Asigned: 13 November

Homework #7

EE 541: Fall 2023

Due: Tuesday, 28 November at 23:59. Submission instructions will follow separately on canvas.

1. Disasters involving explosions and fires pose a substantial threat to human life and property. Managaging chemical fires is complex and requires accurate assessment of fuel sources. Martinka et al. [1] demonstrate a CNN-based approach to discriminate burning liquids using a static flame image. In this assignment you will use transfer learning to fine-tune a residual CNN to predict the same.

Dataset

Download the burning liquid dataset from https://doi.org/10.1007/s10973-021-10903-2 – Supplementary Information, File #2. The dataset consists of 3000 hi-resolution flame images of burning ethanol, pentane, and propanol.

(a) Extract the images into a `data` folder. Then split the images into subfolders named `ethanol`, `pentane`, and `propanol` based on their filename. This structure (shown below) allows you to use the standard PyTorch `ImageFolderDataset` class for the custom dataset. See the `torchvision` documentation for more detail: https://pytorch.org/vision/stable/datasets.html.

```
data/
└
   ├─ ethanol/
   │   └ [...]
   ├─ pentane/
   │   └ [...]
   └─ propanol/
       └ [...]
```

(b) Split training, validation, and testing sets using a ratio appropriate for the dataset size.

(c) Use PyTorch DataSet transforms to resize the images to the model's input dimension of $224 \times 224 \times 3$. See: https://pytorch.org/vision/stable/transforms.html. Then apply normalization and/or contrast enhancement to adjust the intensity-range. Include *reasonable* data augmentations such as rotations, flips, and scaling using.

Model

Load the pretrained `torchvision` ResNet-34 model. Replace the final classification output-layer to match the number of burning liquid classes.

```
import torch.nn as nn
from torchvision.models import resnet34

# https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py
# find, ResNet::forward, self.fc
num_classes = 3
model = resnet34(pretrained=True)
model.fc = nn.Linear(model.fc.in_features, num_classes)
```

Training

Fine-tune the pretrained model using the custom dataset. Experiment with reasonable learning rates, batch sizes, and training epochs.

Freeze layers so that the initially large classifier training gradients do not propagate through the pretrained feature extractor. Use the property `requires_grad = False` to freeze model parameters. For example, to freeze all layers except the new classifier output layer:

```
for param in model.parameters():
  param.requires_grad = False

for param in model.fc.parameters():
  param.requires_grad = True
```

Begin by freezing all layers except the fully connected classifier layer(s). Train the model with a small learning rate (*e.g.*, 1e-4) over several epochs. Then progressively unfreeze layers to adapt the pretrained model to the new dataset. Experiment with smaller smaller learning rates (*e.g.*, 1e-5) as performance plateaus.

Plot learning and accuracy curves for the training and validation sets. Include comments and/or annotate the figures to indicate when you adjusted layer freezing and changed the learning rate.

Layer visualization

Visualize the feature maps of several convolutional layers within the model. Activation intensity can provide insight and explainability of the internal representation.

Create feature maps of the first convolutional layer and a selection of layers from the middle of the network. The following snippet shows how to add a PyTorch hook to programatically capture layer outputs. Refer to the `torchvision` ResNet source code [2] or print a model *summary* to identify specific layers by name. Then use `torchvision.utils.make_grid` or similar to produce an image grid showing output activations for all C_{out} filters in a layer.

```
def visualize_hook(module, input, output):
  plt.figure(figsize=(15, 15))
  for i in range(output.size(1)):
      plt.subplot(8, 8, i + 1)
      plt.imshow(output[0, i].detach().cpu().numpy(), cmap="gray")
      plt.axis("off")
  plt.show()

# Choose a specific layer and register the hook
layer_to_visualize = model.conv1
hook = layer_to_visualize.register_forward_hook(visualize_hook)

# Run a single image through the model
image = torch.randn(1, 3, 224, 224)  # Replace this with a real image from the dataset
_ = model(image)

hook.remove()     # Remove the hook
```

Analysis

- Report the accuracy of the fine-tuned model on the testing set. Compare the accuracy to the baseline vanilla pretrained ResNet-34 model.

- Generate a confusion matrix to show inter-class error rates.

- The `sklearn.metrics` module provides several loss, score, and utility functions to measure classification performance. https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics.

  Create a precision-recall curve for each class. Precision-recall curves show the trade-off between the true positive rate (precision) and the positive predictive value (recall) as the discrimination threshold T varies from 0 to 1. They are a standard metric to compare binary classifiers. https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html.

  Calculate the precision and recall for each class by treating it prediciton as a binary classification (*i.e.*, one-vs-many). Then plot the P-R curves on the same plot. You may use `preprocessing.label_binarize` and `metrics.precision_recall_curve` from `sklearn`.

References

[1] Martinka, J., Nečas, A., Rantuch, P. The recognition of selected burning liquids by convolutional neural networks under laboratory conditions. J Therm Anal Calorim 147, 5787-5799 (2022).

[2] https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py