Note: In order to understand how to execute this project, open the ReadMe file

CS 6384.001 Computer Vision - S22

Vivek Agrawal
VXA200020

Project 2 - Transfer Learning Project

Project Requirement:

In this project you are asked to do transfer learning on the "flowers" dataset from Kaggle.

The dataset has 13 classes. We will assume that classes are described in alphabetical order: classes = {astilbe, bellflower, black-eyed susan, calendula, california poppy, carnation, common daisy, coreopsis, dandelion, iris, rose, sunflower, tulip}.

In this project, I am classifying images of flowers by using transfer learning from a pre-trained network.

A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. I can either use the pre-trained model as is or use transfer learning to customize this model to the given task.

The intuition behind transfer learning for image classification is that if a model is trained on a large and general enough dataset, this model will effectively serve as a generic model of the visual world. You can then take advantage of these learned feature maps without having to start from scratch by training a large model on a large dataset.

The general machine learning workflow:

- Examine and understand the data
- Build an input pipeline, in this case using Keras ImageDataGenerator
- Compose the model
    - Load in the pretrained base model (and pretrained weights)
    - Stack the classification layers on top
- Train the model
- Evaluate model

Procedure:

- Download and extract a zip file containing the images, then create a tf.data.Dataset for training and validation using the tf.keras.utils.image_dataset_from_directory utility. We will need the flowers.zip file available in the directory which will be the base dataset.

Note: In order to understand how to execute this project, open the ReadMe file

- Then I split the dataset into Training and Validation dataset. The dataset has 12847 files belonging to 13 classes. We use 10278 files for training and 2569 files for validation.
- Then I display a sample of images belonging to different classes.

- As the original dataset doesn't contain a test set, you will create one. To do so, determine how many batches of data are available in the validation set using tf.data.experimental.cardinality, then move 20% of them to a test set.
- Then I configured the dataset for performance using tf.data.AUTOTUNE
- When you don't have a large image dataset, it's a good practice to artificially introduce sample diversity by applying random, yet realistic, transformations to the training images, such as rotation and horizontal flipping. This helps expose the model to different aspects of the training data and reduce overfitting.
- I used tf.keras.applications.MobileNetV2 as my base model. This model expects pixel values in [-1, 1], but at this point, the pixel values in my images are in [0, 255]. To rescale them, I used the preprocessing method included with the model.
- MobileNet V2 model was developed at Google. This is pre-trained on the ImageNet dataset, a large dataset consisting of 1.4M images and 1000 classes. ImageNet is a research training dataset with a wide variety of categories like jackfruit and syringe. This base of knowledge will help us classify flowers from our specific dataset.

- Then I added a classification head to generate predictions from the block of features, average over the spatial 5x5 spatial locations, using a tf.keras.layers.GlobalAveragePooling2D layer to convert the features to a single 1280-element vector per image.
- Then I build the model by chaining together the data augmentation, rescaling, base_model and feature extractor layers using the Keras Functional API. I used training=False as my model contains a Batch Normalization layer.
- I compiled the model once before training it with a base learning rate of 0.001. Since there are 13 classes, use the tf.keras.losses.SparseCategoricalCrossentropy loss with from_logits=True since the model provides a linear output.
- The 2.5 million parameters in MobileNet are frozen, but there are 16,653 trainable parameters in the Dense layer. These are divided between two tf.Variable objects, the weights and biases.
- The initial accuracy was 11% but after 5 epochs, the validation accuracy increased to 89.6%.
- Since this isn't enough, I fine tuned the model. I kept the first 50 layers frozen and unfroze the rest layers and training my model again. This time the new learning rate was 0.0001 and I ran it for 5 epochs which gave the new accuracy of 91%. I then tuned it even further for 3 more epochs which gave my final validation accuracy of 94.9%.
- I saved this model as VXA200020_model.h5. Then I further load my particular model to check if the model was properly saved.
- Then I verified the performance of the model on new data using test set. The final output was 97% of training accuracy and 95% of testing accuracy.
- All of these procedures and codes can be found in proj2.ipynb file.

Now explaining the steps involved in testing different folders with images using my given model:

- First I defined a function named load_model which loads my model into the code.
- Then I defined a function named decode_img function which takes my images and its dimensions and decodes a JPEG-encoded image to a uint8 tensor.
- Then finally I created a function named image_label which takes the data frame extracted from cv file, classes and image dimensions as the input parameters. It creates a list of different classes and stores the images in the decoded form and map them with their classes and forms the test dataset.
- The python file takes the model, weights- which are none in my case, and the csv file as arguments and creates the dataset using the csv file and labels and image path in them, loads the model in the program and runs that model over the test dataset to predict and classify the label and then gives an accuracy.
- For the given 4 test images given to us with this project, my model and code is giving 100% accuracy.

```
1/1 [==============================] - 1s 513ms/step - loss: 4.9768e-04 - accuracy: 1.0000
Test model, accuracy: 100.00000%
```

- All of these procedures and codes can be found in proj2_test.py file.