Import Dependencies

In [1]:
```
import numpy as np
import os
import tensorflow as tf
from tensorflow import keras
import re
from tqdm import tqdm
import sys
import random
from tqdm import tqdm
```

Define Filepaths for NPZ Files and TF Record Directory

In [2]:
```
# Define filepath for npz files
npz_filepath = r'T:\TensorFlow_Data\SPNGTOP_LB75_1.5ATR'

# Define function to check if record directory exists. If not, create it
def create_TFrecord_directory(filepath):

    tf_record_filepath = f'{filepath}_TFRECORDS'

    # If file exists and has data in it, throw exception
    if os.path.exists(tf_record_filepath) and os.listdir(tf_record_filepath) != []:
        raise Exception("WARNING: You are trying to overwrite a current directory!")
    else:
        try:
            os.makedirs(tf_record_filepath)
        except FileExistsError:
            pass
    return(tf_record_filepath)

# Define the directory where the TF_record will be stored
tf_record_filepath = create_TFrecord_directory(npz_filepath)
```

Determine Number of TF Record Files to Make

In [3]:
```
# Tensorflow recommends that TF record files be between 100 and 200 MB. Aim for 150 MB

# Return the average file size in a directory in bytes
def get_directory_avg_size(filepath):

        total_size = 0
        start_path = filepath
        for path, dirs, files in os.walk(start_path):
            for f in files:
                fp = os.path.join(path, f)
```

```
                total_size += os.path.getsize(fp)
        avg_size = total_size / len(os.listdir(filepath))
        return(int(avg_size))

# Define how many GAF examples per TF_record file and calculate number of TF_record fi
# Shard density = 150 MB / avg file size. Record size appears to be about 2 times as l
shard_density = (150 * 10**6) / (2 * get_directory_avg_size(npz_filepath))

# Round to the hundreths place
shard_density = int(np.round(shard_density, -2))

# Record shard_density to data info file
def record_shard_density(shard_density):

    global npz_filepath
    import json

    # reading the data from the file
    with open(f'{npz_filepath}.txt') as f:
        data = f.read()

    # Convert to dictionary
    stats_dictionary = json.loads(data)

    # Update with shard_density info
    stats_dictionary.update({"TFRecord Shard Density": shard_density})

    # Serializing json
    json_object = json.dumps(stats_dictionary, indent=4)

    # Rewrite Original file
    with open(f'{npz_filepath}.txt', "w") as outfile:
        outfile.write(json_object)

record_shard_density(shard_density)

print(f'Number of TF_Records is: {int(np.ceil(len(os.listdir(npz_filepath)) / shard_de
```

```
Number of TF_Records is: 116
```

Make Shuffled List of all NPZ Files in Directory

In [5]:

```
# Generate list of trade examples with a full filepath
def make_full_paths(training_path):

    examples = os.listdir(training_path)

    full_paths = [os.path.join(training_path, example) for example in examples]

    return(full_paths)

# Get list of files in the filepath and randomize
trade_examples = make_full_paths(npz_filepath)
random.seed(2) # Set seed & randomly shuffle the names
random.shuffle(trade_examples)
```

Define Standard TF Helper Functions and Custom Saving and Parsing Functions

In [6]:
```
### Define functions for loading and working with the original npz GAF files ###

# Function to load a GAF npz file
def load_GAF(filepath):

    # Load the file
    with np.load(filepath) as data:

        # Parse according to dictionary keys
        CLOSE = data['CLOSE']
        UPPER_WICK = data['UPPER_WICK']
        LOWER_WICK = data['LOWER_WICK']
        REAL_BODY = data['REAL_BODY']
        VOL = data['VOL']

        # Stack the GAFs into one single tensor
        stacked_GAF = np.stack((CLOSE, UPPER_WICK, LOWER_WICK, REAL_BODY, VOL))

    return (stacked_GAF)

# Function to get the lookback_period of the GAF file to pass to TF record file
def get_GAF_lookback_period(filepath):

    file = random.choice(os.listdir(filepath))

    # Load GAF and get the shape
    GAF = load_GAF(os.path.join(filepath, file))
    dimensions = GAF.shape

    # Raise exception if the GAF isn't float32. This will mess up the tf.record decodi
    if GAF.dtype != np.float32:
        raise Exception('GAF is not of dtype float32')

    # Raise exceptions if there aren't 5 channels, if the GAF isn't square, or if a co
    if dimensions[0] != 5:
        raise Exception('GAF does not have the 5 required layers for CULR + Volume')

    if dimensions[1] != dimensions[2]:
        raise Exception('GAF is not a square matrix')

    if dimensions[3] != 2:
        raise Exception('GAF does not have the 2 required color channels')

    # Return the square dimension of the GAF
    return(dimensions[1])

# Define function to figure out if trade was successful or not based on the filename
def decode_trade_label(filepath):

    # Figure out trade labels. Search for successful/unsuccessful in filename. Return
    if re.search('Successful', filepath):
        label = 1 # 1 = successful buy
```

```
    if re.search('Unsuccessful', filepath):
        label = 0 # 0 = unsuccessful buy

    return(label)



### Define functions for creating the TF Record ###

# Convert values to compatible tf.Example types
def _bytes_feature(value):
    # Returns a bytes_list from a string/byte

    if isinstance(value, type(tf.constant(0))):
        value = value.numpy() # BytesList wont unpack a string from an EagerTensor

    return(tf.train.Feature(bytes_list = tf.train.BytesList(value = [value])))

def _float_feature(value):
    # Returns a float_list from a float/double

    return(tf.train.Feature(float_list = tf.train.FloatList(value = [value])))

def _int64_feature(value):
    # Returns a int64_list from a bool/enum/int/unit

    return(tf.train.Feature(int64_list = tf.train.Int64List(value = [value])))

# Create features dictionary for TF Record format
def image_example(image, label, GAF_breadth):

    feature = {
        'image_raw' : _bytes_feature(image.tobytes()),
        'label' : _int64_feature(label),
        'GAF_square_dimension' : _int64_feature(GAF_breadth)
    }

    return(tf.train.Example(features = tf.train.Features(feature = feature)))



### Define functions for loading and reading the TF Record ###

# Define function to parse the dictionary values of the record and return them
def parse_record(record):

    name_to_features = {
        'image_raw' : tf.io.FixedLenFeature([], tf.string),
        'label' : tf.io.FixedLenFeature([], tf.int64),
        'GAF_square_dimension' : tf.io.FixedLenFeature([], tf.int64)
    }

    return(tf.io.parse_single_example(record, name_to_features))

# Define function to decode the parsed records back into their native data type
def decode_record(record):

    image = tf.io.decode_raw(
        record['image_raw'], out_type = tf.float32, little_endian = True, fixed_length
    )
```

```
    label = record['label']
    GAF_square_dimension = record['GAF_square_dimension']

    # Assumes that there will always be 5 layers (Close, Upper_Wick, Lower_Wick, Real_
    # Also assumes that there will always be only 2 color channels. Therefore, these d
    image = tf.reshape(image, (5, GAF_square_dimension, GAF_square_dimension, 2))

    return (image, label)


# Reference: www.machinelearningmindset.com/tfrecords-for-tensorflow/
```

In [7]:

```
# Get the lookback period that the GAF was created with
lookback_period = get_GAF_lookback_period(npz_filepath)


# Create the TFRecord files with shard_density examples per file
for batch_offset in tqdm(range(0, len(trade_examples), shard_density)):

    # Isolate the subbatch of trade example filepaths
    batch = trade_examples[batch_offset : batch_offset + shard_density]

    # Create the writer with a unique filename within the TF Record directory
    with tf.io.TFRecordWriter(rf'{tf_record_filepath}\shard_{int(batch_offset/shard_de

        for example in batch:

            image = load_GAF(example)
            label = np.int64(decode_trade_label(example))

            # Assumes that there will always be 5 layers (Close, Upper_Wick, Lower_Wic
            # Also assumes that there will always be only 2 color channels. Therefore,
            tf_example = image_example(image, label, lookback_period)

            writer.write(tf_example.SerializeToString())

# Define function to compare the size of two directories
def determine_storage_savings(np_path, tf_path):

    def get_directory_size(filepath):

        total_size = 0
        start_path = filepath
        for path, dirs, files in os.walk(start_path):
            for f in files:
                fp = os.path.join(path, f)
                total_size += os.path.getsize(fp)
        return(total_size)

    np_size = get_directory_size(np_path)
    tf_size = get_directory_size(tf_path)

    print(f'TF records are {np.round(np_size / tf_size, 2)} smaller than original npz

# Compare the npz directory size to the TF record directory size
determine_storage_savings(npz_filepath, tf_record_filepath)
```

```
100%|██████████| 116/116 [36:57<00:00, 19.12s/it]
```

```
TF records are 0.57 smaller than original npz files
```

Load Records into TF Dataset and Print One Example

In [ ]:
```
train_dataset = tf.data.TFRecordDataset(train_records)
train_dataset = train_dataset.shuffle(len(train_records))
train_dataset = train_dataset.map(parse_record, num_parallel_calls = 4)
train_dataset = train_dataset.map(decode_record, num_parallel_calls = 4)
train_dataset = train_dataset.batch(BATCH_SIZE = 20)
train_dataset = train_dataset.prefetch(1)
```