Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition

Chapter 19. Training and Deploying TensorFlow Models at Scale

SanDiego Machine Learning
April 16, 2022
Discussion Leader: Nidhin Pattaniyil

Agenda

1. TensorFlow Serving
2. Deploying to the cloud
3. Deploying to Embedded Devices (TensorFlow Lite)
4. Deploying to the browser (TensorFlow.js)
5. Training across multiple devices

TensorFlow Serving

Tensorflow Serving

- efficient, battle-tested model server that's written in C++.
- serve multiple versions of your models and watch a model repository to automatically deploy the latest versions
- Serve multiple models (control if using a GPU / CPU)
- Control automatic batching with max_batch_size and batch_timeout_micros
- Exports prometheus metrics for monitoring

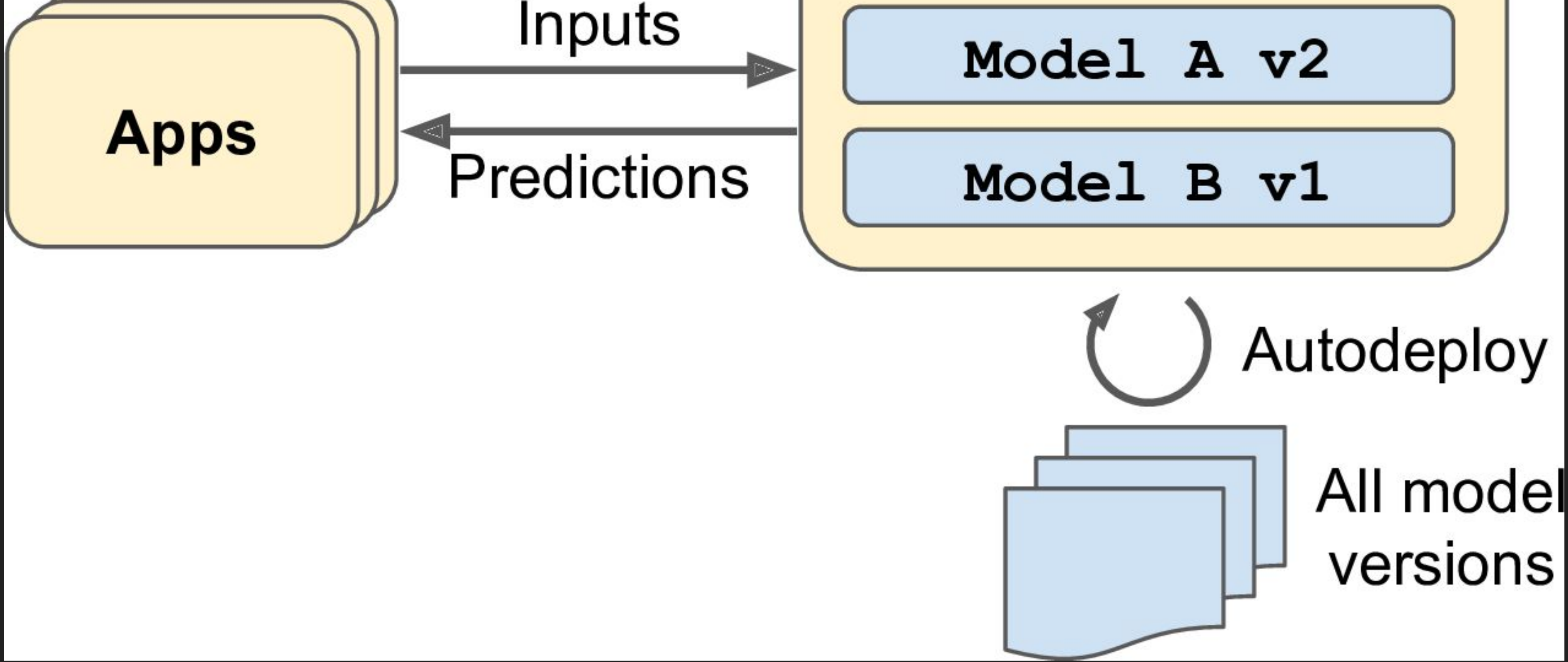


Figure 19-1. TF Serving can serve multiple models and automatically deploy the latest version of each model

Exporting SavedModel

```
model = keras.models.Sequential([...])
model.compile([...])
history = model.fit([...])

model_version = "0001"
model_name = "my_mnist_model"
model_path = os.path.join(model_name, model_version)
tf.saved_model.save(model, model_path)
```

```
my_mnist_model
└── 0001
    ├── assets
    ├── saved_model.pb
    └── variables
        ├── variables.data-00000-of-00001
        └── variables.index
```

TensorFlow also comes with a small `saved_model_cli` command-line tool to inspect SavedModels:

```
$ export ML_PATH="$HOME/ml" # point to this project, wherever it is
$ cd $ML_PATH
$ saved_model_cli show --dir my_mnist_model/0001 --all
MetaGraphDef with tag-set: 'serve' contains the following SignatureDefs:
signature_def['__saved_model_init_op']:
  [...]

signature_def['serving_default']:
  The given SavedModel SignatureDef contains the following input(s):
    inputs['flatten_input'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 28, 28)
        name: serving_default_flatten_input:0
  The given SavedModel SignatureDef contains the following output(s):
    outputs['dense_1'] tensor_info:
        dtype: DT_FLOAT
        shape: (-1, 10)
        name: StatefulPartitionedCall:0
  Method name is: tensorflow/serving/predict
```

SavedModel

- A SavedModel contains one or more metagraphs.
- A metagraph is a computation graph plus some function signature definitions (including their input and output names, types, and shapes).
- Each metagraph is identified by a set of tags.

- Metagraph for train (full computation graph), serve (pruned computation graph)
- Tf.save saves a single metagraph tagged "serve", which contains a default serving function (called serving_default).

```
model = keras.models.load_model(model_path)
y_pred = model.predict(tf.constant(X_new, dtype=tf.float32))
```

Serving a model

```
$ docker run -it --rm -p 8500:8500 -p 8501:8501 \
             -v "$ML_PATH/my_mnist_model:/models/my_mnist_model" \
             -e MODEL_NAME=my_mnist_model \
             tensorflow/serving
[...]
2019-06-01 [...] loaded servable version {name: my_mnist_model version: 1}
2019-06-01 [...] Running gRPC ModelServer at 0.0.0.0:8500 ...
2019-06-01 [...] Exporting HTTP/REST API at:localhost:8501 ...
[evhttp_server.cc : 237] RAW: Entering the event loop ...
```

Making a request

```python
import json

input_data_json = json.dumps({
    "signature_name": "serving_default",
    "instances": X_new.tolist(),
})
```

```python
import requests

SERVER_URL = 'http://localhost:8501/v1/models/my_mnist_model:predict'
response = requests.post(SERVER_URL, data=input_data_json)
response.raise_for_status() # raise an exception in case of error
response = response.json()
```

Reasons why TF-Serving may not be right

- Only serving 1 model or 1 model per container
- Only doing cpu based inference
- Dynamic models
- Custom processing / preprocessing

```
[...]
reserved resources to load servable {name: my_mnist_model version: 2}
[...]
Reading SavedModel from: /models/my_mnist_model/0002
Reading meta graph with tags { serve }
Successfully loaded servable version {name: my_mnist_model version: 2}
Quiescing servable version {name: my_mnist_model version: 1}
Done quiescing servable version {name: my_mnist_model version: 1}
Unloading servable version {name: my_mnist_model version: 1}
```

Deploying to Google Cloud Platform

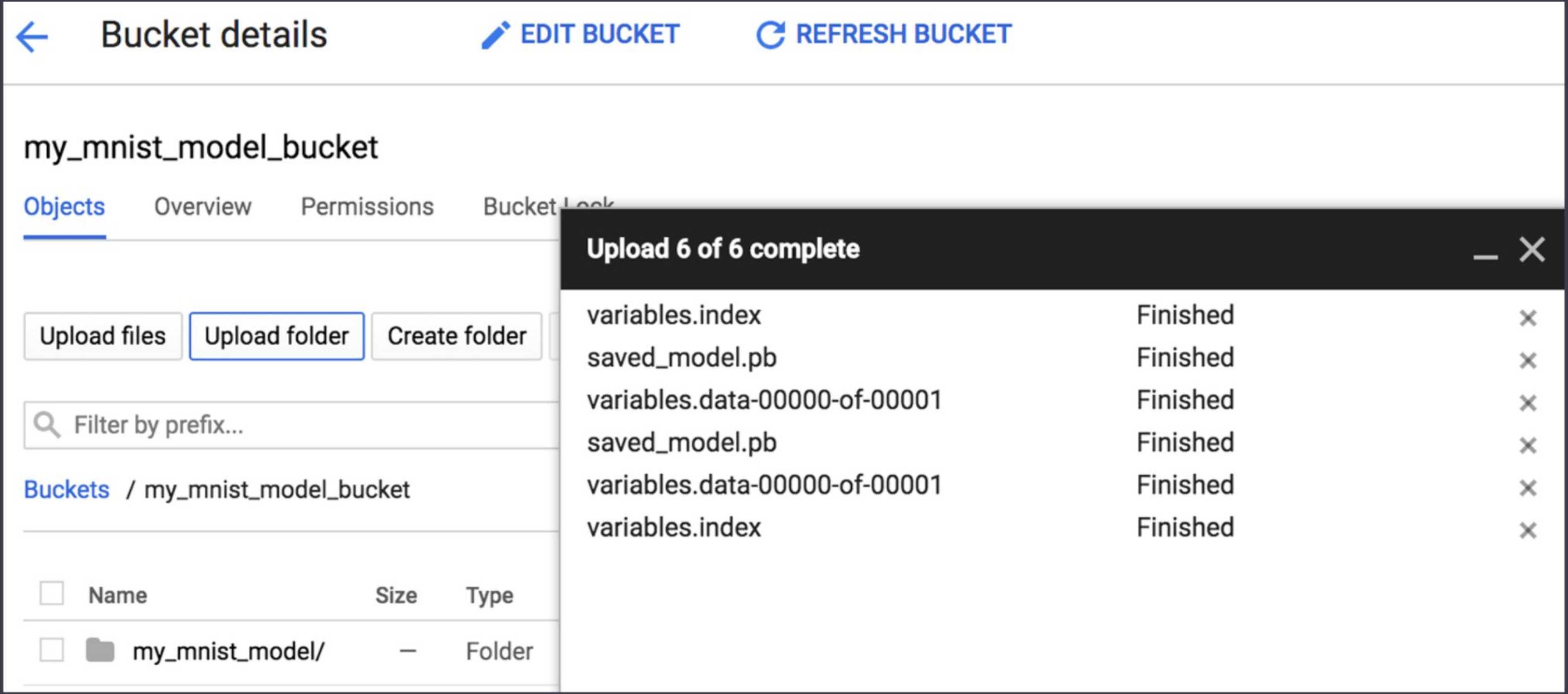


Figure 19-4. Uploading a SavedModel to Google Cloud Storage

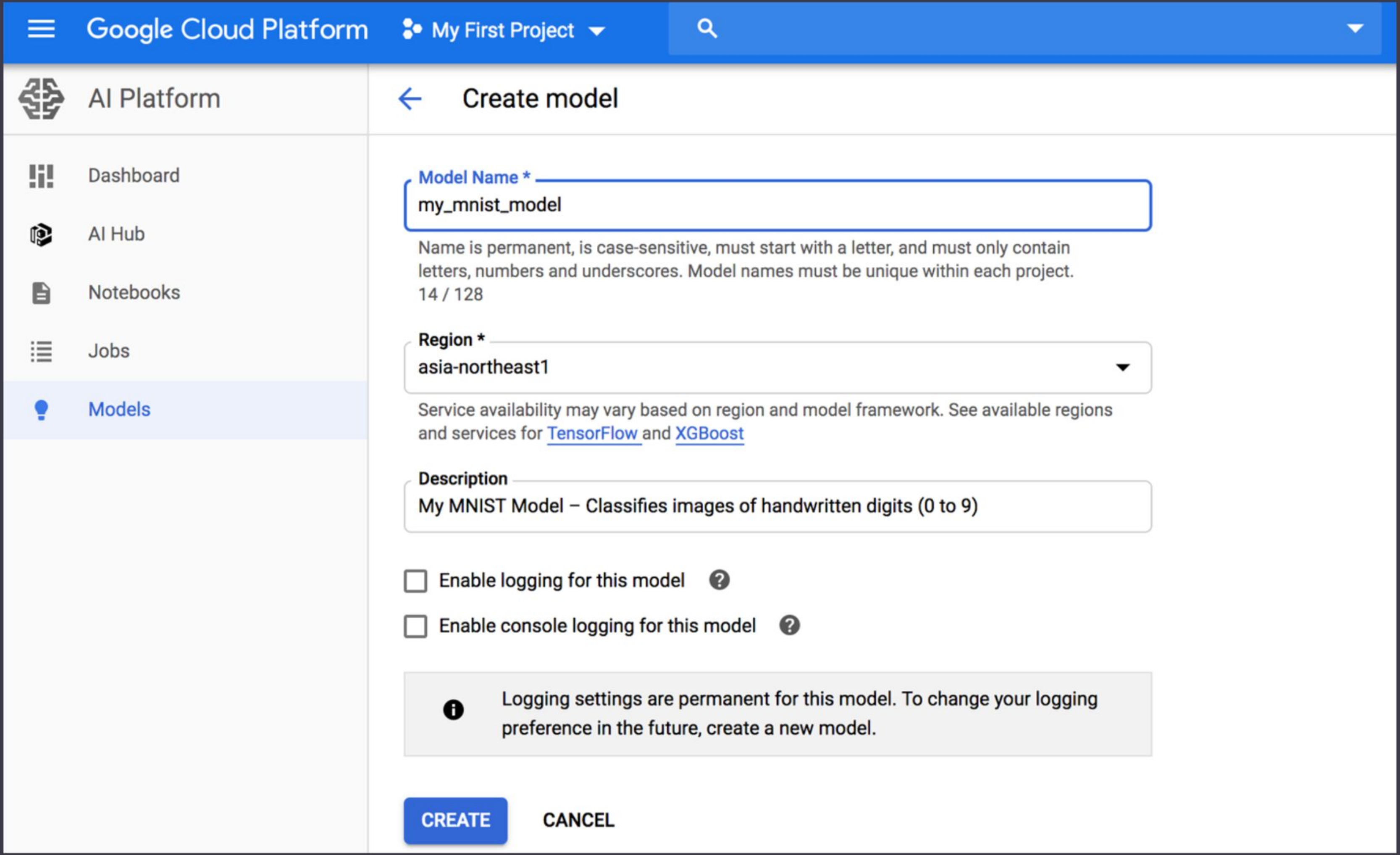


Figure 19-5. Creating a new model on Google Cloud AI Platform

```python
import googleapiclient.discovery

project_id = "onyx-smoke-242003" # change this to your project ID
model_id = "my_mnist_model"
model_path = "projects/{}/models/{}".format(project_id, model_id)
ml_resource = googleapiclient.discovery.build("ml", "v1").projects()
```

```python
def predict(X):
    input_data_json = {"signature_name": "serving_default",
                       "instances": X.tolist()}
    request = ml_resource.predict(name=model_path, body=input_data_json)
    response = request.execute()
    if "error" in response:
        raise RuntimeError(response["error"])
    return np.array([pred[output_name] for pred in response["predictions"]])
```

Deploying to Embedded Device

TensorFlow Lite

- mobile library for deploying models on mobile, microcontrollers and other edge devices.
- Reduce the model size, to shorten download time and reduce RAM usage.
- Reduce the amount of computations needed for each prediction, to reduce latency, battery usage, and heating.
- Adapt the model to device-specific constraints

Conversion Code

```python
converter = tf.lite.TFLiteConverter.from_saved_model(saved_model_path)
tflite_model = converter.convert()
with open("converted_model.tflite", "wb") as f:
    f.write(tflite_model)
```

TFLite Optimization

- FlatBuffers (easier to load directly in memory)
- prunes all the operations that are not needed to make predictions (such as training operations)
- optimizes computations whenever possible; for example, $3×a + 4×a + 5×a$ will be converted to $(3 + 4 + 5)×a$.
- Post training quantization

Quantization

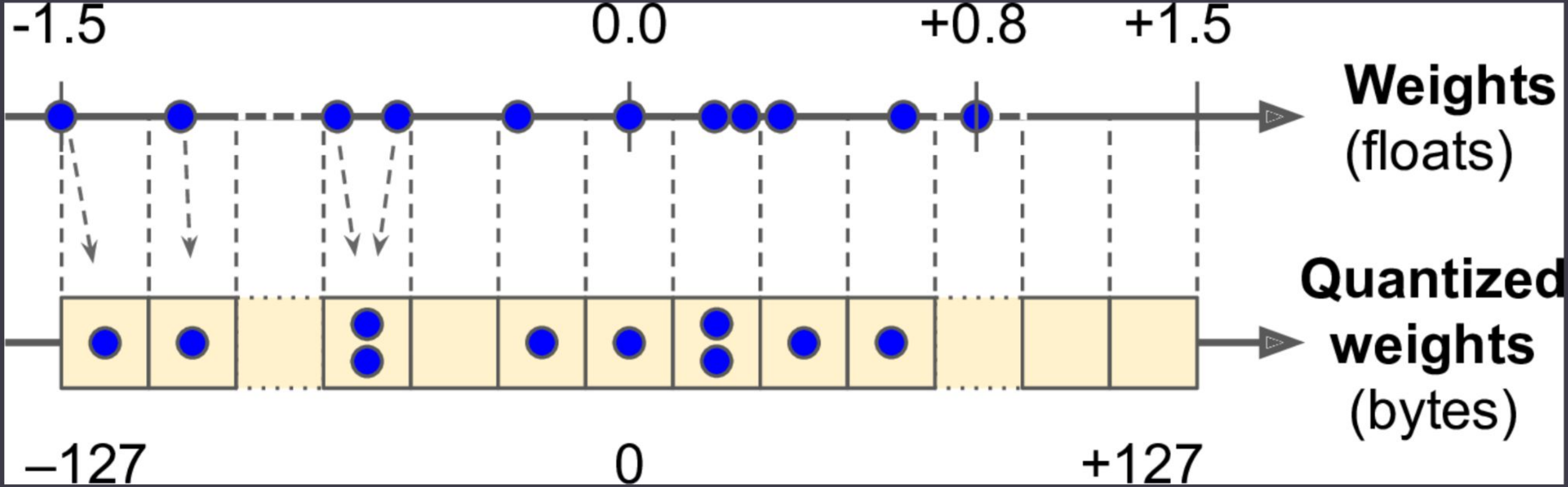


Figure 19-8. From 32-bit floats to 8-bit integers, using symmetrical quantization

Post training Quantization

- Weights can be converted to types with reduced precision, such as 16 bit floats or 8 bit integers.
- Smaller to store and load in memory
- At runtime the quantized weights get converted back to floats before they are used

Post training Quantization Accuracy

Below are the accuracy results for some models that benefit from this mode.

Model	Accuracy metric type	Accuracy (float32 activations)	Accuracy (int8 activations)	Accuracy (int16 activations)
Wav2letter	WER	6.7%	7.7%	7.2%
DeepSpeech 0.5.1 (unrolled)	CER	6.13%	43.67%	6.52%
YoloV3	mAP(IOU=0.5)	0.577	0.563	0.574
MobileNetV1	Top-1 Accuracy	0.7062	0.694	0.6936
MobileNetV2	Top-1 Accuracy	0.718	0.7126	0.7137
MobileBert	F1(Exact match)	88.81(81.23)	2.08(0)	88.73(81.15)

Table 2 Benefits of model quantization with int16 activations

Reference:

https://www.tensorflow.org/lite/performance/model_optimization

TensorFlow Lite MobileNetv2 Device Benchmarks

Device	CPU-F (ms)	CPU-Q (ms)	NN-FP16 (ms)	NN-INT8 (ms)
Google Pixel 2	91	85	116	87
Google Pixel 3	70	63	14	8.2
Google Pixel 4a	63	30	21	8.4
Google Pixel 5	50	22	22	6
Google Pixel 6	64	22	2.1	1.5

CPU-F: FP16 model running on CPU
CPU-Q: INT8 model running on CPU
NN-FP16: Fp16 model running with acceleration (NNAPI or Delegates)
NN-INT8: INT8 model running with acceleration (NNAPI or Delegates)

https://ai-benchmark.com/ranking_detailed.html

Mobile Benchmarks

SoC Model	MobileNet-V2							
	CPU-Q	CPU-F	NN-INT8			NN-FP16		
	ms	ms	ms	ms, bs=8	error, L1	ms	ms, bs=8	acc, digits
Snapdragon 8 Gen 1	15	27	0.6	0.2	0.72	1.4	0.6	4.7
Dimensity 9000	6.4	10.3	1.5	0.4	0.7	2.3	1	4.8
Snapdragon 888 Plus	22	49	0.9	0.3	0.72	7.5	4.7	4.9
Snapdragon 888	19	38	0.9	0.3	0.72	8	4.8	4.9
Google Tensor	18	31	1.6	0.6	0.86	2.2	1.3	4.5
Snapdragon 778G	17	36	1.4	0.5	0.72	12	11	4.9
Exynos 2100	11	19	4.5	2.2	1.08	5	2.1	4.7

https://ai-benchmark.com/ranking_processors_detailed.html

TensorFlow.js

TensorFlow.js

- Develop ML models in JavaScript
- use ML directly in the browser or in Node.js.
- Why:
  - Bad internet connection
  - Fast inference
  - Privacy

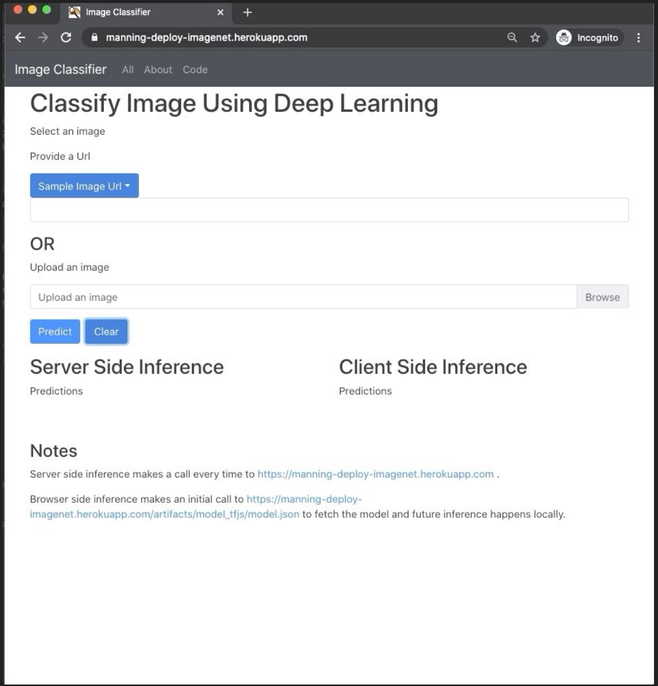


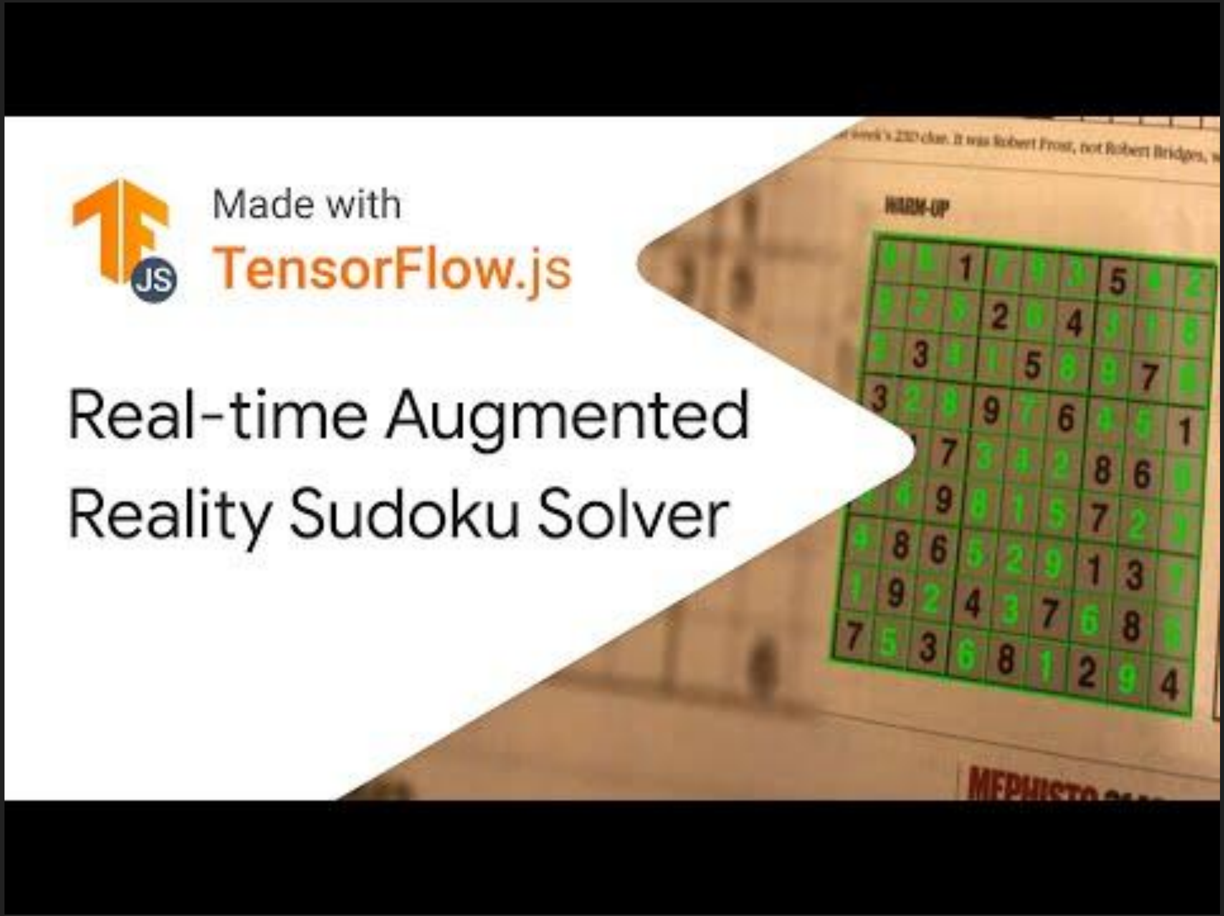
Image Classifier
manning-deploy-imagenet.herokuapp.com
Incognito
Image Classifier
All
About
Code
Classify Image Using Deep Learning
Select an image
Provide a Url
Sample Image Url
OR
Upload an image
Upload an image
Browse
Predict
Clear
Server Side Inference
Predictions
Client Side Inference
Predictions
Notes
Server side inference makes a call every time to https://manning-deploy-imagenet.herokuapp.com .
Browser side inference makes an initial call to https://manning-deploy-imagenet.herokuapp.com/artifacts/model_tfjs/model.json to fetch the model and future inference happens locally.

URL

Code

`tensorflowjs_converter`

```
import * as tf from '@tensorflow/tfjs';
const model = await tf.loadLayersModel('https://example.com/tfjs/model.json');
const image = tf.fromPixels(webcamElement);
const prediction = model.predict(image);
```


Made with
TensorFlow.js
Real-time Augmented
Reality Sudoku Solver


Made with
TensorFlow.js
yogAI with
Cristina Maillo
Tree
00:09
seconds left to hold

Training Models Across Multiple Devices

Environment variables

```
$ CUDA_DEVICE_ORDER=PCI_BUS_ID CUDA_VISIBLE_DEVICES=0,1 python3 program_1.py
# and in another terminal:
$ CUDA_DEVICE_ORDER=PCI_BUS_ID CUDA_VISIBLE_DEVICES=3,2 python3 program_2.py
```

CUDA_VISIBLE_DEVICES environment variable so that each process only sees the appropriate GPU card(s).

Also set the CUDA_DEVICE_ORDER environment variable

Model Parallelism

- Split model across multiple devices

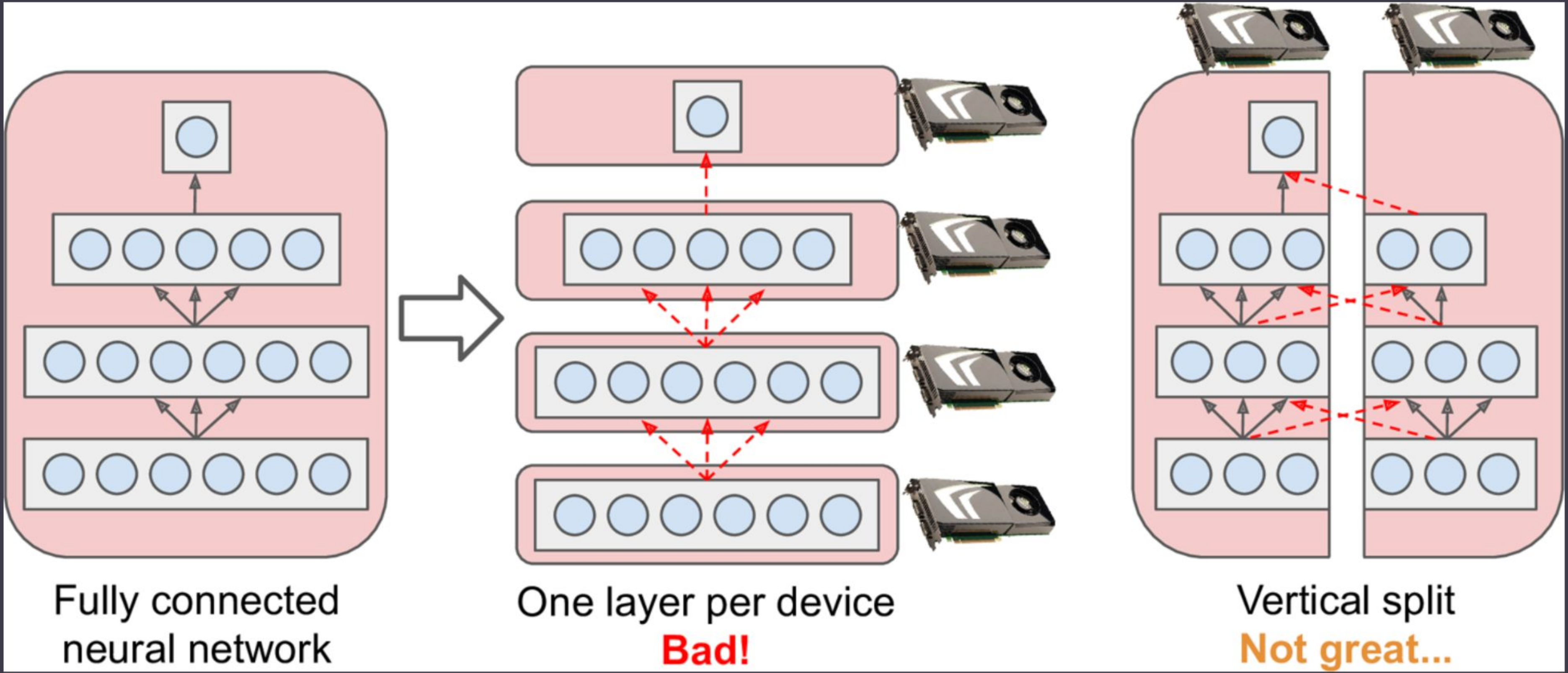


Figure 19-15. Splitting a fully connected neural network

Data Parallelism

- replicate model on every device
- run each training step simultaneously on all replicas, using a different mini-batch for each.
- The gradients computed by each replica are then averaged, and the result is used to update the model parameters.

Data parallelism using the mirrored strategy

- completely mirror all the model parameters across all the GPUs and always apply the exact same parameter updates on every GPU.
- The tricky part when using this approach is to efficiently compute the mean of all the gradients from all the GPUs and distribute the result across all the GPUs.

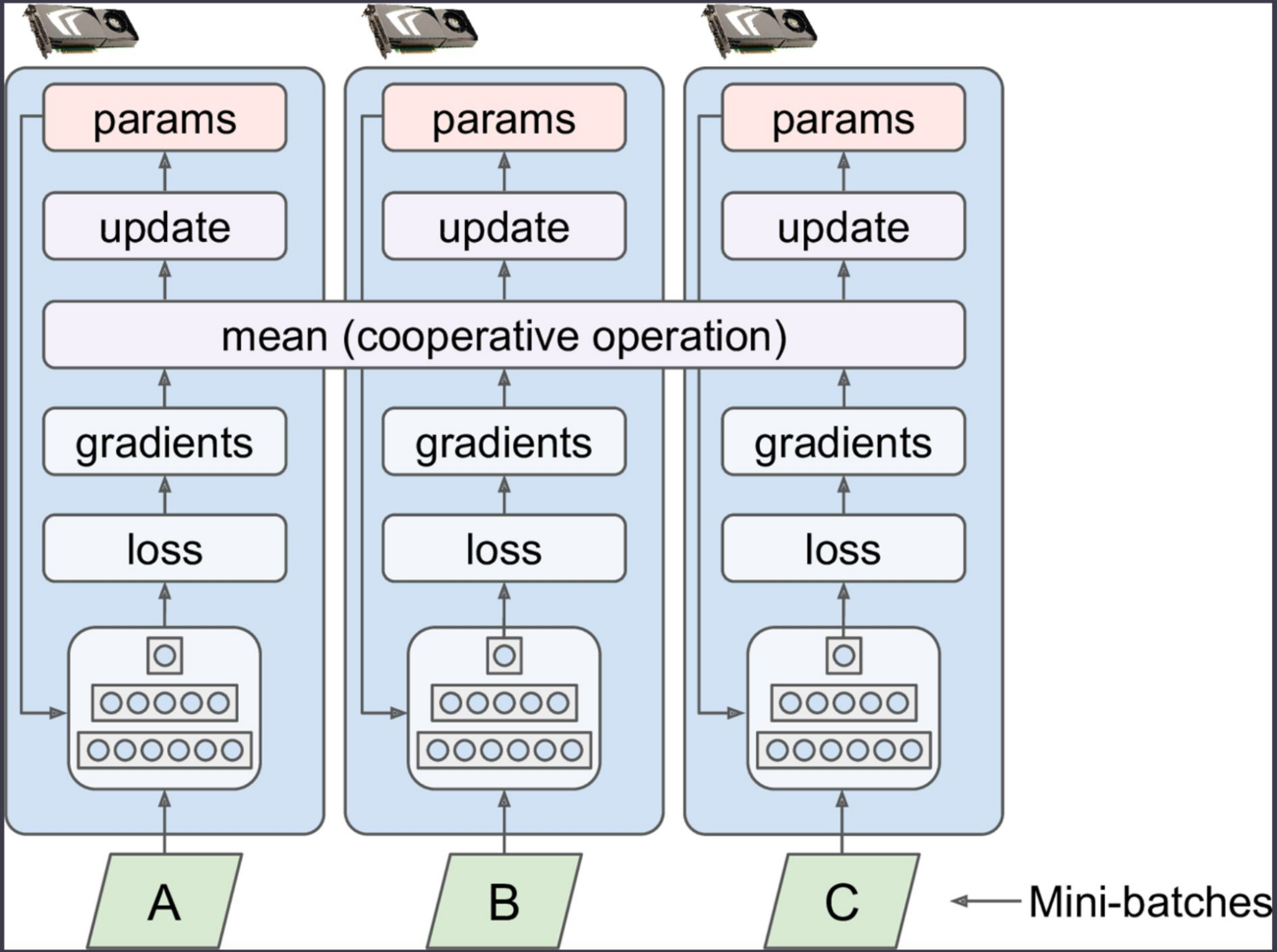


Figure 19-18. Data parallelism using the mirrored strategy

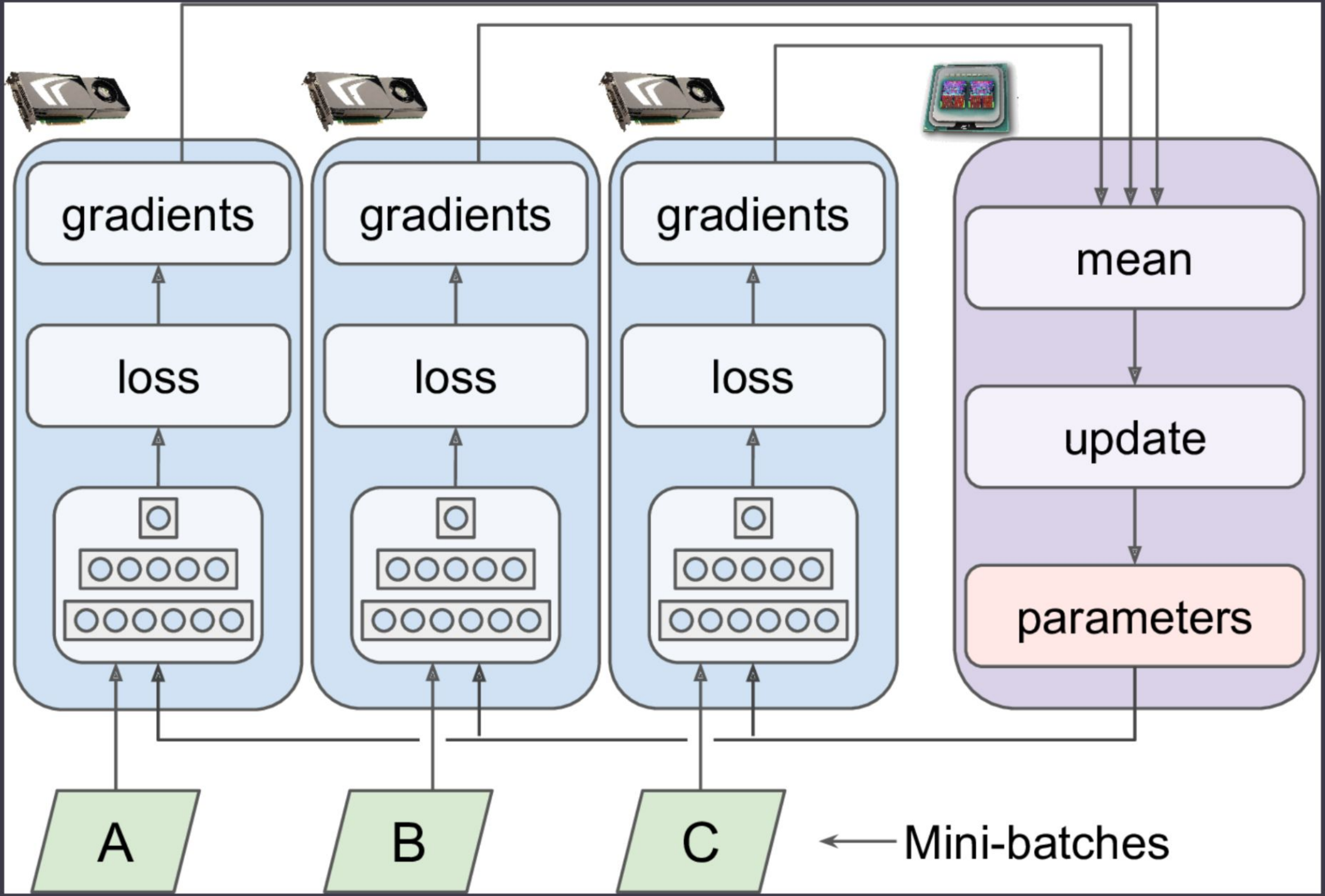


Figure 19-19. Data parallelism with centralized parameters

Data parallelism with centralized parameters

- store the model parameters outside of the GPU devices performing the computations
- Parameter server: host and update the parameters.
- To reduce the waiting time at each step, you could ignore the gradients from the slowest few replicas (typically ~10%). F

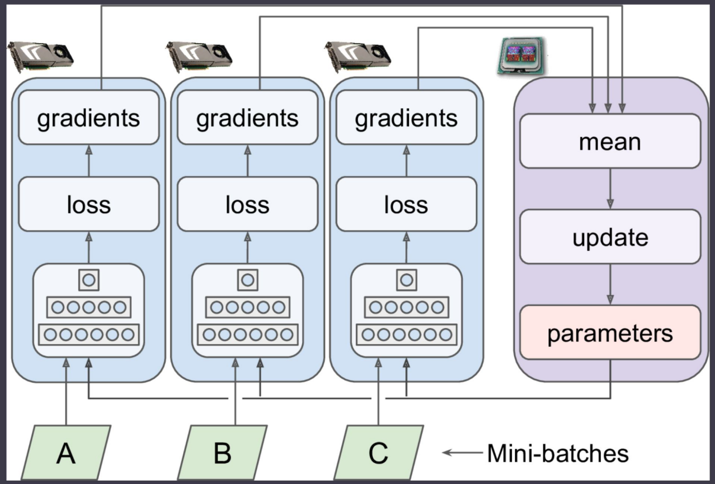


Figure 19-19. Data parallelism with centralized parameters

GPU Speedup

- Neural machine translation: 6× speedup on 8 GPUs
- Inception/ImageNet: 32× speedup on 50 GPUs
- RankBrain: 300× speedup on 500 GPUs

Training at Scale Using the Distribution Strategies API

```
distribution = tf.distribute.MirroredStrategy()

with distribution.scope():
    mirrored_model = keras.models.Sequential([...])
    mirrored_model.compile([...])

batch_size = 100 # must be divisible by the number of replicas
history = mirrored_model.fit(X_train, y_train, epochs=10)
```

replicate all variables and operations across all available GPU devices. Note that the fit() method will automatically split each training batch across all the replicas,

Distributed with Central Parameters

```
distribution = tf.distribute.experimental.CentralStorageStrategy()
```

Training a Model on a TensorFlow Cluster

- A TensorFlow cluster is a group of TensorFlow processes running in parallel,
- Worker, chief, parameter server

```
cluster_spec = {
    "worker": [
        "machine-a.example.com:2222",  # /job:worker/task:0
        "machine-b.example.com:2222"   # /job:worker/task:1
    ],
    "ps": ["machine-a.example.com:2221"] # /job:ps/task:0
}
```

```python
import os
import json

os.environ["TF_CONFIG"] = json.dumps({
    "cluster": cluster_spec,
    "task": {"type": "worker", "index": 0}
})
```

```python
distribution = tf.distribute.experimental.MultiWorkerMirroredStrategy()

with distribution.scope():
    mirrored_model = keras.models.Sequential([...])
    mirrored_model.compile([...])

batch_size = 100 # must be divisible by the number of replicas
history = mirrored_model.fit(X_train, y_train, epochs=10)
```

```
resolver = tf.distribute.cluster_resolver.TPUClusterResolver()
tf.tpu.experimental.initialize_tpu_system(resolver)
tpu_strategy = tf.distribute.experimental.TPUStrategy(resolver)
```

Running Large Training Jobs on Google Cloud AI Platform

```
gcloud ai-platform jobs submit training my_job_20190531_164700 \
  --region asia-southeast1 \
  --scale-tier PREMIUM_1 \
  --runtime-version 2.0 \
  --python-version 3.5 \
  --package-path /my_project/src/trainer \
  --module-name trainer.task \
  --staging-bucket gs://my-staging-bucket \
  --job-dir gs://my-mnist-model-bucket/trained_model \
  --
  --my-extra-argument1 foo --my-extra-argument2 bar
```

References

- 19. Training and Deploying TensorFlow Models at Scale
  GeÌ ron, Aureì lien. Hands-on Machine Learning with Scikit-Learn, Keras and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2nd ed., O'Reilly, 2019.