Reading: Assignment Overview: Vision Transformers and Transfer Learning

Estimated time needed: 5 minutes

Learning objectives:

- Explain the architecture and workflow of Vision Transformers (ViTs), including patch embedding and positional encoding.
- Compare and contrast the implementation of hybrid CNN-ViT models using Keras and PyTorch frameworks.

Vision Transformers and Transfer learning

Vision transformers: Core concepts

This reading introduces Vision transformers (ViTs) as an alternative to CNNs for image classification tasks. It explains the ViT architecture, including patchification, positional encoding, and transformer encoders. The reading also covers hybrid CNN+ViT models and practical implementation strategies in both Keras and PyTorch for scalable and flexible deep learning applications.

From CNNs to transformers

We know that CNNs are very efficient at extracting local features through convolutional filters and spatial hierarchies. However, they are limited in capturing spatial dependencies and global context without deep stacks or large kernels.

Vision transformers address this limitation by treating images as sequences of patches, similar to words in a sentence. Each patch is embedded into a vector, positional encodings are added, and the resulting sequence is processed by transformer encoder blocks using self-attention. This enables ViTs to model both local and global relationships, regardless of spatial distance in the image.

ViT architecture

- **Patchification:** The image is split into fixed-size patches (e.g., 16×16 pixels). Each patch is flattened and projected into a higher-dimensional space.
- **Positional encoding:** Since transformers lack spatial awareness, positional encodings are added to each patch embedding to retain spatial information.
- **Transformer encoder:** Stacked self-attention and feed-forward layers process the sequence, capturing dependencies between all patches.
- **Classification head:** A special [CLS] token or pooled output is used for final prediction.

CNN + ViT

While pure ViTs are powerful, they tend to require large datasets and computational resources due to their minimal inductive bias. To address this, hybrid models combine a CNN front-end for local feature extraction with a ViT back-end for global context modeling. The CNN extracts spatial features and reduces input dimensionality, making the subsequent transformer more efficient and robust, especially on smaller datasets.

Workflow:

1. CNN processes the image, producing a feature map.
2. The feature map is divided into patches (tokens).
3. Each patch is embedded and passed, with positional encoding, into the transformer.
4. The transformer outputs are pooled and fed to a classifier.

Implementation in Keras

Keras, built on TensorFlow, offers high-level APIs and ready-to-use layers for both CNNs and transformers. This makes it ideal for rapid prototyping and deployment.

For integration with ViT, a pre-trained CNN model, such as ResNet50, can be imported as:

```
cnn_base = keras.applications.ResNet50(include_top=False, weights='imagenet', pooling=None)
```

Next, Patch extraction, to flatten spatial dimensions, is performed:

```
patches = layers.Reshape((-1, x.shape[-1]))(x) # (batch, num_patches, channels)
```

Patch embedding is done as:

```
embed_dim = 128
patch_embeddings = layers.Dense(embed_dim)(patches)
```

Then, we setup the positional encoding as:

```
positions = tf.range(start=0, limit=num_patches, delta=1)
pos_embed = layers.Embedding(input_dim=num_patches, output_dim=embed_dim)(positions)
encoded_patches = patch_embeddings + pos_embed
```

The Transformer encoder blocks are declared as:

```
transformer = layers.TransformerBlock
(
  num_heads=4, key_dim=embed_dim, ff_dim=256, dropout=0.1
)
x = transformer(encoded_patches)
x = layers.GlobalAveragePooling1D()(x)
outputs = layers.Dense(1, activation='sigmoid')(x)

model = keras.Model(inputs, outputs)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Keras advantages

- **Simplicity:** High-level layers and functional API make chaining CNNs and transformers straightforward.
- **Pretrained models:** Access to pretrained ViTs and CNNs via KerasCV and TensorFlow Hub. Smaller models can be created rapidly.
- **Deployment:** Seamless integration with TensorFlow Serving and TFLite for production.

Implementation in PyTorch

PyTorch is favored for its flexibility and transparency, especially in exploratory settings. Libraries like torchvision and timm offer pretrained ViT models and utilities for patch extraction and transformer blocks.

PyTorch advantages

- **Flexibility:** Full control over architecture, data flow, and training loop.
- **Research-oriented:** Dynamic computation graph and easy debugging.
- **Ecosystem:** Access to timm, Hugging Face Transformers, and custom ViT implementations.

Vision transformer principles

- **Global context:** Self-attention enables ViTs to model long-range dependencies, capturing global relationships between image regions.
- **Minimal inductive bias:** Unlike CNNs, ViTs do not assume locality or translation invariance, making them more flexible but also more data-hungry.
- **Scalability:** ViTs scale well with model and dataset size, often outperforming CNNs on large benchmarks.
- **Hybrid strength:** Combining CNNs and ViTs leverages the local feature extraction of CNNs and the global context modeling of ViTs, improving performance on diverse datasets and tasks.

Pros and cons of each framework

Both frameworks, Keras and PyTorch, have utilities that help in the seamless integration of a pre-trained CNN into a ViT. However, their implementation is different. The table give below can help a user get a broad understanding of each framework, thus helping them decide which one to use:

Aspect	**Keras (TensorFlow)**	**PyTorch**
Ease of Use	High-level, intuitive API, fast prototyping	More verbose, but highly flexible

Pretrained Models	Good support via KerasCV, TensorFlow Hub	Extensive via timm, torchvision, Hugging Face
Customization	Less flexible for novel research	Full control for custom architectures
Training Loop	Abstracted (fit, callbacks)	Manual, highly customizable
Deployment	Seamless with TensorFlow Serving/TFLite	More manual, but improving
Community	Strong in industry and production	Favored in research and academia

Summary

In this reading, you learned that:

1. Vision Transformers treat images as sequences, enabling modeling of both local and global features using self-attention.
2. Patchification and positional encoding are critical steps in ViT architectures to process images spatially.
3. Hybrid CNN+ViT models combine local feature extraction with global attention, improving performance on small datasets.
4. Keras offers fast prototyping and ease-of-use with pre-trained models and API integration, ideal for production workflows.
5. PyTorch provides greater flexibility and transparency, making it well-suited for research and custom architectures.