cs512 f24 - assignment 5 (vision transformer)

Due by: 11/20/2024

General Instructions:

In this assignment, you will implement and compare two Vision Transformer-based models for image classification on the CIFAR-10 dataset or the Cats and Dogs dataset (one dataset):

1. A standard Vision Transformer (ViT) model for classification.
2. A hybrid model using pretrained VGG16 layers to extract low-level features before passing them through ViT blocks.

You can use Keras and are encouraged to run the models on a GPU (e.g., Google Colab) for faster training. Since in both datasets images are smaller than what is expected by the ViT model, you will need to resize them to 224x224 to be compatible with the Vision Transformer's input requirements. In addition, you need to take a subset of 4000 images to control the training time. We recommend using the Cats and Dogs dataset where you can get better results.

Ensure that your program saves and loads weights so training can resume from a previous state. Additionally, you will log and plot metrics as a function of training and evaluation iterations.

Submission instructions:

- Follow the submission instructions of assignment 0.
- Submit a fully executed Python notebook with no gaps in execution. To receive full credit, please ensure the following:
    - The notebook includes all cell outputs and contains no error messages.
    - It is easy to match each task with its corresponding code solution using clear markdown or comments (e.g., "Task 2").
    - Re-run your notebook before submitting to ensure that cells are numbered sequentially, starting at [1].

Tasks:

1. Dataset Preparation

1. **Load dataset**: CIFAR-10 contains 10 classes (e.g., airplane, car, bird, cat, deer). The Cats and Dogs contains two classes (cats, dogs).
    - Load the dataset using Keras.
    - Resize all images to 224x224 pixels, and normalize pixel values to [0, 1].
    - Split the data into **training** and **testing** sets. Further split the testing set into a testing and validation subsets. Set up data loading for each.

2. Standard Vision Transformer (ViT) Model

1. **Implement the ViT Model**:
    - Load a pretrained Vision Transformer (ViT) model from Hugging Face using the `TFViTModel` class.
    - Add a classification head after the ViT blocks, with 10 output units (one for each class in CIFAR-10), or 1 output unit for the Cats and Dogs problem).
    - Compile the model with an appropriate optimizer, loss function, and metrics.
2. **Training**:
    - Train the model on the training set.
    - Save checkpoints periodically, and log training/validation loss and accuracy.
3. **Evaluation**:
    - Evaluate the model on the test set, reporting accuracy and other relevant metrics.
    - Plot the training and validation accuracy and loss over epochs.

3. Hybrid VGG16 + ViT Model

1. **Pretrained VGG16 for Feature Extraction**:
    - Use pretrained VGG16 layers up to `block3_conv3` or up to `block5_conv3` to extract initial image features. The advantage of `block5_conv3` is that it produces a smaller output (14x14x512) for ViT.
    - Freeze these VGG16 layers to use them as a fixed feature extractor.
2. **Implement the Hybrid Model**:
    - After extracting features with VGG16, feed the output into the ViT blocks.
    - Ensure the output of VGG16 layers matches the ViT input shape requirements.
    - Add a classification head after the ViT blocks to produce predictions.
3. **Training and Evaluation**:
    - Train the hybrid model on the CIFAR-10 or Cats and Dogs training set.
    - Evaluate the hybrid model on the test set, tracking the same metrics as in Task 2.
    - Save checkpoints and visualize some predictions on test images.

4. Analysis and Comparison

1. **Performance Analysis**:
    - Compare the performance of the standard ViT and hybrid VGG16 + ViT models.
    - Report quantitative metrics such as accuracy, loss, and training times.
    - Visualize any training or validation differences with relevant plots.
2. **Discussion**:
    - Reflect on the performance of each model, describing which model performed better and hypothesize why.
    - Discuss trade-offs like computational complexity, training time, and memory usage.
    - Suggest possible ways to further improve model performance, like data augmentation or fine-tuning additional layers in VGG16.

Starter Code for ViT network

```
```

```
# Step 1: Install necessary libraries
!pip install transformers tensorflow

# Step 2: Import libraries
from transformers import ViTFeatureExtractor, TFViTModel
from tensorflow.keras import layers, Model, optimizers
from tensorflow.keras.datasets import cifar10
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt

# Step 3: Load and preprocess CIFAR-10 dataset
(img_train, label_train), (img_test, label_test) = cifar10.load_data()

# Limit to the first 100 images and labels in the training set
img_train = img_train[:100]
label_train = label_train[:100]

# Similarly, limit the test set
img_test = img_test[:100]
label_test = label_test[:100]

IMG_SIZE = 224

def normalize_resize(img, label):
    img = tf.cast(img, tf.float32)
    img = tf.divide(img, 255)
    img = tf.image.resize(img, (IMG_SIZE, IMG_SIZE))
    return img, label


def build_data_pipeline(img, label, batch_size, is_train=True):
  # Create the dataset from slices of the images and labels
  data_ds = tf.data.Dataset.from_tensor_slices((img, label))

  # Shuffle the data for training
  if is_train:
    data_ds = data_ds.shuffle(batch_size*2)

  # Resize and normalization
  data_ds = data_ds.map(normalize_resize)

  # Batch the images
  data_ds = data_ds.batch(batch_size)

  # Prefetch
  data_ds = data_ds.prefetch(tf.data.AUTOTUNE)
  return data_ds


BATCH_SIZE = 10 #32
train_ds = build_data_pipeline(img_train, label_train, batch_size=BATCH_SIZE)
val_ds =  build_data_pipeline(img_test, label_test, batch_size=BATCH_SIZE, is_train=False)
```

```python
# Step 4: Define the Standard Vision Transformer (ViT) Model

# Custom wrapper layer for the ViT model
class ViTModelLayer(layers.Layer):
    def __init__(self, vit_model, **kwargs):
        super(ViTModelLayer, self).__init__(**kwargs)
        self.vit_model = vit_model

    def call(self, inputs):
        # Adjust dimensions to match the expected input shape for ViT: (batch_size, 3,
height, width)
        inputs = tf.transpose(inputs, [0, 3, 1, 2])
        # Pass the adjusted input through the ViT model
        vit_ouput = self.vit_model(pixel_values=inputs).last_hidden_state[:, 0]
        return vit_ouput

# Define the model creation function
def create_vit_model(num_classes):
    # Load the pretrained Vision Transformer model from Hugging Face
    vit_model = TFViTModel.from_pretrained('google/vit-base-patch16-224-in21k')

    # Define input layer
    input_layer = tf.keras.Input(shape=(IMG_SIZE, IMG_SIZE, 3))

    # Wrap ViT model in the custom layer
    vit_output = ViTModelLayer(vit_model)(input_layer)

    # Add a classification head
    output_layer = layers.Dense(num_classes, activation='softmax')(vit_output)

    # Create the final model
    model = Model(inputs=input_layer, outputs=output_layer)
    return model

# Step 5: Instantiate the ViT model and compile it
num_classes = 10  # CIFAR-10 has 10 classes
vit_model = create_vit_model(num_classes)
vit_model.compile(optimizer=optimizers.Adam(learning_rate=1e-4),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

# Step 6: Train the Standard ViT Model
EPOCHS = 10
vit_history = vit_model.fit(train_ds, validation_data=val_ds, epochs=EPOCHS)

# Step 7: Evaluate the Standard ViT Model
val_loss, val_accuracy = vit_model.evaluate(val_ds)
print(f"Validation Loss: {val_loss}, Validation Accuracy: {val_accuracy}")

# Step 8: Plot training and validation curves
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(vit_history.history['accuracy'], label='Training Accuracy')
plt.plot(vit_history.history['val_accuracy'], label='Validation Accuracy')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(vit_history.history['loss'], label='Training Loss')
plt.plot(vit_history.history['val_loss'], label='Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

Starter Code for VGG+ViT network

```
# Define the model creation function
def create_vgg_vit_model(num_classes):
    # ViTConfig() is the configuration class to store the configuration of a ViTModel.
    # It is used to instantiate an ViT model according to the specified arguments and model
architecture.
    # Link:
https://huggingface.co/docs/transformers/main/en/model_doc/vit#transformers.ViTConfig
    from transformers import ViTConfig
    vit_config = ViTConfig()

    # To make it work with hybrid model, you need to change the below three arguments:
    #   image_size: The size (resolution) of each image
    #   num_channels: The number of input channels
    #   patch_size: The size (resolution) of each patch
    # The image_size and num_channels should match the output of VGG16
    vit_config.image_size =
    vit_config.num_channels =

    # Set the patch size to 1, as specified in the original paper (page 5),
    # to handle the intermediate feature maps input
    # Paper link: https://arxiv.org/pdf/2010.11929
    vit_config.patch_size = 1

    # When loading the pretrained model, the configuration Class should be passed via
'config' argument.
    # Also, the 'ignore_mismatched_sizes' should be set to True.
    vit_model = TFViTModel.from_pretrained(
        'google/vit-base-patch16-224-in21k',
        config=vit_config,
        ignore_mismatched_sizes=True,
    )

    # Assemble your hybrid model: VGG16 + ViT
    # ...

    # Create the final mdoel
    model = Model(inputs=input_layer, outputs=output_layer)
    return model
```