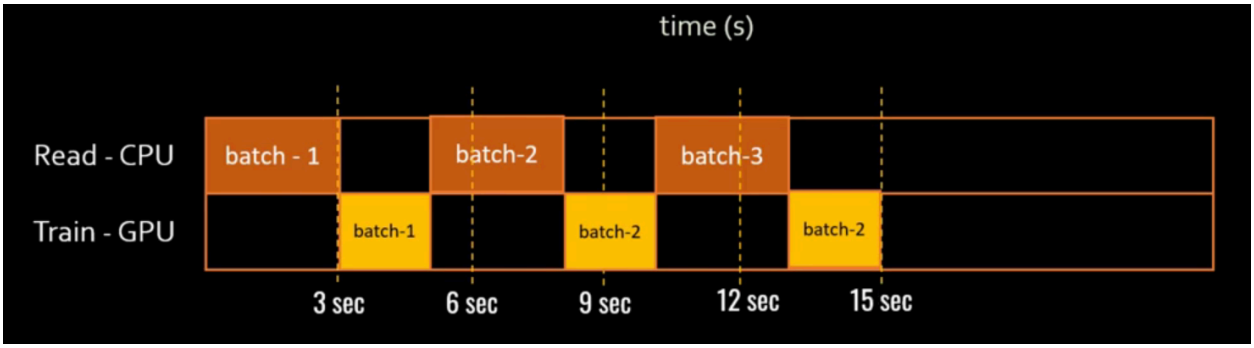
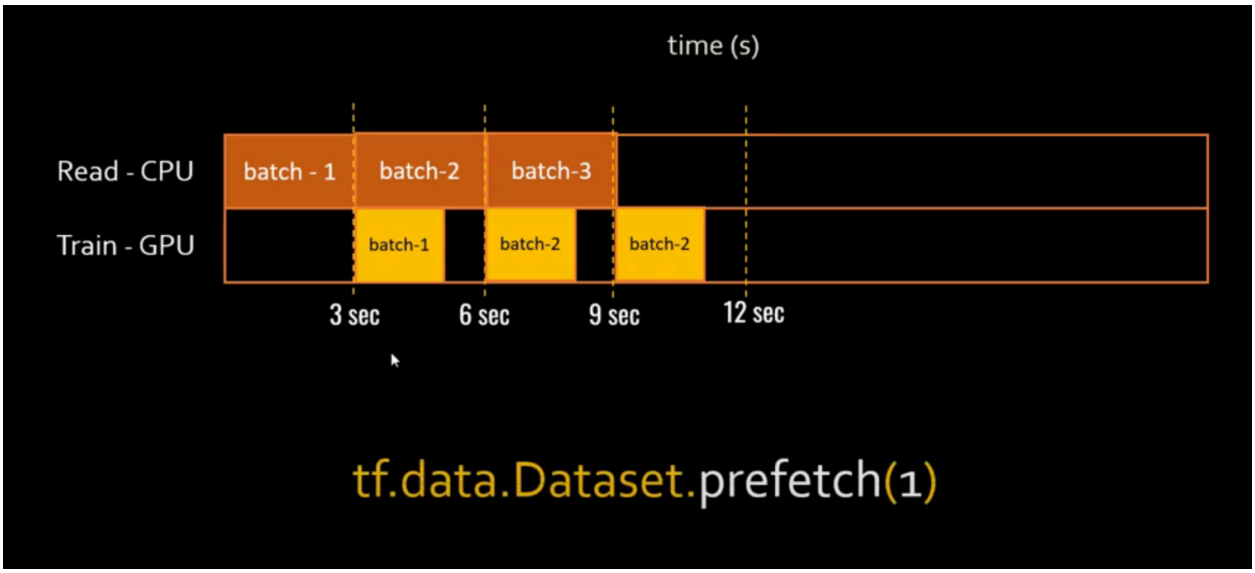
Optimize Tensorflow Pipeline Performance: Prefetch & Cache



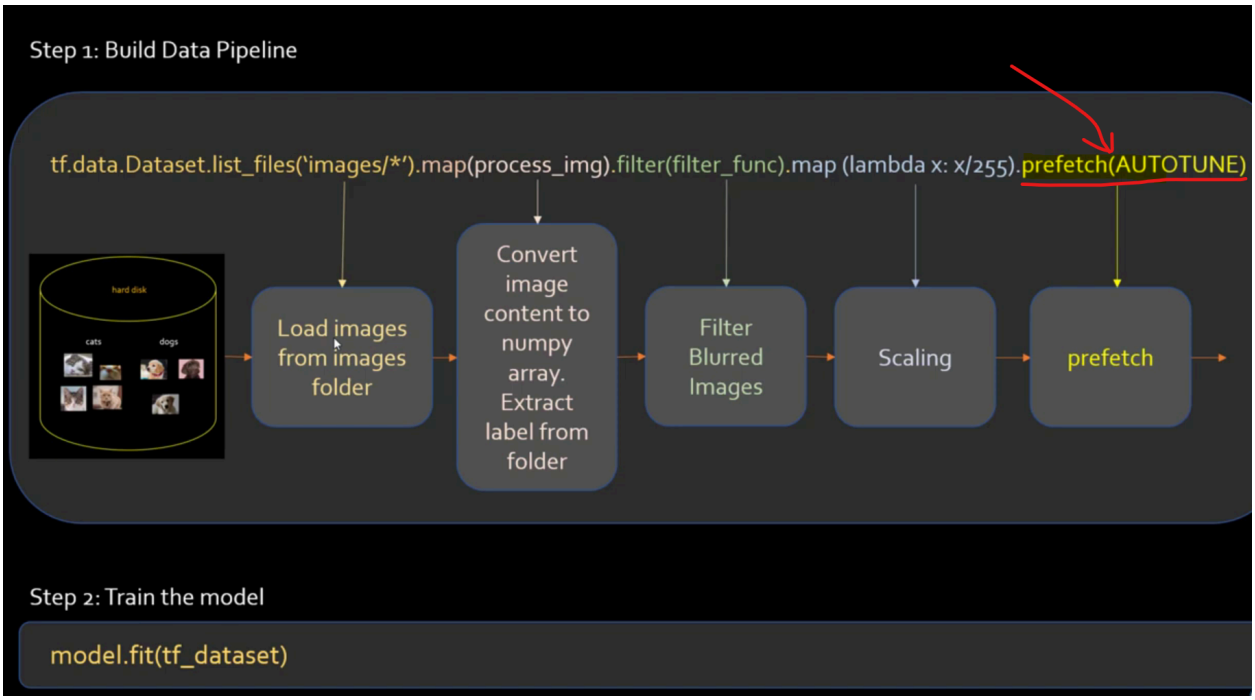
Optimized: Less time 👇



`1` → How many batches i want to prefetch?

- If data isn't ready on time, the GPU sits idle — wasting time.
- `cache()` and `prefetch()` help prepare data faster.

- Generally the number is decided by Tensorflow by `AUTONUNE` argument



Cache

- Avoids reloading data repeatedly

What is `cache()` ?

▶ Purpose:

Saves the dataset in **RAM** (or optionally to disk) **after the first epoch** — so it's **not reloaded or decoded again**.

🔁 Without `cache()` :

Each epoch loads the same data again from disk → **slow**.

💡 When to Use:

- ✅ Data fits in memory (RAM)

- ❌ Avoid it if data is too big → will crash

✅ **Example:**

```
dataset = tf.data.Dataset.from_tensor_slices(my_data)
dataset = dataset.cache()
```

Now from the **second epoch onward**, it's lightning fast.

🔁 What is `prefetch()`?

▶ Purpose:

While the GPU is **training on batch N**, the CPU is **preparing batch N+1**.

> So there's no waiting between batches — like preparing your next plate while eating the first.

💡 When to Use:

- **Always** — it's safe, no memory copying
- Use `.prefetch(tf.data.AUTOTUNE)` for best results

✅ Example:

```
dataset = dataset.prefetch(tf.data.AUTOTUNE)
```

🧠 Combining the Pipeline

Here's a full optimized data pipeline:

```
import tensorflow as tf

def preprocess(x):
    # Any resizing, normalization, etc.
    return x
```

```
dataset = tf.data.Dataset.from_tensor_slices(my_data)
dataset = dataset.map(preprocess)
dataset = dataset.shuffle(buffer_size=1000)
dataset = dataset.batch(32)
dataset = dataset.cache()                    # Cache preprocessed batches
dataset = dataset.prefetch(tf.data.AUTOTUNE)     # Prefetch while training
```

🔧 Optional: `cache()` to Disk (if data too big for RAM)

```
dataset = dataset.cache(filename='my_cache.tf-data')
```

✅ Best Practices

- Always use `prefetch(tf.data.AUTOTUNE)`
- Use `cache()` if dataset fits in memory
- Place `cache()` **after** mapping and shuffling
- Place `prefetch()` **last** in the pipeline

Optimized Pipeline Example

```
def preprocess_image(file_path):
    img = tf.io.read_file(file_path)
    img = tf.io.decode_jpeg(img, channels=3)
    return tf.image.resize(img, [256, 256]) / 255.0  # Normalize


# Build pipeline
dataset = (
    tf.data.Dataset.list_files('images/*.jpg')
    .map(preprocess_image, num_parallel_calls=tf.data.AUTOTUNE)  # Parallel
processing
    .cache()                    # Cache after preprocessing
    .shuffle(1000)                # Shuffle after caching
```

```
    .batch(32)                  # Batch
    .prefetch(tf.data.AUTOTUNE)       # Prefetch last
)
```

Common Pitfalls

- **Caching too early**: Cache *after* preprocessing but *before* shuffling.
- **Over-shuffling**: Shuffle once after caching.
- **Ignoring `AUTOTUNE`**: Always prefer `tf.data.AUTOTUNE` for dynamic tuning.