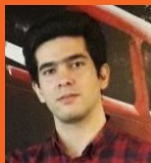
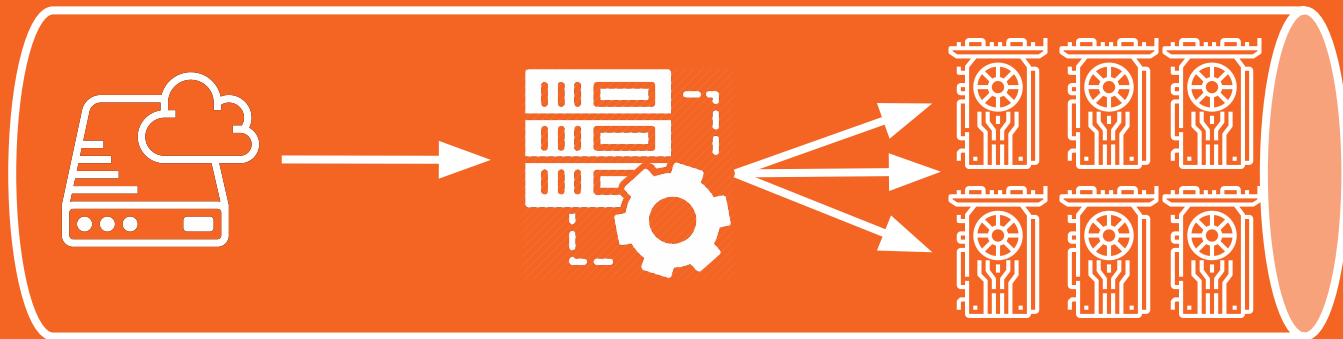




Input Pipelines

How to build efficient data pipelines with **tensorflow**.



Masoud Masoumi Moghaddam 1400 Ordibehesht



Overview

- Why input pipeline?
 - Image Augmentation
 - Keras *ImageDataGenerator*
 - Keras *utils.Sequence*
 - Practice #1: Violence Detection
 - tf.data
 - tf.data vs keras data generators
 - tf.data pipeline
 - Boosting cpu performance
 - Parallel software pipeline
 - Parallel transformation
 - Parallel Extraction
 - tf.snapshot
-



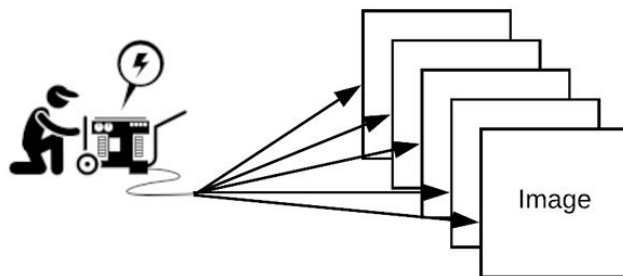
Overview

- Practice #2: Digikala products color classification
 - albumentation
 - Classification Evaluation Metrics
 - transfer learning
 - Tensorboard GradCam plot
 - Tensorboard Confusion matrix
 - Tf.profile
 - Boosting Performance on GPU
 - tf.function
 - XLA
 - MixedPrecision
 - tf.distribute
-



Why input pipeline?

- Data might not fit into memory.
- Data might require (randomized) pre-processing. We need to do things on the fly (like augmentations).
- Efficiently utilize hardware.
- Decouple loading and pre-processing from distribution.





Input pipeline stages

1. Extract:
 - Read from memory/storage
 - Parse file format
2. **Transform:**
 - Text vectorization
 - **Image transformations** (alumentation - imgaug)
 - **Video temporal sampling** (Violence detection use-case)
 - Shuffling, batching
3. Load
 - Transfer data to the accelerator (GPU/TPU)



Image Augmentation

Image augmentation is a technique that can be used to artificially expand the size of a training dataset by creating modified versions of **images** in the dataset.

Source:

[A survey on Image Data Augmentation for Deep Learning](#)

- Geometric transformation
 - Flipping
 - Cropping
 - Rotation
 - Translation (shift top, left, ...)
 - Noise Injection
 -
- Color space transformation
 - RGB shift
 - Hue Saturation value
 - Channel Shuffle
 - Random Contrast
 -



Benefits

- Better Generalization
- Avoid overfitting

Source

[Albumentation](#) repo

[Imgaug](#) repo





Keras ImageDataGenerator



How to use ImageDataGenerator?

Data path format

- train/
 - dog/
 - dog001.png
 - dog002.png
 - ...
 - cat/
 - cat001.png
 - cat002.png
 - ...
- Validation
 - dog
 - dog001.png
 - Dog002.png
 - ...
 - cat
 - cat001.png
 - cat002.png
 -
 -

Split to train/val/test

```
pip install split-folders
```

```
splitfolders.ratio("input_folder", output="output", seed=1400,  
ratio=(.8, .1, .1))
```

```
train_datagen = ImageDataGenerator(  
    rescale=1./255,  
    shear_range=0.2,  
    zoom_range=0.2,  
    horizontal_flip=True)  
  
train_generator = train_datagen.flow_from_directory(  
    'data/train',  
    target_size=(224, 224),  
    batch_size=32)
```

Also useful for segmentation. Check [this link](#)



tf.keras.utils.Sequence

ImageDataGenerator is not always applicable:

- Working with cube tensors (will be discussed through example).
- Object detection tasks.
- Training Siamese networks.
-

How to use *keras.utils.Sequence*:

- Overwrite `__len__` (how many data points exist).
 - `steps_per_epoch = np.ceil(len(self.X_path) / float(self.batch_size))`
- Overwrite `__getitem__` (how to get each item).
 - gets an integer `index`
 - outputs a tuple containing (batch_x, batch_y)
- * Also `on_epoch_end` could be used



Let's code

**Practice#1:
Violence Detection**

Practice #1

Violence Detection

3D Tensors

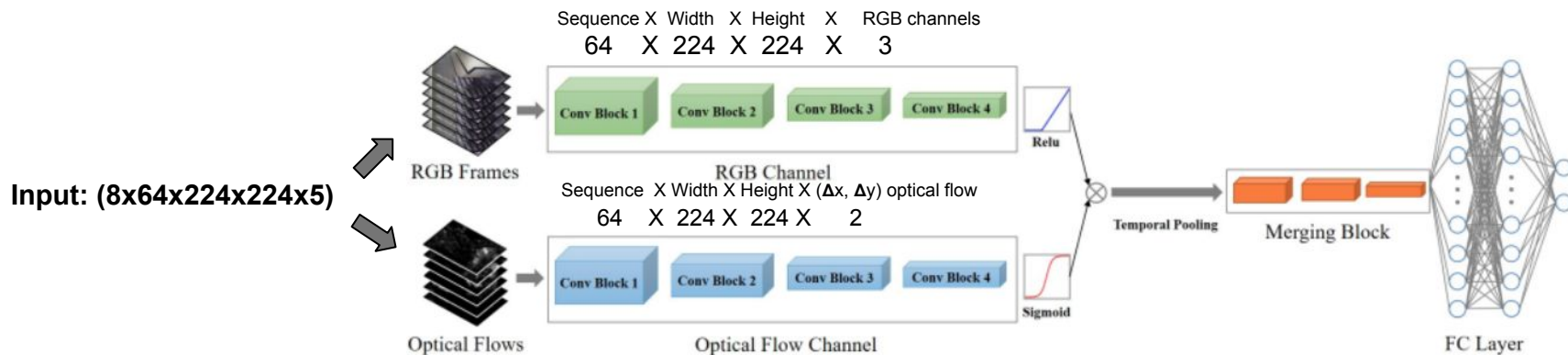
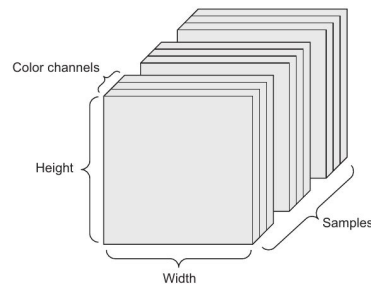


Figure 4. The structure of the Flow Gated Network.



Practice #1

Violence Detection

Path to data

data/

- train/
 - Violence/
 - V001.npy → RGB channels + Optical Flow
 - V002.npy
 - ...
 - Non-Violence/
 - NV001.npy
 - NV002.npy
 - ...

```
self.X_path = ['train/violence/V001.npy',  
               'train/violence/V002.npy',  
               ....  
               'train/non-violence/NV100.npy']  
  
self.Y_dict = {'train/violence/V001.npy': [01],  
               'train/non-violence/NV001.npy': [10],  
               ....  
               }
```

each time `__getitem__` is called:

- video is fetched
- video is sampled (64 frames/300 frames)

operation above is done for one batch



Practice #1:

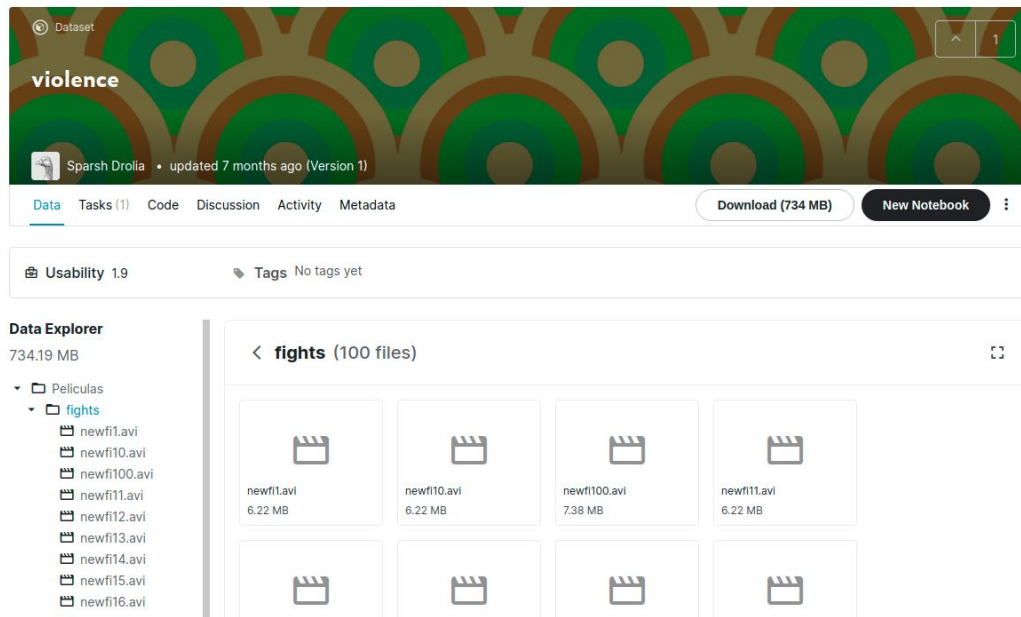
Violence Detection

In this practice we use `keras.utils.sequence_feed_data` to keras model which accepts batches of 3d tensors and classifies violence/non violence videos.

Dataset consists of:

- 100 violence videos
- 101 non-violence videos

Click on the image to open notebook
in kaggle





tf.data



Keras.utils.Sequence VS tf.data

The keras.utils.Sequence method is convenient, but has three downsides:

- It's slow.
- It lacks fine-grained control.
- It is not well integrated with the rest of Tensorflow.



Keras ImageDataGenerator VS tf.data

ImageDataGenerator vs tf.data:

- *keras.utils.Sequence* is better integratable with non-Tensorflow libraries like *PIL* or *numpy* (*multiple python processes*).
- Tensor operations is better applicable with *tf.data.Dataset* (*C++ threads*).

```
# `keras.preprocessing`  
timeit(train_data_gen)
```

```
1000 batches: 79.98093152046204 s  
400.09537 Images/s
```

```
# `tf.data`  
timeit(train_ds)
```

```
1000 batches: 5.8518688678741455 s  
5468.33853 Images/s
```

More discussion:

<https://stackoverflow.com/a/59492947/6118987>

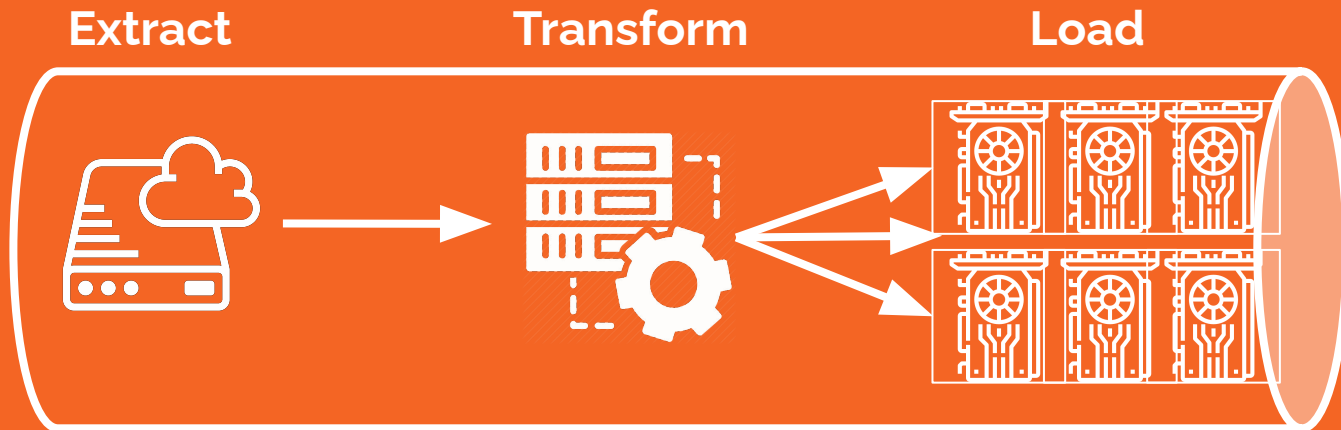


tf.data

- **Fast...** to keep up with GPUs and TPUs.
- **Flexible...** to handle diverse data and use cases.
- **Easy to use...** to democratize machine learning.



Let's design a typical tf.data pipeline





Extraction

E `dataset = tf.data.Dataset.list_files('*/*.jpg')`

T `dataset = dataset.map(preprocess)`
`dataset = dataset.shuffle(dataset.cardinality())`
`dataset = dataset.batch(batch_size=BATCH_SIZE)`

L `....`
`model =`
`model.fit(dataset, epochs=10)`

- `tf.data.Dataset.from_tensors([1, 2, 3])`
- `tf.data.TextLineDataset(['1.txt', ...])`
- `tf.data.TFRecordDataset(['file1.tfrecord'])`
- `tf.data.Dataset.from_generator(callable_func)`
- `tf.data.Dataset.list_files('*/*.jpg')`



Transformation and Loading

E `dataset = tf.data.Dataset.list_files('*/*.jpe*g')`

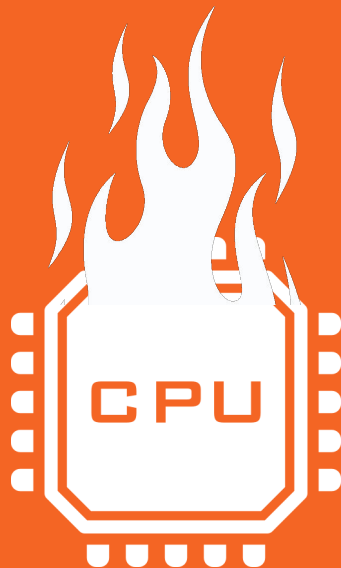
T `dataset = dataset.map(preprocess)`
`dataset = dataset.shuffle(dataset.cardinality())`
`dataset = dataset.batch(batch_size=BATCH_SIZE)`

L `iterator = dataset.make_one_shot_iterator()`
`features = iterator.get_next()`
`model =`
`model.fit(dataset, epochs=10)`

```
def preprocess(file_path):  
    class_names = np.array(['daisy', 'dandelion',  
                             'roses', 'sunflowers',  
                             'tulips'])  
  
    # convert the path to a list of path components  
    parts = tf.strings.split(file_path, os.path.sep)  
    # The second to last is the class-directory  
    one_hot = parts[-2] == class_names  
    # Integer encode the label  
    label = tf.argmax(one_hot)  
  
    img = tf.io.read_file(file_path)  
    img = tf.image.decode_jpeg(img, channels=3)  
    img = tf.image.resize(img, [224, 224])  
  
    return img, label
```

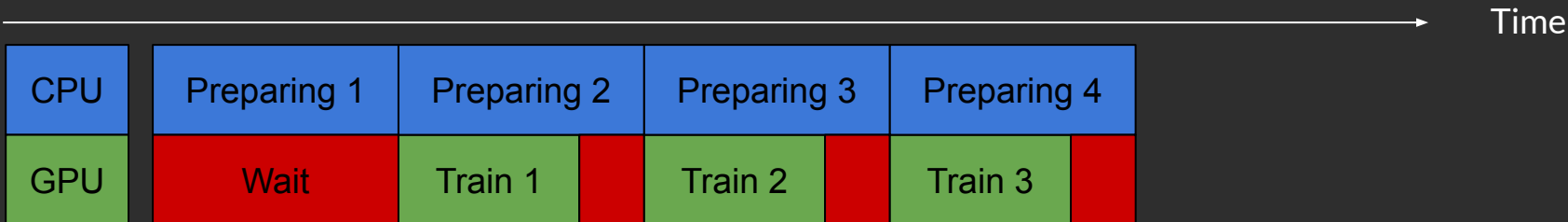
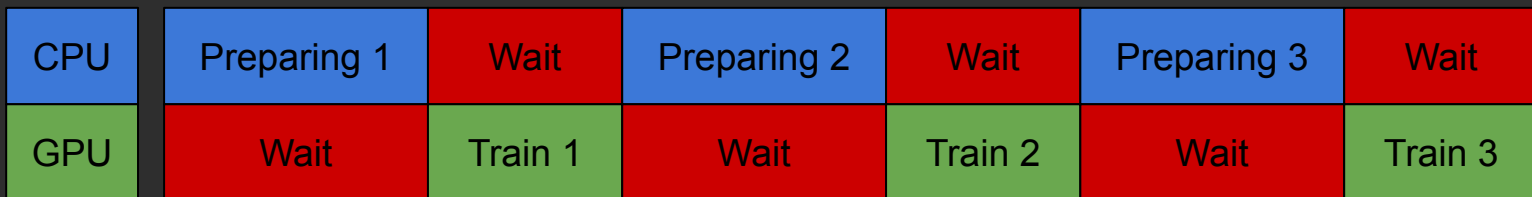


Boosting CPU performance





Parallel software Pipeline





Prefetching next batches of data

```
dataset = tf.data.Dataset.list_files('*/*.jpg')  
  
dataset = dataset.map(preprocess)  
dataset = dataset.shuffle(dataset.cardinality())  
dataset = dataset.batch(batch_size=BATCH_SIZE)  
dataset = dataset.prefetch(buffer_size=X)
```

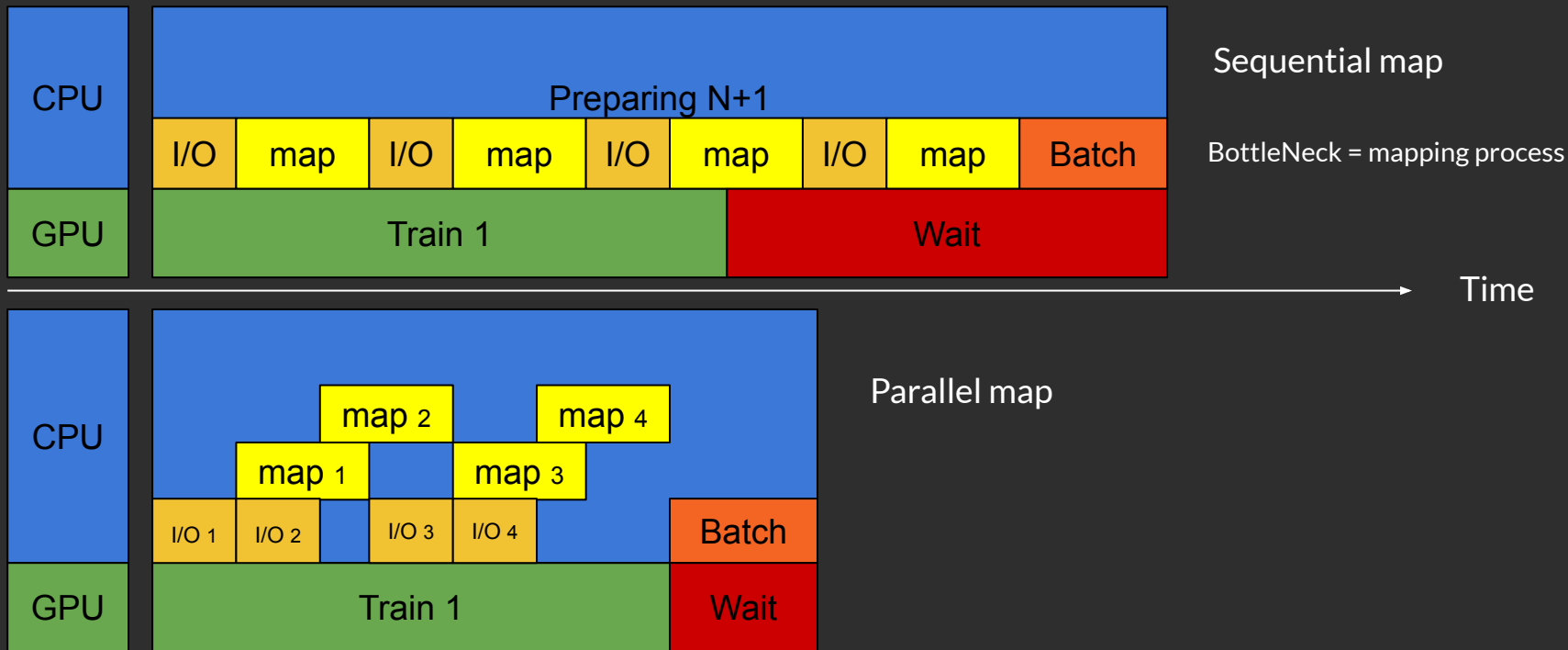
```
model = ....  
model.fit(dataset, epochs=10)
```

This allows later elements to be prepared while the current element is being processed.

- This often improves latency and throughput,
- Costs additional memory to store prefetched elements.



Parallel Transformation





Parallel Transformation

```
dataset = tf.data.Dataset.list_files('*/*.jpg')
```

```
dataset = dataset.map(preprocess, num_parallel_calls=Y)  
dataset = dataset.shuffle(dataset.cardinality()//BATCH_SIZE)  
dataset = dataset.batch(batch_size=BATCH_SIZE)  
dataset = dataset.prefetch(buffer_size=X)
```

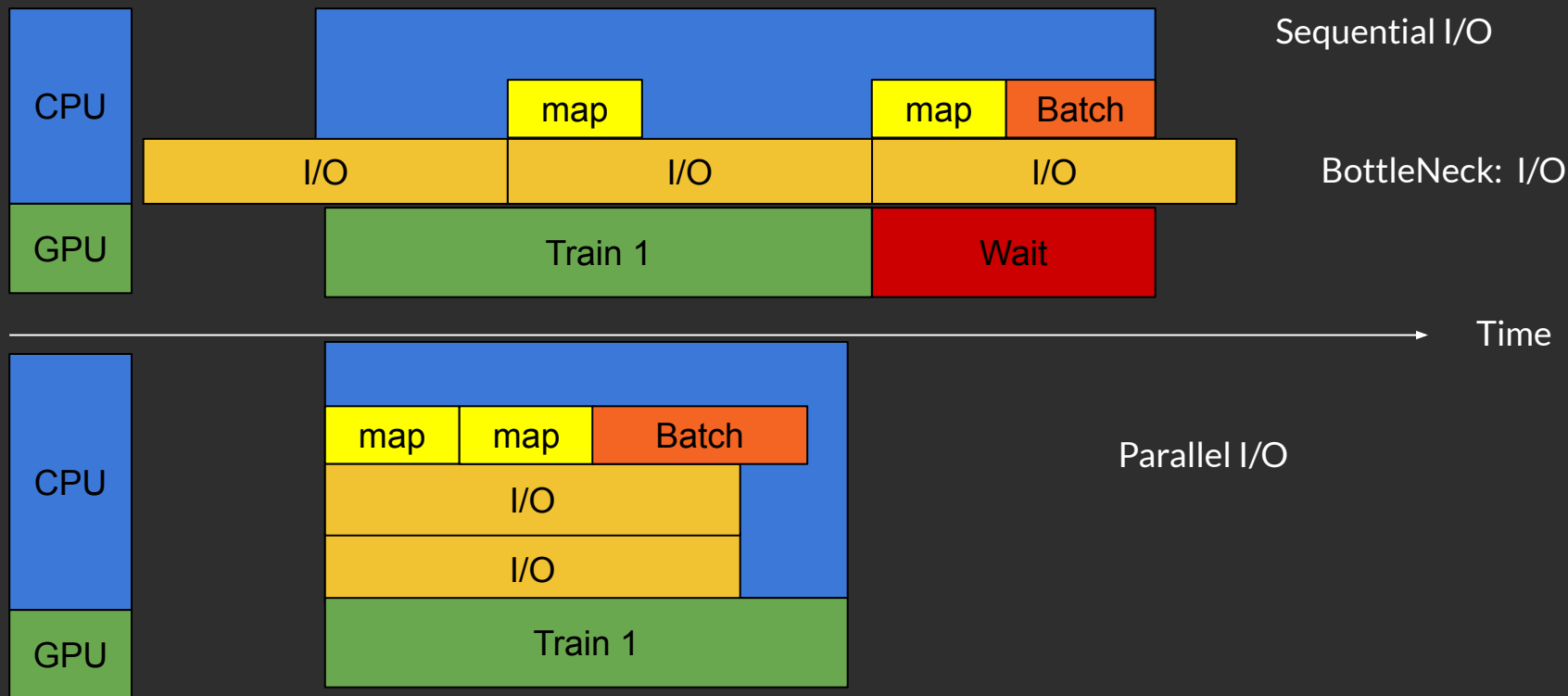
Parallel map

```
model = ....  
model.fit(dataset, epochs=10)
```

```
dataset = ....  
options = tf.data.Options()  
options.experimental_optimization_map_parallelization = True  
dataset = dataset.with_options(options)
```



Parallel Extraction





Parallel Extraction

```
dataset = tf.data.Dataset.list_files('*/*.tfrecord')
```

```
dataset = dataset.interleave(TfRecordPreprocess, num_parallel_calls=Z)
```

```
dataset = dataset.map(preprocess, num_parallel_calls=Y)
```

```
dataset = dataset.shuffle(dataset.cardinality() // BATCH_SIZE)
```

```
dataset = dataset.batch(batch_size=BATCH_SIZE)
```

```
dataset = dataset.prefetch(buffer_size=X)
```

```
model = ....
```

```
model.fit(dataset, epochs=10)
```

Parallel Extraction



Parallel Extraction

```
dataset = tf.data.Dataset.list_files('*/*.tfrecord')
```

```
dataset = dataset.interleave(TFRecordPreprocess, num_parallel_calls=Z)
```

```
dataset = dataset.map(preprocess, num_parallel_calls=Y)
```

```
dataset = dataset.shuffle(dataset.cardinality() // BATCH_SIZE)
```

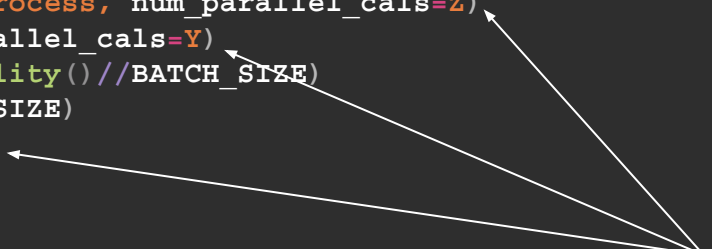
```
dataset = dataset.batch(batch_size=BATCH_SIZE)
```

```
dataset = dataset.prefetch(buffer_size=X)
```

```
model = ....
```

```
model.fit(dataset, epochs=10)
```

tf.data.experimental.AUTOTUNE





Improve single host performance

- Prefetch: Parallelize the process of fetching data and feeding data to GPU/TPU
- interleave: Parallel data extraction
- Parallel map: Parallel transformation on data



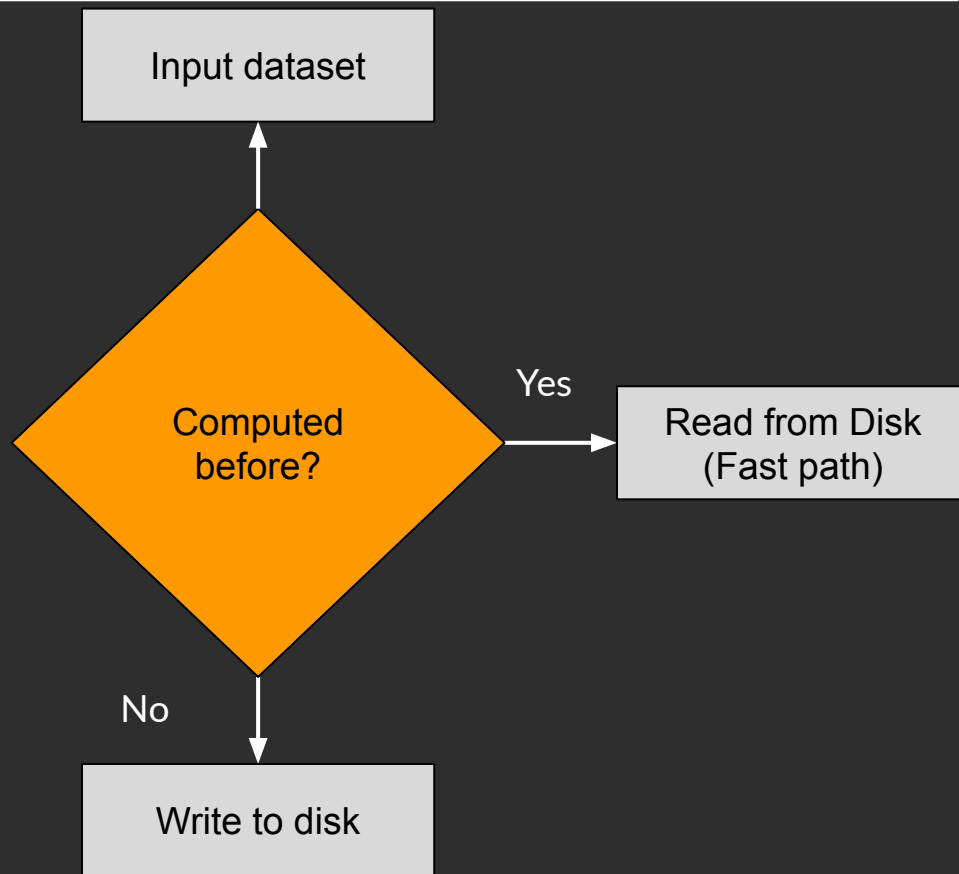
tf.snapshot

Materialize once, use many

Sometimes the input preprocessing stays the same and it's time consuming

You can store the preprocessed data on disk and read it faster next time.

- Experimenting with different model architectures
- Hyperparameter tuning





tf.snapshot


Available in TF 2.3

tf.data.experimental.snapshot

```
AUTOTUNE = tf.data.experimental.AUTOTUNE
dataset = tf.data.Dataset.TFRecordDataset('*/*.tfrecord')

dataset = dataset.map(expensive_preprocess, num_parallel_cals=AUTOTUNE)
dataset = dataset.snapshot('/path/to/snapshot_dir')
dataset = dataset.map(augment, num_parallel_cals=AUTOTUNE)
dataset = dataset.shuffle(buffer_size=1024)
dataset = dataset.batch(batch_size=BATCH_SIZE)
dataset = dataset.prefetch()

model = ....
model.fit(train_ds, epochs=10)
```



No randomization in processes before tf.snapshot. Data would be frozen after tf.snapshot process.



Let's code

**Practice#2:
Digikala Products
Color Classification**



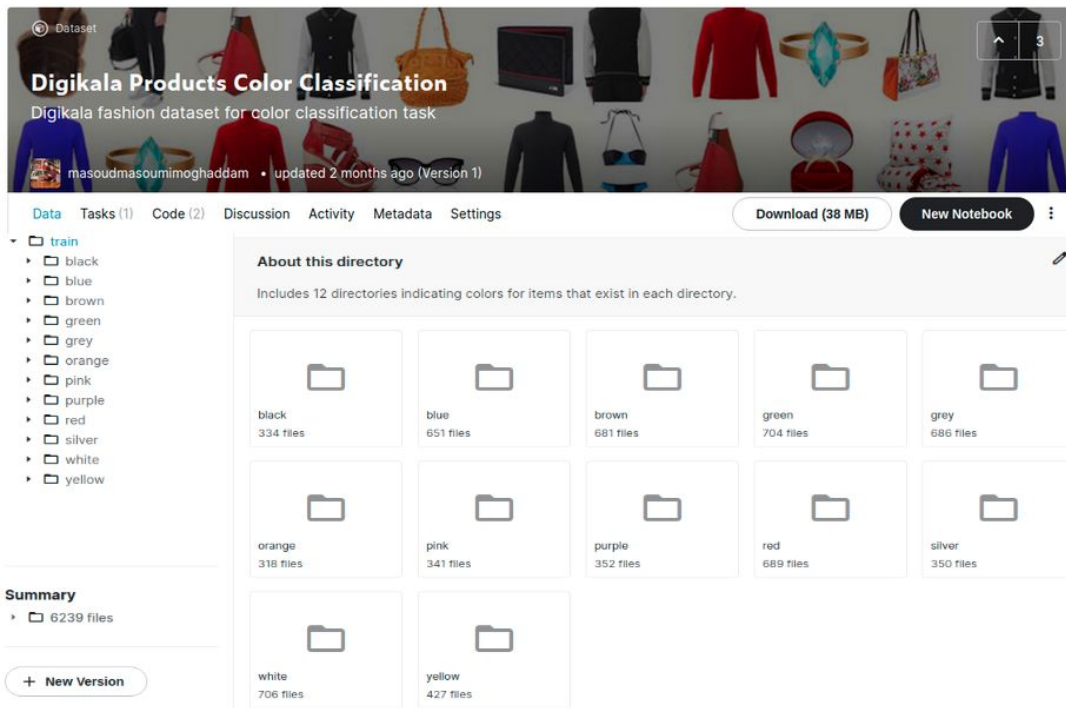
Practice #2: Digikala products color classification

In this task, we practice *tf.data* to feed batches of images to a deep model which have to classify images based on their color features.

What we learn in this notebook:

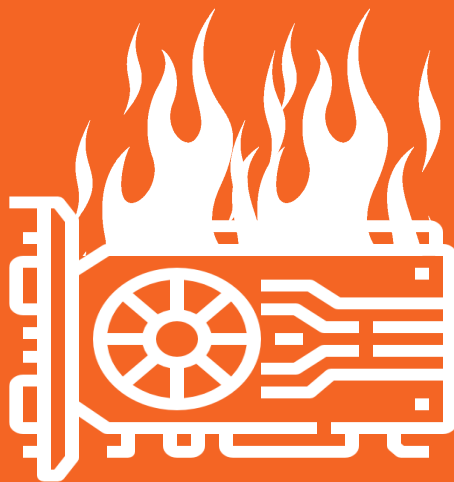
- `tf.py_function`
- `albumentation`
- Classification Evaluation Metrics
- transfer learning
- Tensorboard GradCam plot
- Tensorboard Confusion matrix
- `Tf.profile`

Click on the image to open notebook
in kaggle





Boosting performance on accelerators





tf.function

```
@tf.function
```

```
def train_step(x, y):  
    with tf.GradientTape() as tape:  
        logits = model(x, training=True)  
        loss_value = loss_fn(y, logits)  
    grads = tape.gradient(loss_value, model.trainable_weights)  
    optimizer.apply_gradients(zip(grads, model.trainable_weights))  
    train_acc_metric.update_state(y, logits)  
    return loss_value
```

```
@tf.function
```

```
def test_step(x, y):  
    val_logits = model(x, training=False)  
    val_acc_metric.update_state(y, val_logits)
```

EagerTensor to Graph Mode



EagerTensor VS Graph Mode

Graph Mode

- Platform independent (could be deployed to python-free servers, phone)
- Graph-based optimizations (Faster speed/Memory efficient)
- Must use tf.Session to see the results.
-

EagerTensor

- Simplifies the model building experience
- Quick iteration without building graphs
- Enables inspection of running models
- Dynamic models with complex flow
- Enables profiling the bottlenecks
- Not as fast as graph mode.



XLA

TensorFlow > Resources > XLA

☆☆☆☆

XLA: Optimizing Compiler for Machine Learning

XLA (Accelerated Linear Algebra) is a domain-specific compiler for linear algebra that can accelerate TensorFlow models with potentially no source code changes.

The results are improvements in speed and memory usage: e.g. in BERT [MLPerf](#) submission using 8 Volta V100 GPUs using XLA has achieved a ~7x performance improvement and ~5x batch size improvement:

```
@tf.function(jit_compile=True)
def train_step(x, y):
    with tf.GradientTape() as tape:
        logits = model(x, training=True)
        loss_value = loss_fn(y, logits)
    grads = tape.gradient(loss_value, model.trainable_weights)
    optimizer.apply_gradients(zip(grads, model.trainable_weights))
    train_acc_metric.update_state(y, logits)
    return loss_value
```

tf-nightly

```
tf.keras.backend.clear_session()
tf.config.optimizer.set_jit(True) # Start with XLA Enabled.
```

tensorflow

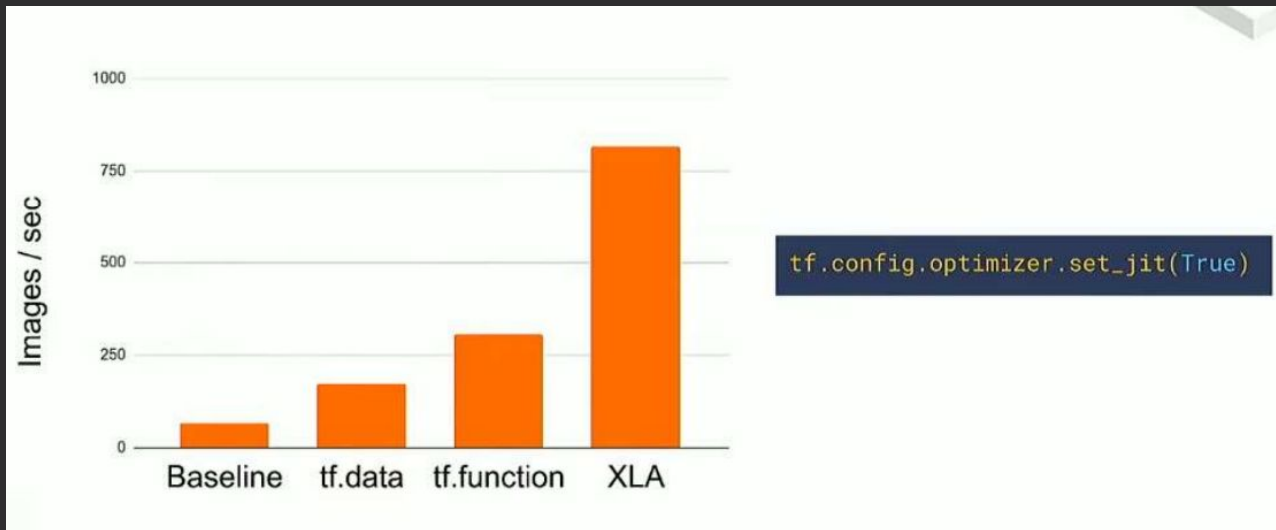
Also available for

- [JAX](#) (Python+NumPy)
- [Julia](#)
- [PyTorch](#)
- [Elixir](#)

Note: 1st epoch is gonna be slow
(compiling in the time of execution)

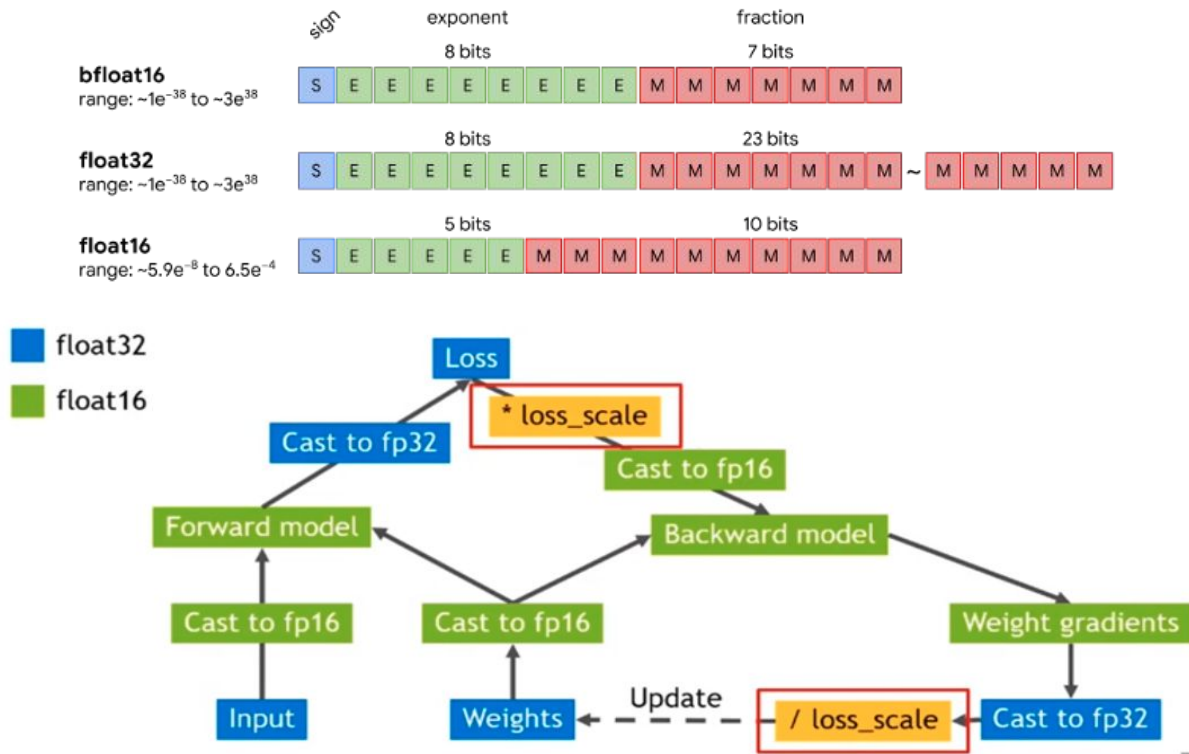


XLA Performance





Mixed Precision Strategy





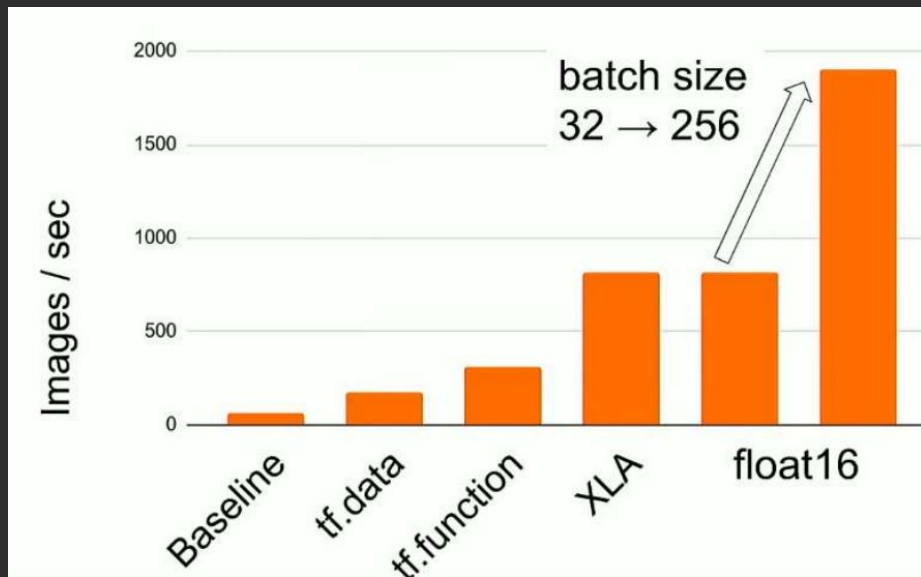
Mixed Precision Strategy

```
@tf.function
def step(features, labels):
    with tf.GradientTape() as tape:
        ...
        scales_loss = optimizer.get_scaled_loss(loss)

    fp32_grads = tape.gradient(loss, model.trainable_variables)
    fp16_grads = [tf.cast(grad, 'float16') for grad in fp32_grads]
    all_reduced_fp16_grads = tf.distribute.get_replica_context().all_reduce(
        tf.distribute.ReduceOp.SUM, fp16_grads
    )
    all_reduced_fp32_grads = [tf.cast(grad, 'float32') for grad in all_reduced_fp16_grads]
    all_reduced_fp32_grads = optimizer.get_unscaled_gradients(all_reduced_fp32_grads)
    optimizer.apply_gradients(zip(all_reduced_fp32_grads, model.trainable_variables),
                              all_reduce_sum_gradients=False)
```

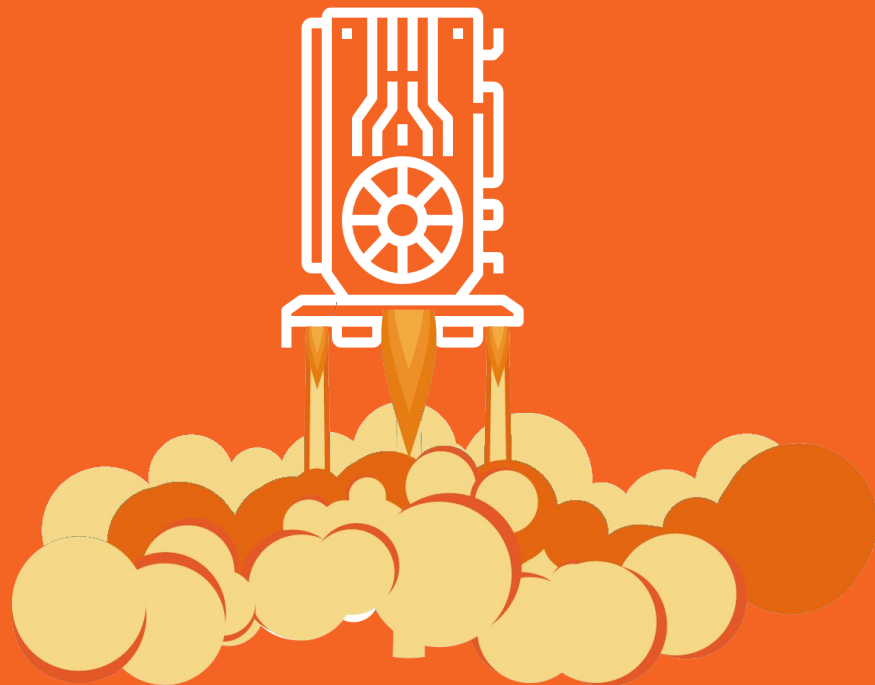


Mixed Precision Strategy

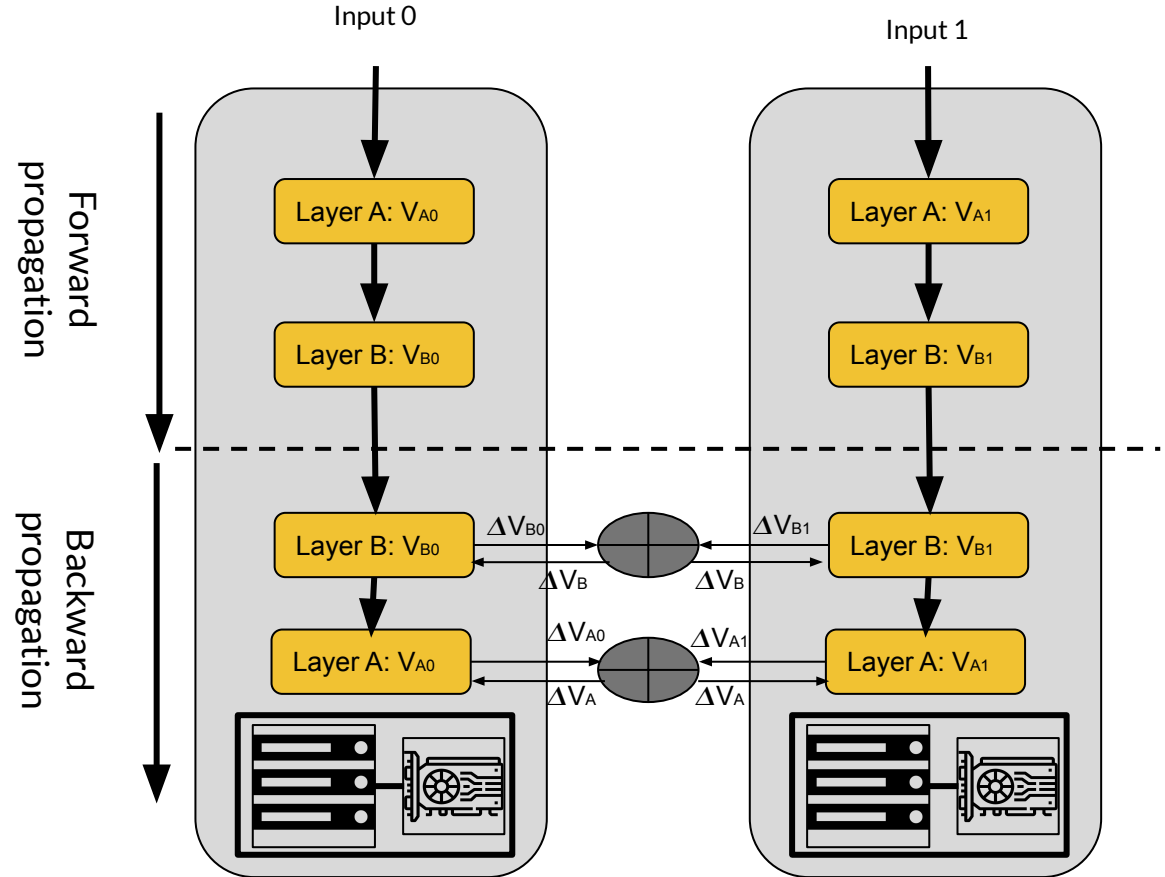




More performance !?



Synchronous Training



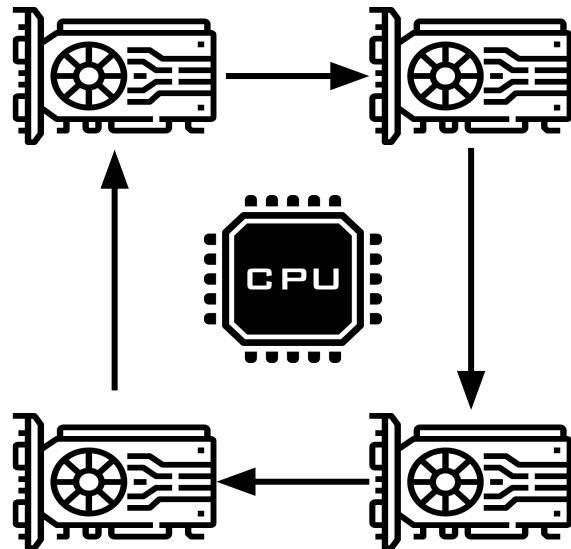


Multi-GPU Training

- replicas are run in lock-step synchronizing gradients at each step.
- All-reduce: network efficient way to aggregate gradients.

```
strategy = tf.distribute.MirroredStrategy()
strategy = tf.distribute.MirroredStrategy(devices=['gpu:0', 'gpu:1'])
strategy = tf.distribute.MirroredStrategy(
    cross_device_ops=tf.distribute.NcclAllReduce(num_packs=2)
)

with strategy.scope():
    model = tf.keras.applications.ResNet50()
    optimizer = tf.keras.optimizers.SGD(learning_rate=0.1)
    model.compile(..., optimizer=optimizer)
    model.fit(train_data, epochs=13)
```

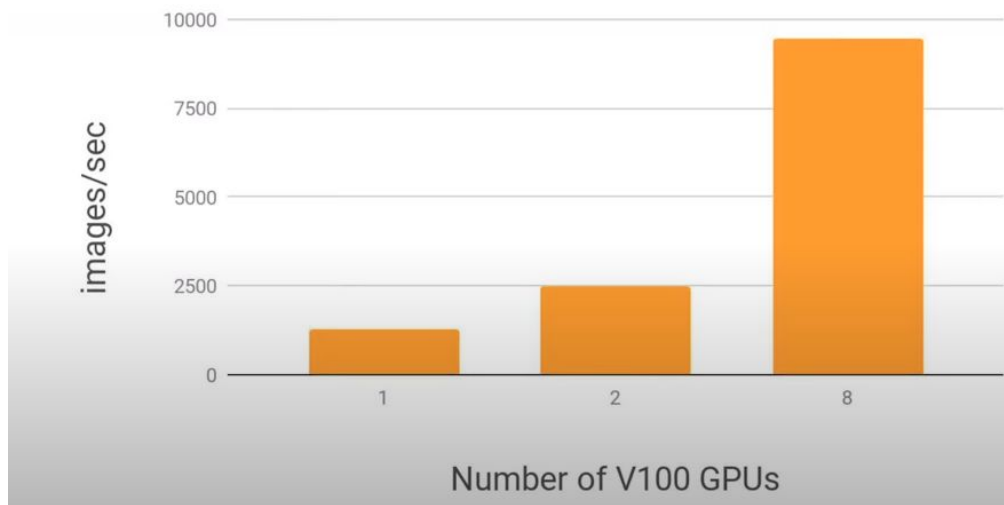




Multi-GPU Performance

Multi-GPU Performance

ResNet50 v1.5 Performance with `tf.distribute.MirroredStrategy`

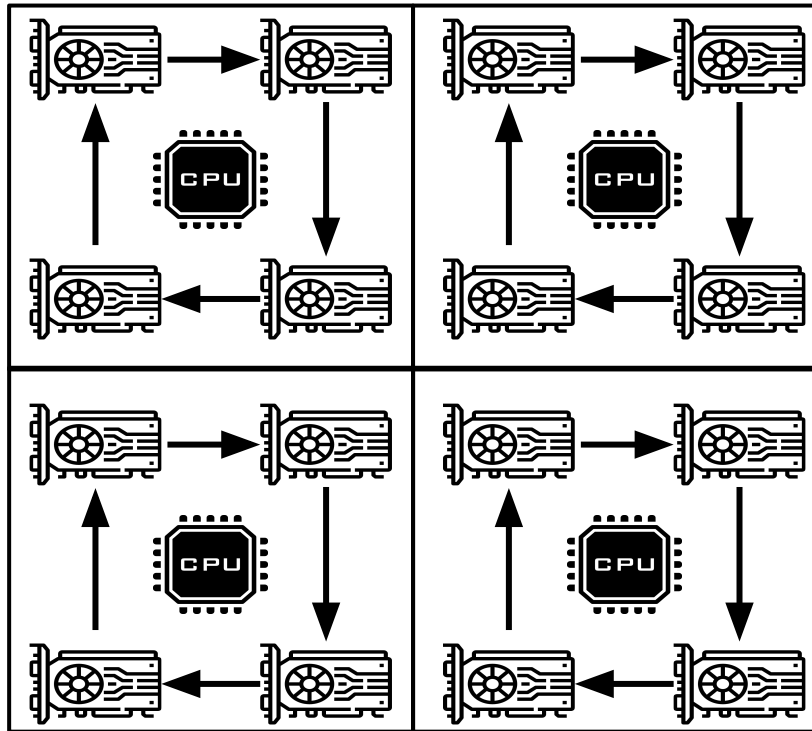




Multi-worker Synchronous Training

- Workers are run in lock-step synchronizing gradients at each step.

```
strategy =  
tf.distribute.experimental.MultiWorkerMirroredStrategy()  
strategy =  
tf.distribute.experimental.MultiWorkerMirroredStrategy(  
    tf.distribute.experimental.CommunicationNCCL  
)  
  
os.environ['TF_CONFIG'] = json.dumps({  
    'cluster': {  
        'worker': ["host1:port", "host2:port", "host3:port"]  
    },  
    'task': {"type": "worker", "index": 1}  
})
```





References

- [Many thanks to Alireza Akhavanpour and his insightful slides](#)
- [A survey on Image Data Augmentation for Deep Learning](#)
- [Albumentation github repository](#)
- [imgaug github repository](#)
- [How to apply keras ImageDataGenerator into segmentation task?](#)
- [Keras how to generate data on the fly](#)
- [Violence detection and classification on kaggle violence dataset \(use case of keras.utils.sequence\)](#)
- [Stackoverflow question: tf.data vs keras.utils.sequence performance](#)
- [Inside TensorFlow: tf.data + tf.distribute](#)
- [Scaling Tensorflow data processing with tf.data \(TF Dev Summit '20\)](#)
- [Kaggle code for image classification on Digikala products \(Use case of tf.data\)](#)
- [Tensorflow docs | tf.function tutorial](#)
- [TowardsDataScience: Eager Execution vs. Graph Execution in TensorFlow: Which is Better?](#)
- [Tensorflow docs for XLA](#)
- [Youtube tensorflow XLA tutorial](#)
- [Tensorflow docs | Mixed Precision and how to apply it in custom loop.](#)
- [NVIDIA Developer How To Series: Mixed-Precision Training](#)
- Icons from: <https://www.flaticon.com/authors/linector>



PartAI Research Center

Thank You

If you like the content, consider:

- Voting for [my code](#) in Kaggle.com
- 50 Claps for [my articles](#) in Medium.com
