Tensorflow Input Pipeline

- If there's too much data, your system can't handle it.
- To solve this, load the data in batches
  - 👉 `tf.data.Dataset`

```
tf_dataset = tf.data.Dataset.list_files('images/*').map(process_img).filter(filter_func).map(lambda x: x/255.0)
```

1. `tf.data.Dataset.list_files('images/*')`

- **Purpose**: Creates a dataset of file paths matching the pattern.
- **Action**:
  - Scans the `images/` directory
  - Lists all files matching the wildcard pattern (all files)
- **Output**: Dataset of strings (file paths like `"images/cat.jpg"`, `"images/dog.png"`)

---

2. `.map(process_img)`

- **Purpose**: Applies a custom function to each file path.
- **Action**:
  - `process_img` should be a function that:
    1. Takes a file path as input
    2. Reads the image file
    3. Decodes it (e.g., using `tf.io.decode_image` or `tf.io.decode_jpeg`)
    4. Resizes/normalizes as needed
- **Example**:

```
def process_img(file_path):
    img = tf.io.read_file(file_path)
    img = tf.io.decode_jpeg(img, channels=3)  # RGB
    return tf.image.resize(img, [256, 256])   # Resize
```

---

3. `.filter(filter_func)`

`.filter()`: This method filters the elements of the dataset based on a provided predicate function. Only elements for which the predicate function returns `True` are kept in the dataset.

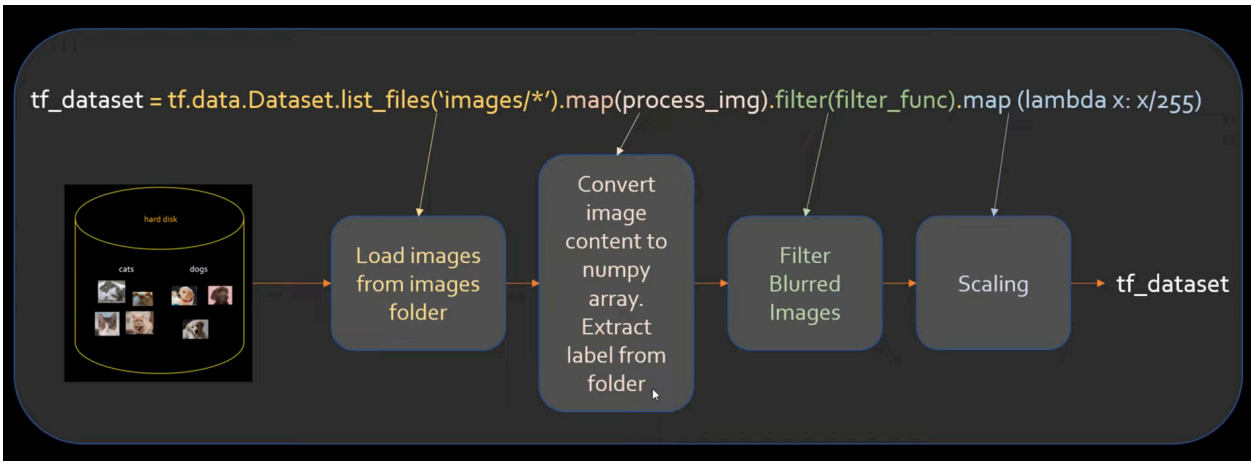
- **Purpose**: Filters out unwanted images.
- **Action**:
  - `filter_func` should return `True` (keep) or `False` (discard)
  - Common uses:
    - Remove corrupt images
    - Filter by image dimensions
- **Example**:

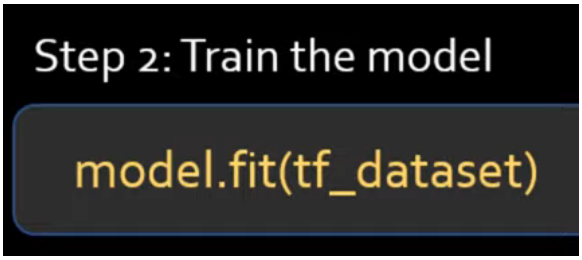
```
def filter_func(img):
    return tf.reduce_all(tf.shape(img) > 0)  # Keep non-empty images
```

---

4. `.map(lambda x: x/255)`

- **Purpose**: Normalizes pixel values to [0, 1] range.
- **Action**:
  - Divides every pixel value by 255 (assuming original range is 0-255)
  - Equivalent to: `tf.image.convert_image_dtype(img, tf.float32)`





Tensorflow input pipeline benefits

- Handle huge datasets by streaming them from disk using batching
- Apply transformations to make dataset ready for model training

Python Code

Create tf dataset object from a list:

```
daily_sales_numbers = [21, 22, -108, 31, -1, 32, 34,31]

tf_dataset = tf.data.Dataset.from_tensor_slices(daily_sales_numbers)
tf_dataset
```

```
<_TensorSliceDataset element_spec=TensorSpec(shape=(), dtype=tf.int32, name=None)>
```

Print elements in the dataset:

```
for sales in tf_dataset:
    print(sales)
```

```
tf.Tensor(21, shape=(), dtype=int32)
tf.Tensor(22, shape=(), dtype=int32)
tf.Tensor(-108, shape=(), dtype=int32)
tf.Tensor(31, shape=(), dtype=int32)
tf.Tensor(-1, shape=(), dtype=int32)
tf.Tensor(32, shape=(), dtype=int32)
tf.Tensor(34, shape=(), dtype=int32)
tf.Tensor(31, shape=(), dtype=int32)
```

```
for sales in tf_dataset:
    print(sales.numpy())
```

```
21
22
-108
31
-1
32
34
31
```

- If you don't want to use `.numpy()`, you can use **`as_numpy_iterator()`**

```
for sales in tf_dataset.as_numpy_iterator():
    print(sales)
```

Iterate through first n elements in tf dataset:

- `.take()`

```
for sales in tf_dataset.take(3):
```

```
    print(sales.numpy())
```

```
21
22
-108
```

Filter

- `.filter()`

```
tf_dataset = tf_dataset.filter(lambda x: x>0)

for sales in tf_dataset.as_numpy_iterator():
    print(sales)
```

```
21
22
31
32
34
31
```

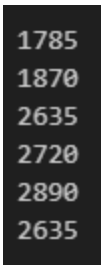
- This will filter out negative numbers

Convert sales numbers from USA dollars ($) → Indian Rupees (INR):

- `.map()`

```
tf_dataset = tf_dataset.map(lambda x: x*85)

for sales in tf_dataset.as_numpy_iterator():
    print(sales)
```

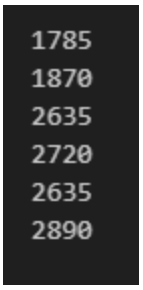


Shuffle the Elements:

- `shuffle()`

```
tf_dataset = tf_dataset.shuffle(2)

for sales in tf_dataset.as_numpy_iterator():
    print(sales)
```



`2` → **Buffer Size:** The `buffer_size` parameter specifies how many elements from the dataset will be randomly sampled from to create the shuffled dataset.

How ds.shuffle() works

`dataset.shuffle(buffer_size=3)` will allocate a buffer of size 3 for picking random entries. This buffer will be connected to the source dataset. We could image it like this:

```
Random buffer
   |
   |   Source dataset where all other elements live
   |          |
   ↓          ↓
[1,2,3] <= [4,5,6]
```

Let's assume that entry `2` was taken from the random buffer. Free space is filled by the next element from the source buffer, that is `4`:

```
2 <= [1,3,4] <= [5,6]
```

We continue reading till nothing is left:

```
1 <= [3,4,5] <= [6]
5 <= [3,4,6] <= []
3 <= [4,6]   <= []
6 <= [4]     <= []
4 <= []      <= []
```

https://stackoverflow.com/questions/53514495/what-does-batch-repeat-and-shuffle-do-with-tensorflow-dataset

Batching

`.batch()`

```
for sales_batch in tf_dataset.batch(2):
    print(sales_batch.numpy())
```

```
[1584 2232]
[2304 2448]
[2232 1512]
```

`.batch(2)` → Batch of size 2

Perform all of the above operations in one shot

Create dataset:

```
tf_dataset = tf.data.Dataset.from_tensor_slices(daily_sales_numbers)
```

```
tf_dataset = tf_dataset.filter(lambda x: x>0).map(lambda y: y*72).shuffle(2).batch(2)

for sales in tf_dataset.as_numpy_iterator():
    print(sales)
```

Images

```
images_ds = tf.data.Dataset.list_files('flower_photos/flower_photos/*/*', shuffle=False)
```

Print 3 paths:

```
for i in images_ds.take(3):
    print(i.numpy())
```

```
b'flower_photos\\flower_photos\\daisy\\100080576_f52e8ee070_n.jpg'
b'flower_photos\\flower_photos\\daisy\\10140303196_b88d3d6cec.jpg'
b'flower_photos\\flower_photos\\daisy\\10172379554_b296050f82_n.jpg'
```

- It has stored the image paths
- It has not yet read the images

```
image_count = len(images_ds)
image_count
```

```
3670
```

Shuffle:

```
images_ds = images_ds.shuffle(200)
for file in images_ds.take(3):
    print(file.numpy())
```

Create class names:

```
class_names = ["daisy","dandelion", "roses", "sunflowers", "tulips"]
```

```
b'flower_photos\\flower_photos\\daisy\\176375506_201859bb92_m.jpg'
b'flower_photos\\flower_photos\\daisy\\1286274236_1d7ac84efb_n.jpg'
b'flower_photos\\flower_photos\\daisy\\12891819633_e4c82b51e8.jpg'
```

Split train & test data:

```
train_size = int(image_count*0.8)
train_ds = images_ds.take(train_size)
test_ds = images_ds.skip(train_size)
```

`int()` → Whole number

`skip()` → Opposite of take

- `train_ds` gets the first 80% of the dataset.
- `test_ds` gets the remaining 20%, starting **right after the last element of** `train_ds`.

```
len(train_ds)
```

2936

len(test_ds)

734

Get Labels

```
a= 'flower_photos\\flower_photos\\daisy\\176375506_201859bb92_m.jpg'
a.split('\\')
```

```
['flower_photos', 'flower_photos', 'daisy', '176375506_201859bb92_m.jpg']
```

```
a= 'flower_photos\\flower_photos\\daisy\\176375506_201859bb92_m.jpg'
a.split('\\')[-2]
```

```
'daisy'
```

💡 We performed split on a string.

BUT **'SymbolicTensor' object has no attribute 'split'**

Make function to read label & process image:

Get Label:

```
def get_label(file_path):
    import os
    return tf.strings.split(file_path, os.path.sep)[-2]
```

```
train_ds.map(get_label)
```

```
<_MapDataset element_spec=TensorSpec(shape=(), dtype=tf.string, name=None)>
```

```
for i in train_ds.map(get_label):
    print(i.numpy())
```

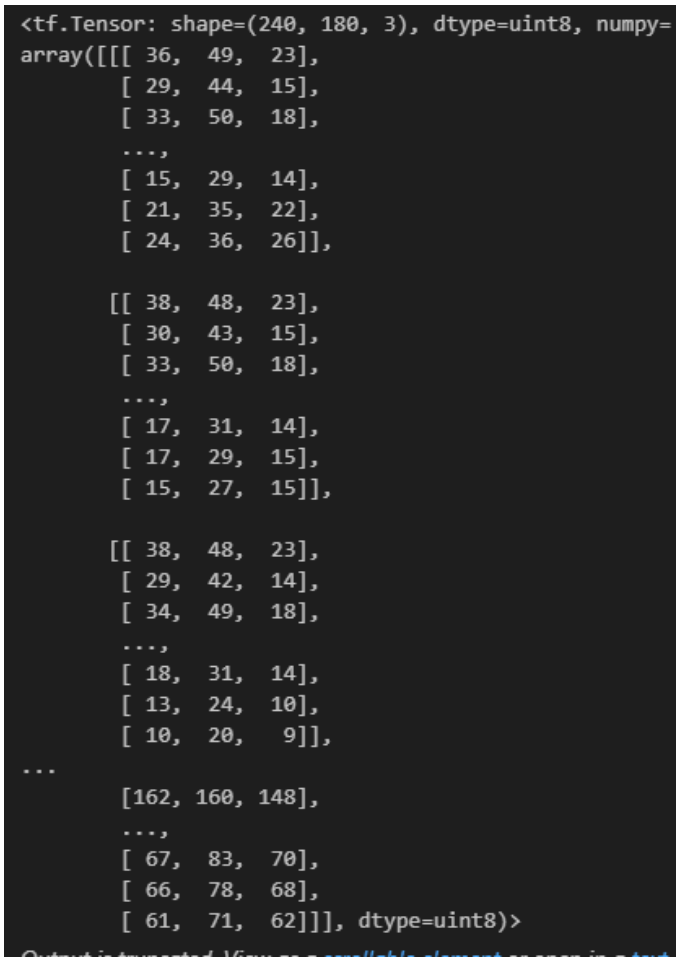
```
b'daisy'
b'daisy'
b'daisy'
b'daisy'
b'daisy'
b'daisy'
b'daisy'
b'daisy'
b'daisy'
b'daisy'
b'daisy'
```

Process Image:

```
def process_image(file_path):
    label = get_label(file_path)
    img = tf.io.read_file(file_path) # load the raw data from the file as a string
    img = tf.image.decode_jpeg(img)
    img = tf.image.resize(img, [128, 128])
    return img, label
```

`tf.io.read_file(file_path)` → Reads the image file

`tf.image.decode_jpeg()` → Converts the above image into array 👇



```
<tf.Tensor: shape=(240, 180, 3), dtype=uint8, numpy=
array([[[ 36,  49,  23],
        [ 29,  44,  15],
        [ 33,  50,  18],
        ...,
        [ 15,  29,  14],
        [ 21,  35,  22],
        [ 24,  36,  26]],

       [[ 38,  48,  23],
        [ 30,  43,  15],
        [ 33,  50,  18],
        ...,
        [ 17,  31,  14],
        [ 17,  29,  15],
        [ 15,  27,  15]],

       [[ 38,  48,  23],
        [ 29,  42,  14],
        [ 34,  49,  18],
        ...,
        [ 18,  31,  14],
        [ 13,  24,  10],
        [ 10,  20,   9]],
...
        [162, 160, 148],
        ...,
        [ 67,  83,  70],
        [ 66,  78,  68],
        [ 61,  71,  62]]], dtype=uint8)>
```

`tf.image.resize(img, [128, 128])` → Resizes the image

Apply the above function to train & test data:

```
train_ds = train_ds.map(process_image)
test_ds = test_ds.map(process_image)
```

Scale:

```
def scale(image, label):
    return image/255, label
```

Apply:

```
train_ds = train_ds.map(scale)
```

```
for image, label in train_ds.take(5):
    print("****Image: ",image.numpy()[0][0])
    print("****Label: ",label.numpy())
```

```
****Image:  [0.01568628 0.04362745 0.        ]
****Label:  b'daisy'
****Image:  [0. 0. 0.]
****Label:  b'daisy'
****Image:  [0.67075676 0.6703891  0.6490043 ]
****Label:  b'daisy'
****Image:  [0.08952206 0.00514706 0.03780637]
****Label:  b'daisy'
****Image:  [0.39421457 0.445195   0.16355005]
****Label:  b'daisy'
```