In [ ]:

```
import torch
from torch import nn, einsum
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torchvision.transforms import v2

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import numpy as np


import einops
from einops import rearrange, repeat
from einops.layers.torch import Rearrange

from pyspark.sql import SparkSession
```

Creating a spark dataframe from the parquet files

In [2]:

```
# Creating a spark session
spark = SparkSession.builder \
    .appName("DatasetCreator") \
    .master('local[*]') \
    .config("spark.driver.memory", "15g") \
    .getOrCreate()

# Loading the parquet files from the directory
parquet_files_path = "/kaggle/input/regress"
df = spark.read.parquet(parquet_files_path)
```

```
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLeve
l).
24/03/14 15:01:31 WARN NativeCodeLoader: Unable to load native-hadoop library for your pl
atform... using builtin-java classes where applicable
```

In [3]:

```
# Limiting the number of rows to 10000
df = df.limit(10000)
```

In [4]:

```
#Sampling random 1000 rows from the dataset
df = df.sample(withReplacement=False, fraction=0.1)
```

In [5]:

```
#Converting to pandas dataframe
df = df.toPandas()
```

In [6]:

```
spark.stop()
```

Processing the Data :-

1. Dividing the data into train and test sets

2. Creating a pytorch Dataset and a Dataloader

Finding mean and variance

In [7]:

```
images = torch.stack([torch.tensor(img) for img in df['X_jet']], dim=0)
```

In [8]:

```
mean = images.mean(dim=(0, 2, 3))
std = images.std(dim=(0,2,3))
```

In [9]:

```
del images
```

In [10]:

```
X = df['X_jet']
y = df['m']

scaler = StandardScaler()
y = scaler.fit_transform(np.array(y).reshape(-1, 1))

# Splitting the data into train and test sets with 80% for training and 20% for testing
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42
)
```

In [11]:

```
#Creating a custom Pytorch Dataset
class RegressDataset(Dataset):
    def __init__(self, X, y, transform=False):
        self.X = X
        self.y = y
        self.transform = transform
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        img = np.array(self.X.iloc[idx])
        img = torch.tensor(img, dtype=torch.float32)
        img = v2.Normalize(mean=mean, std=std)(img)
        y = torch.tensor(self.y[idx], dtype=torch.float32)
        return img, y

# Train and Test pytorch Datasets
train_dataset = RegressDataset(X_train, y_train)
test_dataset = RegressDataset(X_test, y_test)

#Defining the batch size
BATCH_SIZE = 32

# Train and test pytorch Dataloaders
train_dataloader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=BATCH_SIZE,  shuffle=False)
```

Model Building

This paper https://arxiv.org/abs/2103.11886 notes that ViT struggles to attend at greater depths (past 12 layers), and suggests mixing the attention of each head post-softmax as a solution, dubbed Re-attention. The below solution is the implementation of the above paper to build DeepVit or DeepVisionTransoformer for better performance

In [ ]:

```
class FeedForward(nn.Module):
    def __init__(self, dim, hidden_dim, dropout = 0.):
        super().__init__()
        self.net = nn.Sequential(
            nn.LayerNorm(dim),
            nn.Linear(dim, hidden_dim),
            nn.GELU(),
            nn.Dropout(dropout),
            nn.Linear(hidden_dim, dim),
            nn.Dropout(dropout)
        )
    def forward(self, x):
        return self.net(x)

class Attention(nn.Module):
    def __init__(self, dim, heads = 8, dim_head = 64, dropout = 0.):
        super().__init__()
        inner_dim = dim_head *  heads
        self.heads = heads
        self.scale = dim_head ** -0.5

        self.norm = nn.LayerNorm(dim)
        self.to_qkv = nn.Linear(dim, inner_dim * 3, bias = False)

        self.dropout = nn.Dropout(dropout)

        self.reattn_weights = nn.Parameter(torch.randn(heads, heads))

        self.reattn_norm = nn.Sequential(
            Rearrange('b h i j -> b i j h'),
            nn.LayerNorm(heads),
            Rearrange('b i j h -> b h i j')
        )

        self.to_out = nn.Sequential(
            nn.Linear(inner_dim, dim),
            nn.Dropout(dropout)
        )

    def forward(self, x):
        b, n, _, h = *x.shape, self.heads
        x = self.norm(x)

        qkv = self.to_qkv(x).chunk(3, dim = -1)
        q, k, v = map(lambda t: rearrange(t, 'b n (h d) -> b h n d', h = h), qkv)

        # attention

        dots = einsum('b h i d, b h j d -> b h i j', q, k) * self.scale
        attn = dots.softmax(dim=-1)
        attn = self.dropout(attn)

        # re-attention

        attn = einsum('b h i j, h g -> b g i j', attn, self.reattn_weights)
        attn = self.reattn_norm(attn)

        # aggregate and out

        out = einsum('b h i j, b h j d -> b h i d', attn, v)
        out = rearrange(out, 'b h n d -> b n (h d)')
        out =  self.to_out(out)
        return out

class Transformer(nn.Module):
    def __init__(self, dim, depth, heads, dim_head, mlp_dim, dropout = 0.):
        super().__init__()
        self.layers = nn.ModuleList([])
        for _ in range(depth):
            self.layers.append(nn.ModuleList([
                Attention(dim, heads = heads, dim_head = dim_head, dropout = dropout),
                FeedForward(dim, mlp_dim, dropout = dropout)
```

```
            ]))
    def forward(self, x):
        for attn, ff in self.layers:
            x = attn(x) + x
            x = ff(x) + x
        return x

class DeepViT(nn.Module):
    def __init__(self, *, image_size, patch_size, num_classes, dim, depth, heads, mlp_di
m, pool = 'cls', channels = 3, dim_head = 64, dropout = 0., emb_dropout = 0.):
        super().__init__()
        assert image_size % patch_size == 0, 'Image dimensions must be divisible by the
patch size.'
        num_patches = (image_size // patch_size) ** 2
        patch_dim = channels * patch_size ** 2
        assert pool in {'cls', 'mean'}, 'pool type must be either cls (cls token) or mea
n (mean pooling)'

        self.to_patch_embedding = nn.Sequential(
            Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1 = patch_size, p2 = pa
tch_size),
            nn.LayerNorm(patch_dim),
            nn.Linear(patch_dim, dim),
            nn.LayerNorm(dim)
        )

        self.pos_embedding = nn.Parameter(torch.randn(1, num_patches + 1, dim))
        self.cls_token = nn.Parameter(torch.randn(1, 1, dim))
        self.dropout = nn.Dropout(emb_dropout)

        self.transformer = Transformer(dim, depth, heads, dim_head, mlp_dim, dropout)

        self.pool = pool
        self.to_latent = nn.Identity()

        self.mlp_head = nn.Sequential(
            nn.LayerNorm(dim),
            nn.Linear(dim, num_classes)
        )

    def forward(self, img):
        x = self.to_patch_embedding(img)
        b, n, _ = x.shape

        cls_tokens = repeat(self.cls_token, '() n d -> b n d', b = b)
        x = torch.cat((cls_tokens, x), dim=1)
        x += self.pos_embedding[:, :(n + 1)]
        x = self.dropout(x)

        x = self.transformer(x)

        x = x.mean(dim = 1) if self.pool == 'mean' else x[:, 0]

        x = self.to_latent(x)
        return self.mlp_head(x)
```

In [15]:

```
model = DeepViT(
    image_size = 125,
    patch_size = 25,
    num_classes = 1,
    dim = 1024,
    depth = 12,
    heads = 16,
    channels = 8,
    mlp_dim = 2048,
    dropout = 0.1,
    emb_dropout = 0.1
)
```

In [16]:

```
device = 'cuda' if torch.cuda.is_available else 'cpu'
```

In [17]:

```
model.to(device)
```

Out[17]:

```
DeepViT(
  (to_patch_embedding): Sequential(
    (0): Rearrange('b c (h p1) (w p2) -> b (h w) (p1 p2 c)', p1=25, p2=25)
    (1): LayerNorm((5000,), eps=1e-05, elementwise_affine=True)
    (2): Linear(in_features=5000, out_features=1024, bias=True)
    (3): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (transformer): Transformer(
    (layers): ModuleList(
      (0-11): 12 x ModuleList(
        (0): Attention(
          (norm): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
          (to_qkv): Linear(in_features=1024, out_features=3072, bias=False)
          (dropout): Dropout(p=0.1, inplace=False)
          (reattn_norm): Sequential(
            (0): Rearrange('b h i j -> b i j h')
            (1): LayerNorm((16,), eps=1e-05, elementwise_affine=True)
            (2): Rearrange('b i j h -> b h i j')
          )
          (to_out): Sequential(
            (0): Linear(in_features=1024, out_features=1024, bias=True)
            (1): Dropout(p=0.1, inplace=False)
          )
        )
        (1): FeedForward(
          (net): Sequential(
            (0): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
            (1): Linear(in_features=1024, out_features=2048, bias=True)
            (2): GELU(approximate='none')
            (3): Dropout(p=0.1, inplace=False)
            (4): Linear(in_features=2048, out_features=1024, bias=True)
            (5): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
  )
  (to_latent): Identity()
  (mlp_head): Sequential(
    (0): LayerNorm((1024,), eps=1e-05, elementwise_affine=True)
    (1): Linear(in_features=1024, out_features=1, bias=True)
  )
)
```

Setting the loss function and the optimizer

In [18]:

```
#Using the Adamw loss function as it is widely used for transformer architectures
loss_fn = torch.nn.MSELoss()
optimizer = torch.optim.AdamW(model.parameters(), lr=0.001, weight_decay=0.0001)
```

Training and testing the model

In [22]:

```
from tqdm.auto import tqdm
epochs = 30
for epoch in tqdm(range(epochs)):
```

```python
    model.train()
    for images, weights in train_dataloader:
        images, weights = images.to(device), weights.to(device)

        optimizer.zero_grad()
        outputs = model(images)
        loss = loss_fn(outputs.squeeze(), weights.squeeze())

        loss.backward()
        optimizer.step()

    model.eval()
    with torch.inference_mode():
        for images, weights in test_dataloader:
            images, weights = images.to(device), weights.to(device)
            outputs = model(images)
            test_loss = loss_fn(outputs.squeeze(), weights.squeeze())

    print(f'Epoch: {epoch} || Train Loss: {loss:.3f} || Test Loss: {test_loss:.3f}')
```

```
Epoch: 0 || Train Loss: 0.397 || Test Loss: 0.046
Epoch: 1 || Train Loss: 1.684 || Test Loss: 0.093
Epoch: 2 || Train Loss: 1.415 || Test Loss: 0.030
Epoch: 3 || Train Loss: 0.805 || Test Loss: 0.253
Epoch: 4 || Train Loss: 1.089 || Test Loss: 0.063
Epoch: 5 || Train Loss: 0.584 || Test Loss: 0.328
Epoch: 6 || Train Loss: 1.038 || Test Loss: 0.044
Epoch: 7 || Train Loss: 1.415 || Test Loss: 0.132
Epoch: 8 || Train Loss: 0.842 || Test Loss: 0.318
Epoch: 9 || Train Loss: 1.017 || Test Loss: 0.292
Epoch: 10 || Train Loss: 0.744 || Test Loss: 0.098
Epoch: 11 || Train Loss: 0.949 || Test Loss: 0.212
Epoch: 12 || Train Loss: 0.837 || Test Loss: 0.090
Epoch: 13 || Train Loss: 0.811 || Test Loss: 0.728
Epoch: 14 || Train Loss: 0.754 || Test Loss: 0.111
Epoch: 15 || Train Loss: 1.634 || Test Loss: 0.208
Epoch: 16 || Train Loss: 0.924 || Test Loss: 0.050
Epoch: 17 || Train Loss: 0.499 || Test Loss: 0.456
Epoch: 18 || Train Loss: 0.586 || Test Loss: 0.080
Epoch: 19 || Train Loss: 0.570 || Test Loss: 0.103
Epoch: 20 || Train Loss: 1.095 || Test Loss: 0.069
Epoch: 21 || Train Loss: 0.735 || Test Loss: 0.131
Epoch: 22 || Train Loss: 0.476 || Test Loss: 0.041
Epoch: 23 || Train Loss: 0.929 || Test Loss: 0.259
Epoch: 24 || Train Loss: 1.467 || Test Loss: 0.026
Epoch: 25 || Train Loss: 1.718 || Test Loss: 0.057
Epoch: 26 || Train Loss: 1.068 || Test Loss: 0.283
Epoch: 27 || Train Loss: 1.260 || Test Loss: 0.020
Epoch: 28 || Train Loss: 0.784 || Test Loss: 0.001
Epoch: 29 || Train Loss: 1.608 || Test Loss: 0.001
```

Therefore we get a MSE Loss of 0.001 on the test dataset

In [23]:

```python
checkpoint = {
    'state_dict' : model.state_dict(),
    'optimizer' : optimizer.state_dict()
}
torch.save(checkpoint, 'checkpoint.pth')
```