Reading: Assignment Overview: Vision Transformers in Keras

Estimated time: 3 minutes

Introduction

In this lab, you will integrate Convolutional Neural Networks (CNNs) with Vision Transformers (ViTs), leveraging the spatial feature extraction strengths of CNNs alongside the global context modeling capabilities of ViTs.

Let's go through the overview of steps and tasks that you will perform during this lab:

Steps and tasks

1. In your first exercise, before integrating with ViT, you will load the pre-trained CNN model to extract and feed local features from the input image to the Transformer layers. If you want to use the model you previously saved from other labs, you can upload it to the left column of your JupyterLab environment. Otherwise, you can download the pre-trained model, trained on the same data, from the links provided in this lab.

```
from tensorflow.keras.models import load_model
cnn_model = load_model('path_to_pretrained_cnn_model')
cnn_model.summary()
```

2. In your next exercise, you will select the convolutional layer from the CNN architecture for feature extraction.
3. The features extracted by the CNN convolutional layer are converted to patches

```
# Feature extraction
features = cnn_model.get_layer(feature_layer_name).output

# Patch Creation by flattening the spatial grid
H, W, C = features.shape[1], features.shape[2], features.shape[3]
x = layers.Reshape((H * W, C))(features)
```

4. Next, positional embedding is added to the patches

```
# Positional embedding that Keras can track
@tf.keras.utils.register_keras_serializable(package="Custom")
class AddPositionEmbedding(layers.Layer):
    def __init__(self, num_patches, embed_dim, **kwargs):
        super().__init__(**kwargs)
        self.num_patches = num_patches
        self.embed_dim   = embed_dim
        self.pos = self.add_weight(
            name="pos_embedding",
            shape=(1, num_patches, embed_dim),
            initializer="random_normal",
            trainable=True)

    def call(self, tokens):
        return tokens + self.pos

    x = AddPositionEmbedding(H * W, C)(x)
```

5. Then, you will feed this into the transformer layers via TransformerBlock class

```
class TransformerBlock(layers.Layer):
    def __init__(self, embed_dim, num_heads=8, mlp_dim=2048, dropout=
        super().__init__(**kwargs)
        self.embed_dim = embed_dim
        self.num_heads = num_heads
        self.mlp_dim   = mlp_dim
        self.dropout   = dropout
        self.mha  = layers.MultiHeadAttention(num_heads, key_dim=embe
        self.norm1 = layers.LayerNormalization(epsilon=1e-6)
        self.norm2 = layers.LayerNormalization(epsilon=1e-6)
        self.mlp = tf.keras.Sequential([
```

6. In your next exercise, you will define the hybrid model architecture. You should always tune the number of transformer heads, depth, and CNN layers for your specific image domain and dataset size.
7. In your final exercise, you will define the hybrid model's training configuration with hyperparameters such as the number of epochs and batch size
8. You will finish this lab by plotting the accuracy and loss values of the model training.
9. This approach gives you a modular, hybrid architecture that leverages the best of both CNNs and Vision Transformers.
10. After completing this lab, you will need to download and save it on your computer for final submission and evaluation at the end of this course. Good luck, and let's get started.

Note: All code displayed in the screenshots is provided within the Jupyter notebooks that you will use to complete the lab exercises.

Skills Network