UVA Landmarks Dataset

Use transfer learning to recognize UVA historic landmarks

N. Rich Nguyen, PhD

SYS 6016

Problem Statement

UVA Grounds *is known for its Jeffersonian architecture and place in U.S. history as a model for college and university campuses throughout the country. Throughout its history, the University of Virginia has won praises for its unique* ***Jeffersonian architecture****. In this assignment, you will attempt the build an image recognition model to classify different buildings and landmarks on Grounds. You will earn up to 70 points for this assignment plus 10 bonus points if your classifier performs exceed 94% accuracy. You can use Tensorflow and Keras API and any existing CNN framework.*

Instruction

1. **LOAD:** Load the **UVaLandmark18** dataset from my Firebase server, the dataset comes with 14,286 images of 18 different buildings on Ground.
2. **CODE:** Using what we have learned, design your own neural network architecture or transfer any existing framework. Write code in a Google Colab, download as a .ipynb file and submit via UVACollab
3. **EVALUATE:** Upon returning the class accuracy on the test set (which is 20% of the dataset under a random seed=49). The maximum points you can earn is 70 pts plus up to 10 bonus pts. To earn the bonus points, your model must yield an accuracy of 95%.

From ML model to mobile app

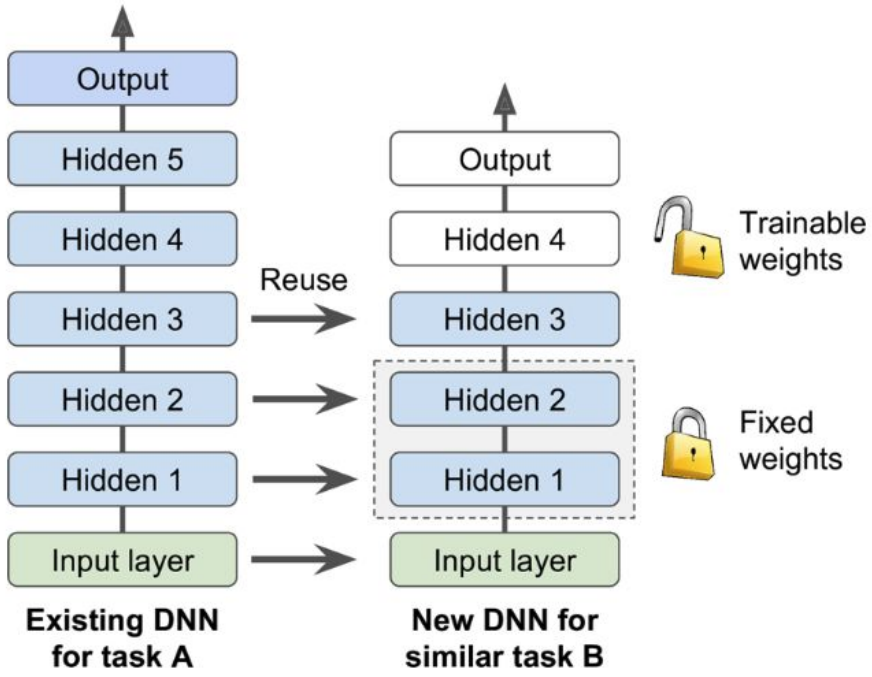
- Highly accurate model may be considered to be deploy in a mobile app my research group.
- You can download and try our mobile app following these steps:
  - For iOS devices, download the app here
  - For Android devices, download the app here
- Ask TAs and myself if you have any question. We are here to help!
- **Tip: You must consider the transfer learning of another existing CNN framework to achieve high performance.**

Transfer Learning

Transfer Learning with Keras

It is generally **not** a good idea to train a very large CNN from scratch, instead try to use an existing network that accomplishes a similar task. Transfer Learning not only speeds up training, but also requires less training data.

- New Input layer must have the same size.
- Hidden layers can be kept.
- New Output layer must be replaced and retrained for the new task



Using Pre-trained Models from Keras

In general, you won't have to implement models like GoogLeNet, or ResNet

Load models from keras.applications

```
model = keras.applications.resnet50.ResNet50(weights="imagenet")
```

To use it, you need to ensure your images have the right size since ResNet-50 model expects **224x224** pixel image.

```
images_resized = tf.image.resize(images, [224, 224])
```

```
inputs = keras.applications.resnet50.preprocess_input(images_resized * 255)
```

Train using Transfer Learning

To build an image classifier without lots of training data, it's often a good idea to reuse the lower layers of a pre-trained model.

We use Xception model pre-trained on ImageNet. We exclude the top layers of the network and add our own global average pooling layer, followed by the dense output layer with softmax.

After that, you can use pre-trained model to make predictions:

```
base_model = keras.applications.xception.Xception(weights="imagenet",
                                                  include_top=False)
avg = keras.layers.GlobalAveragePooling2D()(base_model.output)
output = keras.layers.Dense(n_classes, activation="softmax")(avg)
model = keras.Model(inputs=base_model.input, outputs=output)
```

Training the added top layers

It’s a good idea to freeze the weights of pretrained layers, in order to train the top layers. (Tip: remember to switch hardware acceleration to GPU on Colab)

```
for layer in base_model.layers:
    layer.trainable = False

optimizer = keras.optimizers.SGD(lr=0.2, momentum=0.9, decay=0.01)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set,
                    steps_per_epoch=int(0.75 * dataset_size / batch_size),
                    validation_data=valid_set,
                    validation_steps=int(0.15 * dataset_size / batch_size),
                    epochs=5)
```

Training the entire network

Now that the top layers in well trained, we are ready to unfreeze all the layers and continue training with a much lower learning rate so that you don't damage the pre-trained weights

```
for layer in base_model.layers:
    layer.trainable = True

optimizer = keras.optimizers.SGD(learning_rate=0.01, momentum=0.9,
                                 nesterov=True, decay=0.001)
model.compile(loss="sparse_categorical_crossentropy", optimizer=optimizer,
              metrics=["accuracy"])
history = model.fit(train_set,
                    steps_per_epoch=int(0.75 * dataset_size / batch_size),
                    validation_data=valid_set,
                    validation_steps=int(0.15 * dataset_size / batch_size),
                    epochs=40)
```

Good luck and have fun!