# Video Embeddings Regression Model

Given a folder containing folders of sequence of indexed images (because I want to run regression analysis on videos, so a video will be split into frames).
Make sure that the number of images for each sequence doesn't exceed 196.
Then, using SigLip2 I want to create a tensor of embeddings from the images (each converted into an embedding). The resulting embeddings tensor is of size [196, 768] (padding the short sequences, of course).

Keep in mind that the resulting output embeddings tensor will be fed into the following model architecture as the input tensor, and the model training will be administered using tensorflow:

```
import tensorflow as tf
from transformers import TFViTModel

# 1. Define input for a sequence of 196 frame embeddings (each 768-D)
frame_seq_input = tf.keras.Input(shape=(196, 768), dtype=tf.float32, name="frame_sequence")

# 2. Linear projection layer to match ViT Tiny's embedding size (192)
proj_layer = tf.keras.layers.Dense(192, name="proj_768_to_192")
projected_seq = proj_layer(frame_seq_input)  # shape: (batch_size, 196, 192)

# 3. Load pre-trained ViT Tiny model (no classification head)
vit_base = TFViTModel.from_pretrained('facebook/deit-tiny-patch16-224')

# 4. Pass projected embeddings to ViT model using the `inputs_embeds` argument
outputs = vit_base(inputs_embeds=projected_seq)
# `outputs.last_hidden_state` has shape (batch_size, 197, 192) -> includes CLS token at index 0
```

```
cls_output = outputs.last_hidden_state[:, 0, :]   # shape: (batch_size, 192)

# 5. Regression head: predict 2 continuous values from CLS representation
regression_output = tf.keras.layers.Dense(2, name="regression_head")(cls_output)

# 6. Build and compile the model
regression_model = tf.keras.Model(inputs=frame_seq_input, outputs=regression_output)
regression_model.compile(optimizer=tf.keras.optimizers.Adam(1e-4), loss='mse')
regression_model.summary()
```

Notes:

The input tensor frame_seq_input expects a sequence of 196 embeddings, each of dimension 768.

The projection layer reduces each 768-dimensional embedding to 192 dimensions to match the expected input size of the DeiT-Tiny model.

The TFViTModel from Hugging Face's Transformers library is used to load the pre-trained DeiT-Tiny model without its classification head.

By providing the inputs_embeds argument, we feed our custom embeddings directly into the model, bypassing the internal patch embedding layer.

The model automatically adds a [CLS] token, resulting in an output sequence of length 197. We extract the embedding corresponding to the [CLS] token for regression.

A final dense layer maps the [CLS] embedding to two continuous output values, suitable for regression tasks.

To process your video frames into embeddings suitable for your TensorFlow regression model, you can utilize the SigLIP2 model from Hugging Face. Here's a step-by-step guide to achieve this:

---

## 🔧 Step 1: Install Required Libraries

Ensure you have the necessary libraries installed:

```bash
pip install transformers torch torchvision pillow
```

---

## 📁 Step 2: Organize Your Data

Assuming your data is structured as follows:

```
video_frames/
├── video1/
│   ├── frame_0001.jpg
│   ├── frame_0002.jpg
│   └── ...
├── video2/
│   ├── frame_0001.jpg
│   ├── frame_0002.jpg
│   └── ...
└── ...
```

Each subfolder (`video1`, `video2`, etc.) contains frames from a single video.

---

## 🖼️ Step 3: Load SigLIP2 Model and Processor

python

```python
from transformers import AutoProcessor, AutoModel
import torch

# Load the SigLIP2 model and processor
model_name = "google/siglip2-base-patch16-224"
processor = AutoProcessor.from_pretrained(model_name)
model = AutoModel.from_pretrained(model_name)
model.eval()  # Set model to evaluation mode
```

---

## 🧠 Step 4: Define a Function to Process Video Frames

python

```python
import os
from PIL import Image
import numpy as np

def process_video_frames(video_folder, max_frames=196, embedding_dim=768):
    # List and sort frame files
    frame_files = sorted([
        os.path.join(video_folder, f)
        for f in os.listdir(video_folder)
        if f.lower().endswith(('.png', '.jpg', '.jpeg'))
    ])

    # Limit to max_frames
    frame_files = frame_files[:max_frames]

    embeddings = []
    for frame_file in frame_files:
        image = Image.open(frame_file).convert("RGB")
        inputs = processor(images=image, return_tensors="pt")
        with torch.no_grad():
            image_features = model.get_image_features(**inputs)
        embeddings.append(image_features.squeeze(0).numpy())

    # Pad if necessary
```

```
    while len(embeddings) < max_frames:
        embeddings.append(np.zeros(embedding_dim))

    return np.stack(embeddings)
```

---

## 📦 Step 5: Process All Videos and Prepare Dataset

python

```
import glob

video_dirs = glob.glob("video_frames/*")
dataset = []

for video_dir in video_dirs:
    embedding_tensor = process_video_frames(video_dir)
    dataset.append(embedding_tensor)

# Convert to NumPy array
dataset = np.stack(dataset)  # Shape: (num_videos, 196, 768)
```

---

## 🔄 Step 6: Convert to TensorFlow Dataset

python

```
import tensorflow as tf

# Convert NumPy array to TensorFlow tensor
tf_dataset = tf.convert_to_tensor(dataset, dtype=tf.float32)
```

Now, `tf_dataset` is a TensorFlow tensor of shape `(num_videos, 196, 768)` and can be fed into your regression model as defined.

---

## 🧪 Optional: Visualize Embeddings

If you're interested in visualizing the high-dimensional embeddings, consider using TensorFlow's Embedding Projector. You'll need to:

1. Save the embeddings and associated metadata.
2. Upload them to a public URL or GitHub Gist.
3. Load them into the Embedding Projector for visualization.

---

This pipeline ensures that each video is represented as a sequence of 196 embeddings, each of dimension 768, suitable for input into your TensorFlow regression model.