https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model

Training a compiled model (High level Keras API)

Run the training loop a given number of epochs (default = 1 epoch)

`model.fit()`

Actually NO mandatory parameters(!)

However, you must at least specify the data to train on
For anything to happen
`x=train_dataset`

In this example we use a tf.data dataset, which contains both the training data and the target output used by the loss function.

Typical additional parameters are

```
epochs=5
steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE)

validation_data=validation_dataset
validation_freq=1
```

For a complete set of options, see the fit() function at
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model

Testing a trained model (High level Keras API)

This function returns the loss value & metrics values for the model in test mode.

```
model.evaluate()
```

Again NO mandatory parameters(!)

However, you must at least specify the data to evaluate on
For anything to happen…

```
x=test_dataset
```

Again, we use a tf.data dataset, which contains both the training data and the target output used by the loss function and evaluation metric.

For a complete set of options, see the evaluate() function at
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/Model

Exercise 2: ”First Tensorflow Training Example”

Exercise 2:
https://colab.research.google.com/drive/1f8s8L4dGFP4nHz415iH5oTNh3TRCmC7h

Answers to exercise 2:
https://colab.research.google.com/drive/12hoctT83TjimvNnwtq9O6BkNf5eKdqRi

Learning rate schedules

Allows to vary the learning rate over time during training

```
lr_schedule = ExponentialDecay(initial_learning_rate=0.001,
              decay_steps=steps_per_epoch,
              decay_rate=0.4)
```

Optional additional parameters

```
staircase=True
```

In this example we also used

```
steps_per_epoch=math.ceil(num_train_examples/BATCH_SIZE)
```

Documentation and more schedules at
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/optimizers/schedules

Learning rate schedules

Use explicit class instance since we want non-default initialization

```
optimizer = tf.keras.optimizers.Adam(learning_rate=lr_schedule)
```

Use explicit class instance instead of name string when compiling for training

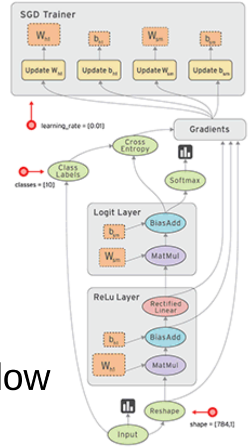
```
model.compile(optimizer=optimizer,
              loss='sparse_categorical_crossentropy',
              metrics=['accuracy'])
```

More optimizers found at
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/optimizers

Tensorboard – visualizing of training metrics

In terms of code, almost trivial

```
tensorboard_callback = tf.keras.callbacks.TensorBoard()
```



Optional additional parameters

```
histogram_freq=1
log_dir="logs/fit/"
        + datetime.datetime.now()
        .strftime("%Y%m%d-%H%M%S")
```

Computation of activation and weight histograms

Good practice: timestamped subdirectory to allow easy selection of different training runs

Callback is added as parameter to the fit() command

```
model.fit(…, callbacks=[tensorboard_callback])
```

Visualize from terminal with

```
tensorboard –logdir logs/fit
```

Point to the root folder to get all subfolders visualized (e.g. for multiple runs)

More about tensorboard callbacks found at
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/callbacks/TensorBoard

Tensorboard – running in colab

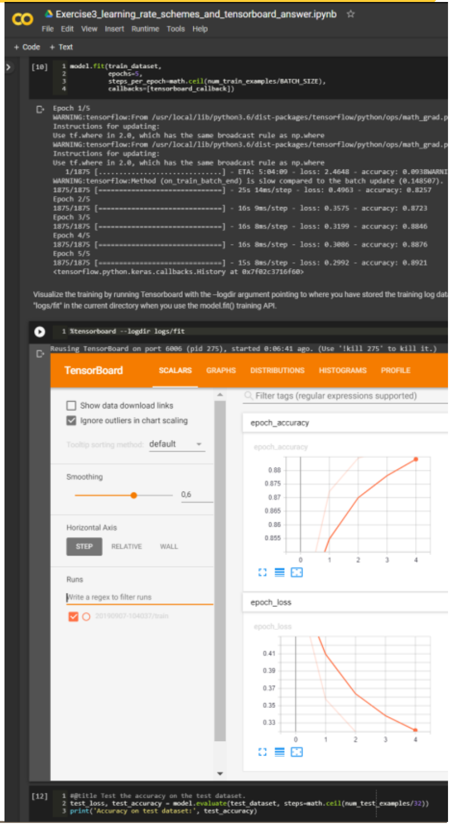
Load the Tensorboard notebook extension

`%load_ext tensorboard`

Visualization is not a terminal command anymore

`%tensorboard –logdir logs/fit`

Note: Tensorflow update bug (no dynamic updates)



AXIS COMMUNICATIONS

Exercise 3: ”Tensorboard and Learning rate Schedules”

Exercise 3:
https://colab.research.google.com/drive/107oomMnOoNL0RZYq09dUJCWe3sK026yl

Answers to exercise 3:
https://colab.research.google.com/drive/1JyFqekeoXDNVRAZaDSx7pvLWXtA5aX-5

Where to look for things…

- Starting point: https://www.tensorflow.org/
- Learn -> Tensorflow : ”The core library”
  - https://www.tensorflow.org/overview
- Learn -> Tensorflow -> TF 2.0 Beta :
  - https://www.tensorflow.org/beta
- API -> r2.0 (beta)
  - https://www.tensorflow.org/versions/r2.0/api_docs/python/tf
- Keras:
  - https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras
  - https://keras.io/ (Independent API spec)
- Lightweight free course (under construction)
  - https://eu.udacity.com/course/intro-to-tensorflow-for-deep-learning--ud187

Outlook – where do we go from here?

Data

Dataset
Create your own dataset
and input pipeline

Transfer learning:
enabling you to do well
with thousands of images
instead of millions

Flexibility

Customization
Customize the training loop,
Get more detailed control

Advanced CNN architectures
Going beyond simple ”stacked layers”

Dynamic training
Batch mining and more
flexible learning rate schemes

Efficiency

Containerized development
The modern way
for efficient development

Part 3

Dataset Management

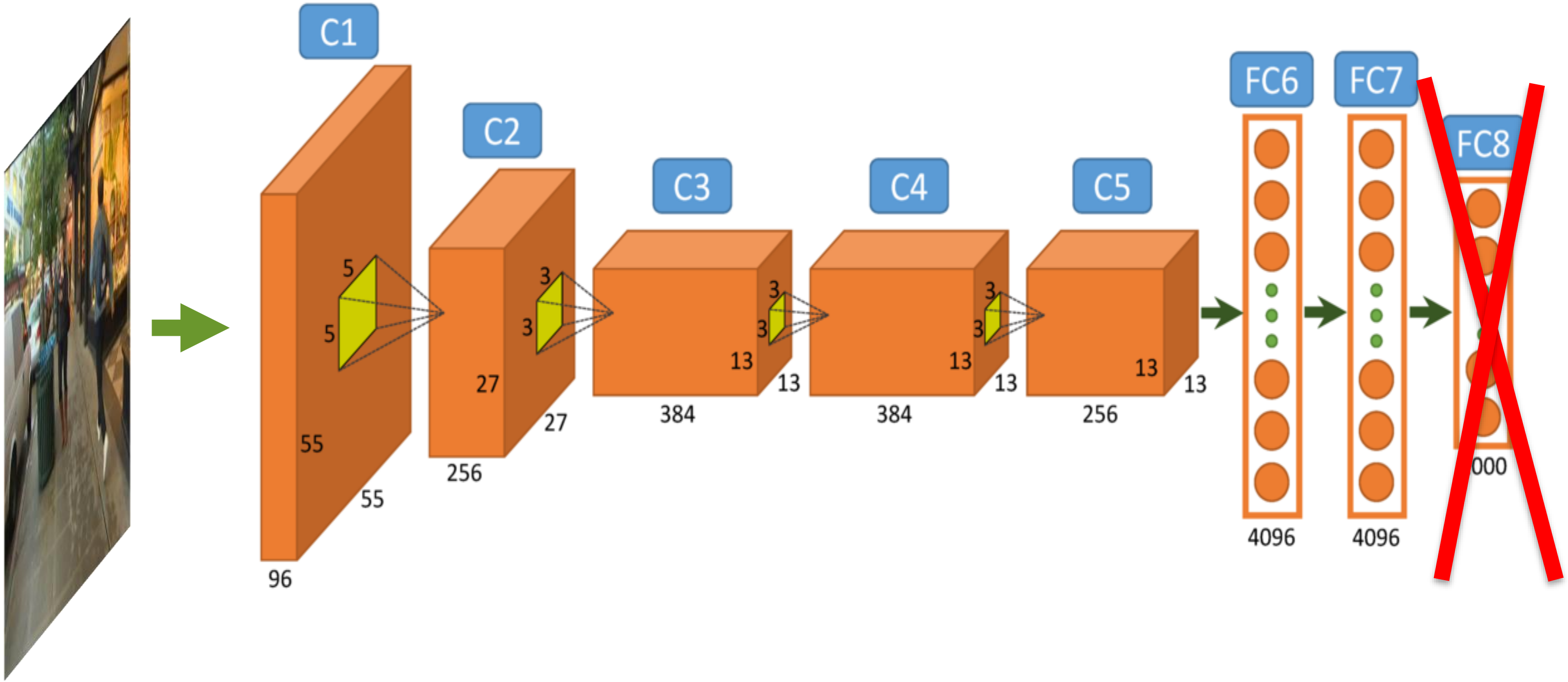
A short note on dataset management

- Nothing can be done without training data.
- Good training data & data handling is *essential* for good results
- Out of scope for this short intro
- More information found here:
  - https://www.tensorflow.org/guide/data

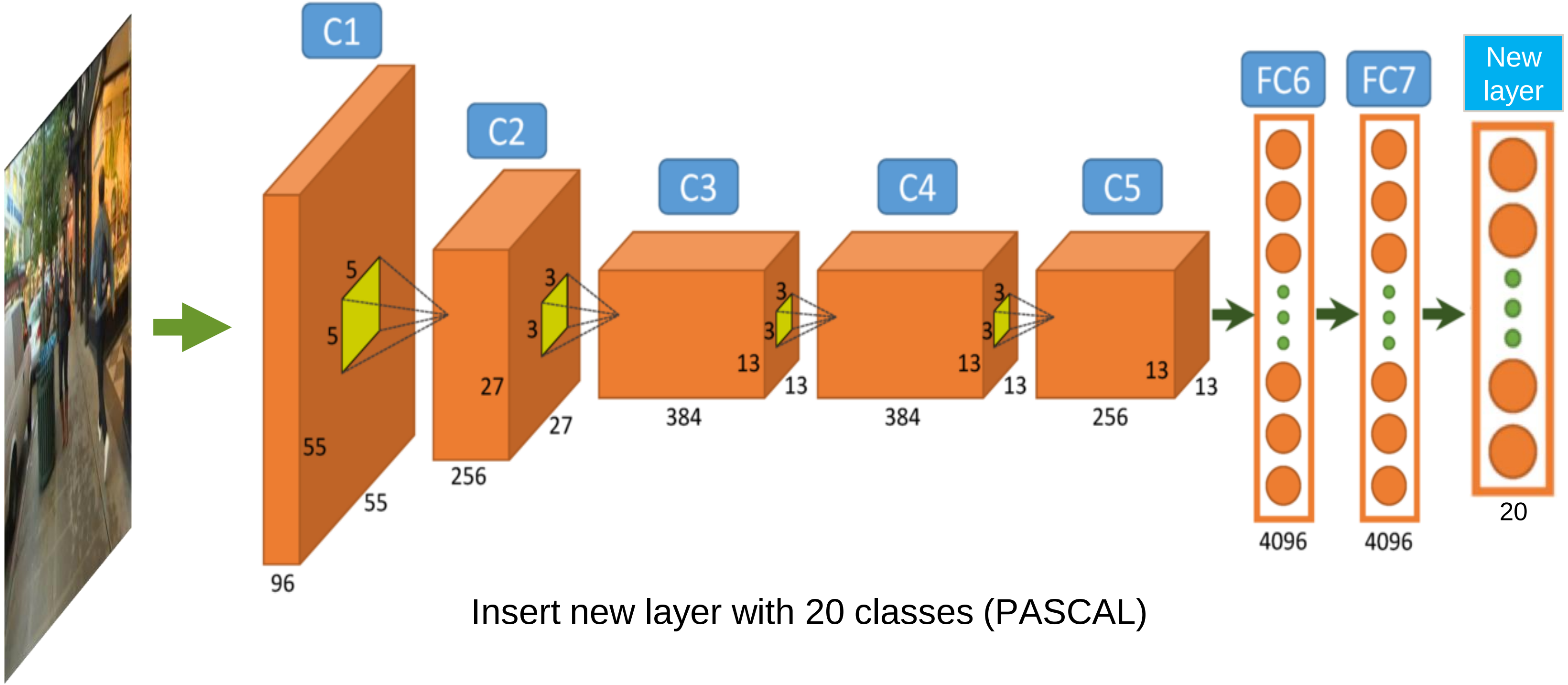
Part 4

Transfer Learning

Finetuning the last layer of a classifier



Finetuning the last layer of a classifier



Insert new layer with 20 classes (PASCAL)

Pretrained feature extractor, or base CNN

Several possible choices depending on your needs

TF Hub

Very simple to use
Opaque CNN
Only access to signatures
From remote server
Can not modify layer definition

Keras Applications

Very simple to use
Visible CNN architecture
Can access internal layers
From remote server
Can not modify layer definition
(except in-memory)

Custom approach

More work to use
Visible CNN architecture
Can access internal layers
Local architecture definition & weights
Can modify layer definition

Transfer Learning Using Tensorflow Hub

https://www.tensorflow.org/hub/

Separate install and import

```
pip install -q tensorflow_hub
import tensorflow_hub as hub
```

Find a feature extractor model (classifier with the last layer removed)
https://tfhub.dev/google/tf2-preview/mobilenet_v2/feature_vector/4

Download model with pretrained weights

```
feature_extractor_url = "https://tfhub.dev/google/tf2-
                         preview/mobilenet_v2/feature_vector/4"
pretrained_feature_extractor = hub.KerasLayer(feature_extractor_url,
                                         input_shape=(224,224,3))
```

Transfer Learning Using Tensorflow Hub

Freeze the layers of the feature extractor

```
pretrained_feature_extractor.trainable = False
```

TensorFlow Hub

Treat the "hub.Keraslayer" as any other Keras layer

```
model = tf.keras.Sequential([
    pretrained_feature_extractor,
    tf.keras.layers.Dense(20, activation='softmax')
])
```

Check the model and trainable weights information

```
model.summary()
```

Train as usual, e.g. using model.compile, model.fit(), model.evaluate()

Optionally, finetune further

- unfreezing the feature extractor layers: `trainable=True`
- Lower the learning rate (e.g. a factor 10)
- Run model.compile() again and train further by calling model.fit() and model.evaluate()

Finetuning using Keras Applications

Included in the core Tensorflow framework
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/applications

Keras Applications is actually a dependency of TensorFlow
(i.e. tf.keras imports keras_applications).
https://keras.io/applications/
https://github.com/keras-team/keras-applications

Find the desired pretrained CNN in the above TF link
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/applications/MobileNetV2

Download pretrained CNN and weights

```
pretrained_feature_extractor =
tf.keras.applications.MobileNetV2(input_shape=(224,224,3),
                                               include_top=False,
                                               weights='imagenet')
```

Transfer Learning using Keras Applications

Model definition similar to TF Hub, except feature extractor output format can vary

```
model = tf.keras.Sequential([
    pretrained_feature_extractor,
    tf.keras.layers.GlobalAveragePooling2D(),
    tf.keras.layers.Dense(20, activation='softmax')
])
```

model.summary()
useful here!

Note that the Keras Applications feature extractor is a **`Model`** instance,
Whereas for TF Hub we got a **`Layer`** instance via **`hub.KerasLayer()`**

The rest is the same as for training using TF Hub…

Exercise 4: ”Transfer Learning”

Exercise 4:
https://colab.research.google.com/drive/1U6FU7LNClWQ1NJEUDJ44NrHecRkpgzvg

Answers to exercise 4:
https://colab.research.google.com/drive/1GUDePuCBtT-h73zMPLDOWbWAsOT2fMr9


Thank you