

```
!rm -rf /content/Landmark-classification-and-tagging
!git clone https://github.com/SwastikGorai/Landmark-classification-and-tagging
```

```
Cloning into 'Landmark-classification-and-tagging'...
remote: Enumerating objects: 6409, done.
remote: Counting objects: 100% (29/29), done.
remote: Compressing objects: 100% (22/22), done.
remote: Total 6409 (delta 12), reused 22 (delta 7), pack-reused 6380
Receiving objects: 100% (6409/6409), 739.72 MiB | 45.22 MiB/s, done.
Resolving deltas: 100% (15/15), done.
```

```
!mv -v Landmark-classification-and-tagging/* /content/
```

```
renamed 'Landmark-classification-and-tagging/app.ipynb' -> '/content/app.ipynb'
renamed 'Landmark-classification-and-tagging/best_val_loss.pt' -> '/content/best_val_loss.pt'
renamed 'Landmark-classification-and-tagging/cnn_from_scratch.ipynb' -> '/content/cnn_from_scratch.ipynb'
renamed 'Landmark-classification-and-tagging/mean_and_std.pt' -> '/content/mean_and_std.pt'
renamed 'Landmark-classification-and-tagging/original_exported.pt' -> '/content/original_exported.pt'
renamed 'Landmark-classification-and-tagging/README.md' -> '/content/README.md'
renamed 'Landmark-classification-and-tagging/requirements.txt' -> '/content/requirements.txt'
renamed 'Landmark-classification-and-tagging/src' -> '/content/src'
renamed 'Landmark-classification-and-tagging/static_images' -> '/content/static_images'
renamed 'Landmark-classification-and-tagging/transfer_learning.ipynb' -> '/content/transfer_learning.ipynb'
```

## ✓ Convolutional Neural Networks

### Project: Write an Algorithm for Landmark Classification

#### Transfer learning

In the previous notebook we have trained our own CNN and we got a certain performance. Let's see how hard it is to match that performance with transfer learning.

#### > Step 0: Setting up

The following cells make sure that your environment is setup correctly and check that your GPU is available and ready to go. You have to execute them every time you restart your notebook.

```
# Install requirements
!pip install -r requirements.txt -q
```

```
1.6/1.6 MB 34.9 MB/s eta 0:00:00
21.3/21.3 MB 75.1 MB/s eta 0:00:00
```

```
from src.helpers import setup_env
```

```
# If running locally, this will download dataset (make sure you have at
# least 2 Gb of space on your hard drive)
setup_env()
```

```
GPU available
Downloading and unzipping https://udacity-dlfn.s3-us-west-1.amazonaws.com/datasets/landmark\_images.zip. This will take a while...
done
Reusing cached mean and std
```

#### ✓ > Step 1: Create transfer learning architecture

Open the file `src/transfer.py` and complete the `get_model_transfer_learning` function. When you are done, execute this test:

```
!pytest -vv src/transfer.py
```

```
===== test session starts =====
platform linux -- Python 3.10.12, pytest-7.4.4, pluggy-1.5.0 -- /usr/bin/python3
cachedir: .pytest_cache
rootdir: /content
plugins: typeguard-4.3.0, anyio-3.7.1
collected 1 item
```

```
src/transfer.py::test_get_model_transfer_learning PASSED [100%]

===== warnings summary =====
src/transfer.py::test_get_model_transfer_learning
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since
warnings.warn(

src/transfer.py::test_get_model_transfer_learning
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` fo
warnings.warn(msg)

-- Docs: https://docs.pytest.org/en/stable/how-to/capture-warnings.html
===== 1 passed, 2 warnings in 3.17s =====
```

## ✓ > Step 2: Train, validation and test

Let's train our transfer learning model! Let's start defining the hyperparameters:

```
batch_size = 64 # size of the minibatch for stochastic gradient descent (or Adam)
valid_size = 0.2 # fraction of the training data to reserve for validation
num_epochs = 20 # number of epochs for training
num_classes = 50 # number of classes. Do not change this
learning_rate = 0.001 # Learning rate for SGD (or Adam)
opt = 'adam' # optimizer. 'sgd' or 'adam'
weight_decay = 0.001 # regularization. Increase this to combat overfitting

from src.data import get_data_loaders
from src.optimization import get_optimizer, get_loss
from src.train import optimize
from src.transfer import get_model_transfer_learning

# Get a model using get_model_transfer_learning. Use one of the names reported here:
# https://pytorch.org/vision/0.10/models.html
# For example, if you want to load ResNet 18, use "resnet18"
# NOTE: use the hyperparameters defined in the previous cell, do NOT copy/paste the
# values
model_transfer = get_model_transfer_learning("wide_resnet50_2", n_classes=num_classes)

# train the model
data_loaders = get_data_loaders(batch_size=batch_size)
optimizer = get_optimizer(
    model_transfer,
    learning_rate=learning_rate,
    optimizer=opt,
    weight_decay=weight_decay,
)
loss = get_loss()

optimize(
    data_loaders,
    model_transfer,
    optimizer,
    loss,
    n_epochs=num_epochs,
    save_path="checkpoints/model_transfer.pt",
    interactive_tracking=True
)
```



Test Accuracy: 75% (943/1250)

0.9546849593520164

## ✓ > Step 4: Export using torchscript

Now, just like we did with our original model, we export the best fit model using torchscript so that it can be used in our application:


```
from src.predictor import Predictor
from src.helpers import compute_mean_and_std

# First let's get the class names from our data loaders
class_names = data_loaders["train"].dataset.classes

# Then let's move the model_transfer to the CPU
# (we don't need GPU for inference)
model_transfer = model_transfer.cpu()
# Let's make sure we use the right weights by loading the
# best weights we have found during training
# NOTE: remember to use map_location='cpu' so the weights
# are loaded on the CPU (and not the GPU)
model_transfer.load_state_dict(
    torch.load("checkpoints/model_transfer.pt", map_location="cpu")
)

# Let's wrap our model using the predictor class
mean, std = compute_mean_and_std()
predictor = Predictor(model_transfer, class_names, mean, std).cpu()

# Export using torch.jit.script
scripted_predictor = torch.jit.script(predictor)
scripted_predictor.save("checkpoints/transfer_exported.pt")

 Reusing cached mean and std

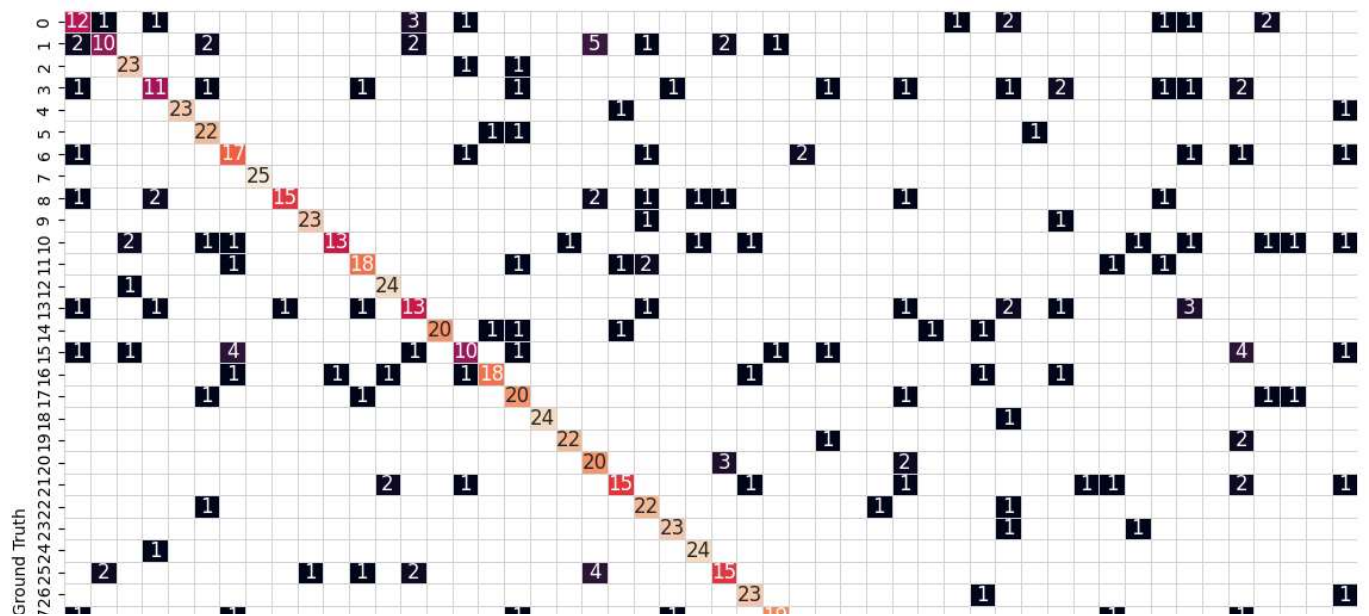
import torch
from src.predictor import predictor_test
from src.helpers import plot_confusion_matrix

model_reloaded = torch.jit.load("checkpoints/transfer_exported.pt")

pred, truth = predictor_test(data_loaders['test'], model_reloaded)

plot_confusion_matrix(pred, truth)
```

100% | 1250/1250 [09:49<00:00, 2.12it/s]  
Accuracy: 0.7584



Start coding or generate with AI.

