CENG442 FINAL SPRING 2024

DUE DATE AND TIME: 31/05/2024 23:59

The TRACE

Start: 25st MAY 2024 - 23:30

Project Overview

You will be given a nearly complete Jupyter notebook code as an image to analyze. This code utilizes the COCO dataset and employs the VGG16 model for image classification. Your task is to trace the code, identify the model built within it, and understand its functionality. You are required to produce a detailed written report explaining **the code, the model, and its components**, as well as create a video presentation of your findings. The libraries are listed in the appendix at the end.

Project Requirements

In your written report, include an introduction, a detailed analysis of the code (covering preprocessing, feature extraction, model building with VGG16, training on the COCO dataset, and evaluation), and a conclusion summarizing your insights. The COCO dataset is a large-scale object detection, segmentation, and captioning dataset, and VGG16 is a convolutional neural network model known for its depth and simplicity. Your video presentation (10-15 minutes) should explain the code sections, provide insights, and be uploaded to YouTube as unlisted. You are responsible for ensuring the video's accessibility. Both the report and the video link must be submitted by the deadline.

Grading Criteria

Grading will be based on the clarity and thoroughness of your written report and video presentation. The written report accounts for 50% of the grade, focusing on the accuracy of the code analysis and quality of explanations. The video presentation also accounts for 50%, evaluating the depth of explanation, presentation skills, and video accessibility. Your work will be graded comparatively based on the quality and depth of your analysis relative to other groups' work.

COCO Dataset

The COCO (Common Objects in Context) dataset is a large-scale dataset designed for object detection, segmentation, and image captioning tasks. It contains over 200,000 labeled images, featuring more than 1.5 million object instances across 80 object categories. COCO is unique due to its focus on contextual information, capturing objects in complex scenes with multiple objects interacting. This makes it a challenging and comprehensive dataset for training and evaluating machine learning models, especially in tasks requiring detailed understanding of image content and context.

VGG16 Model

VGG16 is a convolutional neural network (CNN) model developed by the Visual Geometry Group (VGG) at the University of Oxford. It is renowned for its simple and effective architecture, consisting of 16 layers including convolutional and fully connected layers. VGG16 uses small 3x3 convolution filters, which help in capturing fine details in images while maintaining computational efficiency. The model is pretrained on the ImageNet dataset, making it highly effective for various image classification tasks. Its architecture and depth enable it to learn rich feature representations, making it a popular choice for transfer learning in computer vision applications.

Trace:

```
# Downloading and extracting the COCO dataset
coco.download_and_extract_if_needed()

# Loading training records
_, train_images, train_captions = coco.get_dataset(train=True)
total_train_images = len(train_images)

# Loading validation records
_, val_images, val_captions = coco.get_dataset(train=False)
total_val_images = len(val_images)

def fetch_image(file_path, dimensions=None):
    image = Image.open(file_path)

    if dimensions is not None:
        image = image.resize(size=dimensions, resample=Image.LANCZOS)

    image_array = np.array(image)

    image_array = image_array / 255.0

    if image_array.ndim == 2:
        image_array = np.stack((image_array,) * 3, axis=-1)

    return image_array

def display_image(index, is_training):
    if is_training:
        directory = coco.train_directory
        file_name = train_images[index]
        image_captions = train_captions[index]
    else:
        directory = coco.val_directory
        file_name = val_images[index]
        image_captions = val_captions[index]

    image_path = os.path.join(directory, file_name)

    for caption in image_captions:
        print(caption)

    image = fetch_image(image_path)
    plt.imshow(image)
    plt.show()

display_image(index=1, is_training=True)
```

```
#A giraffe feeding from the top branches of a tree.
#A giraffe standing near a tree.
#A mother giraffe with her calf in the forest.
#Two giraffes in a wooded area.
#A giraffe standing next to a tree-filled forest.
```

```
pretrained_model = VGG16(include_top=True, weights='imagenet')
pretrained_model.summary()
```

```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 224, 224, 3)]     0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
_________________________________________________________________
```

```
feature_layer = pretrained_model.get_layer('fc2')
model_transfer = Model(inputs=pretrained_model.input,
                       outputs=feature_layer.output)
image_dimensions = K.int_shape(pretrained_model.input)[1:3]
feature_vector_size = K.int_shape(feature_layer.output)[1]
feature_vector_size
image_dimensions
# (224, 224)

def display_progress(current_count, total_count):
    percent_complete = current_count / total_count
    status_message = "\r- prg status: {0:.1%}".format(percent_complete)
    sys.stdout.write(status_message)
    sys.stdout.flush()

def process_image_batch(data_directory, file_names, batch_size=32):
    total_images = len(file_names)

    input_shape = (batch_size,) + image_dimensions + (3,)
    image_batch_array = np.zeros(shape=input_shape, dtype=np.float16)

    output_shape = (total_images, feature_vector_size)
    feature_vectors = np.zeros(shape=output_shape, dtype=np.float16)

    current_index = 0

    while current_index < total_images:
        display_progress(current_count=current_index, total_count=total_images)

        end_index = current_index + batch_size

        if end_index > total_images:
            end_index = total_images

        current_batch_size = end_index - current_index

        for i, file_name in enumerate(file_names[current_index:end_index]):

            image_path = os.path.join(data_directory, file_name)

            image = fetch_image(image_path, dimensions=image_dimensions)

            image_batch_array[i] = image

        feature_batch = model_transfer.predict(image_batch_array[:current_batch_size])

        feature_vectors[current_index:end_index] = feature_batch[:current_batch_size]

        current_index = end_index

    print()

    return feature_vectors

def process_training_images():
    print("The images are processed {0}".format(len(train_images)))
    cache_file_path = os.path.join(coco.data_directory, "feature_vectors_train.pkl")
    feature_vectors = cache(cache_path=cache_file_path,
                            fn=process_image_batch,
                            data_directory=coco.train_directory,
                            file_names=train_images)
    return feature_vectors
```

```
def process_validation_images():
    print("The images are processed {0}".format(len(val_images)))
    cache_file_path = os.path.join(coco.data_directory, "feature_vectors_val.pkl")
    feature_vectors = cache(cache_path=cache_file_path,
                            fn=process_image_batch,
                            data_directory=coco.val_directory,
                            file_names=val_images)
    return feature_vectors

feature_vectors_train = process_training_images()
print("dtype:", feature_vectors_train.dtype)
print("shape:", feature_vectors_train.shape)

feature_vectors_val = process_validation_images()
print("dtype:", feature_vectors_val.dtype)
print("shape:", feature_vectors_val.shape)

start_marker = 'xxxx '
end_marker = ' wwww'
def add_markers_to_captions(captions_nested_list):
    marked_captions = [[start_marker + caption + end_marker
                        for caption in captions]
                        for captions in captions_nested_list]
    return marked_captions

marked_train_captions = add_markers_to_captions(train_captions)
marked_train_captions[0]
train_captions[0]

def flatten_captions(nested_captions_list):
    flat_captions_list = [caption
                          for captions in nested_captions_list
                          for caption in captions]
    return flat_captions_list

flat_train_captions = flatten_captions(marked_train_captions)
vocab_size = 10000
```

```
class ExtendedTokenizer(Tokenizer):
    def __init__(self, text_data, max_words=None):
        super().__init__(num_words=max_words)
        self.fit_on_texts(text_data)
        self.token_to_word_map = {index: word for word, index in self.word_index.items()}

    def token_to_word(self, token):
        return " " if token == 0 else self.token_to_word_map.get(token, "")

    def tokens_to_string(self, tokens):
        words = [self.token_to_word_map[token] for token in tokens if token != 0]
        return " ".join(words)

    def captions_to_tokens(self, nested_captions_list):
        token_sequences = [self.texts_to_sequences(captions) for captions in nested_captions_list]
        return token_sequences

tokenizer = ExtendedTokenizer(text_data=flat_train_captions,
                              max_words=vocab_size)

start_token = tokenizer.word_index[start_marker.strip()]
end_token = tokenizer.word_index[end_marker.strip()]
train_tokens = tokenizer.captions_to_tokens(marked_train_captions)

def select_random_caption_tokens(indices):
    selected_tokens = []
    for index in indices:
        random_index = np.random.choice(len(train_tokens[index]))
        random_tokens = train_tokens[index][random_index]
        selected_tokens.append(random_tokens)
    return selected_tokens

def generate_batches(batch_size):
    while True:
        indices = np.random.randint(total_train_images, size=batch_size)
        feature_vectors = feature_vectors_train[indices]
        caption_tokens = select_random_caption_tokens(indices)
        num_tokens = [len(tokens) for tokens in caption_tokens]
        max_num_tokens = np.max(num_tokens)
        padded_tokens = pad_sequences(caption_tokens,
                                      maxlen=max_num_tokens,
                                      padding='post',
                                      truncating='post')
        decoder_input_data = padded_tokens[:, 0:-1]
        decoder_output_data = padded_tokens[:, 1:]
        x_data = {
            'decoder_input': decoder_input_data,
            'feature_vectors_input': feature_vectors
        }
        y_data = {
            'decoder_output': decoder_output_data
        }
        yield (x_data, y_data)
```

```
batch_gen = generate_batches(batch_size=batch_size)
batch = next(batch_gen)
batch_inputs = batch[0]
batch_outputs = batch[1]

num_train_captions = [len(captions) for captions in train_captions]
total_train_captions = np.sum(num_train_captions)
steps_per_epoch = int(total_train_captions / batch_size)

feature_vectors_input = Input(shape=(feature_vector_size,),
                              name='feature_vectors_input')
decoder_feature_mapping = Dense(units=state_size,
                                activation='tanh',
                                name='decoder_feature_mapping')
decoder_input_sequence = Input(shape=(None,), name='decoder_input_sequence')
decoder_embedding_layer = Embedding(input_dim=vocab_size,
                                    output_dim=embedding_size,
                                    name='decoder_embedding_layer')
decoder_gru_layer1 = GRU(units=state_size, name='decoder_gru_layer1', return_sequences=True)
decoder_gru_layer2 = GRU(units=state_size, name='decoder_gru_layer2', return_sequences=True)
decoder_gru_layer3 = GRU(units=state_size, name='decoder_gru_layer3', return_sequences=True)
decoder_output_layer = Dense(units=vocab_size,
                             activation='softmax',
                             name='decoder_output_layer')
def build_decoder(feature_vectors):
    initial_state = decoder_feature_mapping(feature_vectors)
    net = decoder_input_sequence
    net = decoder_embedding_layer(net)
    net = decoder_gru_layer1(net, initial_state=initial_state)
    net = decoder_gru_layer2(net, initial_state=initial_state)
    net = decoder_gru_layer3(net, initial_state=initial_state)
    decoder_output = decoder_output_layer(net)
    return decoder_output
decoder_output = build_decoder(feature_vectors=feature_vectors_input)
decoder_model = Model(inputs=[feature_vectors_input, decoder_input_sequence],
                      outputs=[decoder_output])
#####################################################END#########################################################
```

The Grading Rubric

Criteria	Excellent (90-100%)	Good (75-89%)	Satisfactory (60-74%)	Needs Improvement (0-59%)
Written Report (50%)	Thorough analysis, clear explanations, accurate identification of all components. Drawing the model fully.	Good analysis with most components correctly identified and explained. Drawing the model partially.	Basic analysis with some components identified and explained.	Incomplete analysis with many missing or incorrect components.
Video Presentation (50%)	Clear, professional presentation, in-depth explanations, good video quality and accessibility.	Good presentation, most explanations clear, decent video quality and accessibility.	Basic presentation, some explanations unclear, acceptable video quality and accessibility.	Poor presentation, unclear explanations, low video quality, or inaccessible video.

Reminder: **Your work will be graded comparatively based on the quality and depth of your analysis relative to other groups' work.**

Appendix

%matplotlib inline - A magic command in Jupyter Notebooks to display Matplotlib plots inline.

matplotlib.pyplot - A module in the Matplotlib library used for plotting graphs.

tensorflow - A library for machine learning and artificial intelligence.

numpy - A library for numerical computing in Python.

sys - A module that provides access to some variables used or maintained by the Python interpreter.

os - A module that provides a way of using operating system-dependent functionality.

PIL (from PIL import Image) - Python Imaging Library, used for opening, manipulating, and saving images.

backend - Keras backend module.

models - Keras model-related functions.

cache (from cache import cache) - A module typically used for caching purposes (Note: This is likely a custom or less common library, as it is not a standard Python library).

layers - Keras layers like Input, Dense, GRU, Embedding.

callbacks - Functions for training callbacks like ModelCheckpoint and TensorBoard.

preprocessing.text - Functions for text preprocessing like Tokenizer.

applications - Pre-trained models like VGG16.

optimizers - Optimization algorithms like RMSprop.

preprocessing.sequence - Functions for sequence preprocessing like pad_sequences.