Ants and Bees - Classification to Demonstrate Transfer Learning

- This notebook is supposed to be an amateur's first foray into Transfer Learning.
- It operates on a dataset that is relatively small - with 120 images for ants and bees. There are 75 validation images for each class as well.
- Usually, this is a very small dataset to generalize upon, if trained from scratch. Since we are using transfer learning, we should be able to generalize reasonably well.

Downloading the Data

In [ ]:
```
from io import BytesIO
from typing import Union
from zipfile import ZipFile
import requests
from os import path, rename


"""Downloads a zip file from `url`, extracts it at `dest`, renames the extracted fold


def download_zip_and_extract(url: str, dest: str, new_dir_name: Union[str, None] = No
  file_name = ".".join(url.split('/')[-1].split('.')[:-1])
  dest_path = path.join(path.curdir, dest)

  if path.exists(path.join(dest_path, new_dir_name if new_dir_name else file_name)):
    print("Destination folder already exists")
    return


  print("Downloading started")
  req=requests.get(url)
  print("Downloading completed. Extraction started")

  zipfile=ZipFile(BytesIO(req.content))
  zipfile.extractall(dest_path)

  print("Extraction completed")

  if new_dir_name:
    rename(path.join(dest_path, file_name), new_dir_name)
```

In [ ]:
```
DATA_URL = "https://download.pytorch.org/tutorial/hymenoptera_data.zip"
data_dest = "."
download_zip_and_extract(DATA_URL, data_dest, "data")
```

```
Destination folder already exists
```

Relevant Imports

In [ ]:
```
import torch
from torch import nn, optim
from torch.optim import lr_scheduler
from torch.utils.data import DataLoader
import numpy as np
import torchvision
from torchvision import datasets, transforms as T, models
import matplotlib.pyplot as plt
import time
import os

plt.ion()
```

Out[ ]:
```
<matplotlib.pyplot._IonContext at 0x20f896af820>
```

Load Data

In [ ]:
```
data_transforms = {
    'train': T.Compose([
        T.RandomResizedCrop(224),
        T.RandomHorizontalFlip(),
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ]),
    'val': T.Compose([
        T.Resize(256),
        T.CenterCrop(224),
        T.ToTensor(),
        T.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])
    ])
}
```

In [ ]:
```
batch_size = 4
```

In [ ]:
```
data_dir = path.join(data_dest, "data")

image_datasets = {x: datasets.ImageFolder(
    os.path.join(data_dir, x), transform=data_transforms[x]) for x in ['train', 'val
dataloaders = {x: DataLoader(image_datasets[x], batch_size=batch_size, shuffle=(
    True if x == 'train' else False), num_workers=4) for x in ['train', 'val']}

dataset_sizes = {x: len(image_datasets[x]) for x in ['train', 'val']}
category_names = image_datasets['train'].classes
```

In [ ]:
```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
device
```

Out[ ]:
```
device(type='cuda')
```

Visualising a Few Images

- We'll plot a few images to see the effects of the data transforms.

In [ ]:
```
from torch import Tensor


def imshow(inp: Tensor, title=None):
  """Plot image if `inp` is a tensor"""

  inp = inp.cpu().numpy().transpose((1, 2, 0))
  mean = np.array([0.485, 0.456, 0.406])
  std = np.array([0.229, 0.224, 0.225])

  inp = std * inp + mean
  np.clip(inp, 0, 1)

  if title is not None:
    plt.title(title)

  return plt.imshow(inp)
```

In [ ]:
```
# Get a batch of training data
inputs, classes = next(iter(dataloaders['train']))

# make a grid from batch
out = torchvision.utils.make_grid(inputs)

imshow(out, [category_names[x] for x in classes])
```

Out[ ]:
```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
<matplotlib.image.AxesImage at 0x20f909d2df0>
```

Training the Model

It will help us to:

- schedule the learning rate
- saving the best model

In [ ]:

```
from copy import deepcopy
from torch.nn import Module


def train_model(model: Module, criterion, optimizer: optim.Optimizer, scheduler, num_
  since = time.time()

  best_model_wts = deepcopy(model.state_dict())
  best_acc = 0.0

  for epoch in range(num_epochs):
    print(f"Epoch: {epoch} / {num_epochs}")

    for phase in ['train', 'val']:
      if phase == 'train':
        model.train()

      else:
        model.eval()

      running_loss, running_corrects = 0.0, 0

      for inputs, labels in dataloaders[phase]:
        inputs, labels = inputs.to(device), labels.to(device)

        # zero all the layers' parameters gradients
        optimizer.zero_grad()

        # feed forward
        with torch.set_grad_enabled(phase == 'train'):
          outputs = model(inputs)
          _, preds = torch.max(outputs, dim=1)
          loss = criterion(outputs, labels)

          # backpropagate loss only if training
          if phase == 'train':
            loss.backward()
            optimizer.step()

        # default loss item is mean, therefore we multiply it with the number of iter
        running_loss += loss.item() * inputs.size(0)
        running_corrects += torch.sum(preds == labels)

      if phase == 'train':
        scheduler.step()

      epoch_loss = running_loss / dataset_sizes[phase]
      epoch_acc = running_corrects.double() / dataset_sizes[phase]

      print(f'{phase} Loss: {epoch_loss:.4f} Acc: {epoch_acc:.4f}')

      # deep copy the model
      if phase == 'val' and epoch_acc > best_acc:
        best_acc = epoch_acc
        best_model_wts = deepcopy(model.state_dict())

      print()
```

```
  time_elapsed = time.time() - since
  print(
      f'Training complete in {time_elapsed // 60:.0f}m {time_elapsed % 60:.0f}s')
  print(f'Best val Acc: {best_acc:4f}')

  model.load_state_dict(best_model_wts)
  return model
```

Visualising Model Predictions

In [ ]:

```
from matplotlib.image import AxesImage


def process_tensor_to_display(inp: Tensor) -> np.ndarray:
  inp = inp.cpu().numpy().transpose((1, 2, 0))
  mean = np.array([0.485, 0.456, 0.406])
  std = np.array([0.229, 0.224, 0.225])

  inp = std * inp + mean
  np.clip(inp, 0, 1)

  return inp
```

In [ ]:

```
def visualise_predictions(model: Module, num_images: int, n_rows: int, n_cols: int):
  was_training = model.training
  model.eval()
  images_so_far = 0
  fig, ax = plt.subplots(n_rows, n_cols)

  with torch.no_grad():
    for i, (inputs, labels) in enumerate(dataloaders['val']):
      inputs, labels = inputs.to(device), labels.to(device)

      outputs = model(inputs)
      _, preds = torch.max(outputs, dim=1)

      for j in range(inputs.size(0)):

        # below call takes number of rows, number of cols, index of item that starts
        plotX, plotY = images_so_far // n_cols, images_so_far % n_cols
        ax[plotX, plotY].imshow(process_tensor_to_display(inputs[j].detach()))
        ax[plotX, plotY].axis('off')
        ax[plotX, plotY].set_title(f"{category_names[preds[j]]}")
        images_so_far += 1

        # imshow(inputs[j].detach())

        if images_so_far == num_images:
          model.train(was_training)
          return

  model.train(was_training)
```

Finetuning the convnet

Load a pretrained model and reset final fully connected layer

In [ ]:
```
model_ft = models.resnet18(pretrained=True)
num_ftrs = model_ft.fc.in_features

model_ft.fc = nn.Linear(num_ftrs, len(category_names))

model_ft = model_ft.to(device)
```

In [ ]:
```
criterion = nn.CrossEntropyLoss()

# Observe that all parameters are being optimized
optimizer_ft = optim.SGD(model_ft.parameters(), lr=0.001, momentum=0.9)

# Decay LR by a factor of 0.1 every 7 epochs
exp_lr_scheduler = lr_scheduler.StepLR(optimizer_ft, 7, gamma=0.1)
```

In [ ]:
```
model_ft = train_model(model_ft, criterion, optimizer_ft, exp_lr_scheduler, num_epoch
```

```
Epoch: 0 / 30
d:\ProgramFiles\anaconda\envs\fastapi-ml\lib\site-packages\torch\nn\functional.py:71
8: UserWarning: Named tensors and all their associated APIs are an experimental featu
re and subject to change. Please do not use them for anything important until they ar
e released as stable. (Triggered internally at  ..\c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
train Loss: 0.5452 Acc: 0.6844

val Loss: 0.2099 Acc: 0.9216

Epoch: 1 / 30
train Loss: 0.4097 Acc: 0.8279

val Loss: 0.2839 Acc: 0.8824

Epoch: 2 / 30
train Loss: 0.5586 Acc: 0.7746

val Loss: 0.3774 Acc: 0.8366

Epoch: 3 / 30
train Loss: 0.5762 Acc: 0.7418

val Loss: 0.2786 Acc: 0.8889

Epoch: 4 / 30
train Loss: 0.4245 Acc: 0.8033

val Loss: 0.3033 Acc: 0.8889

Epoch: 5 / 30
train Loss: 0.5666 Acc: 0.7746

val Loss: 0.3868 Acc: 0.8497
```

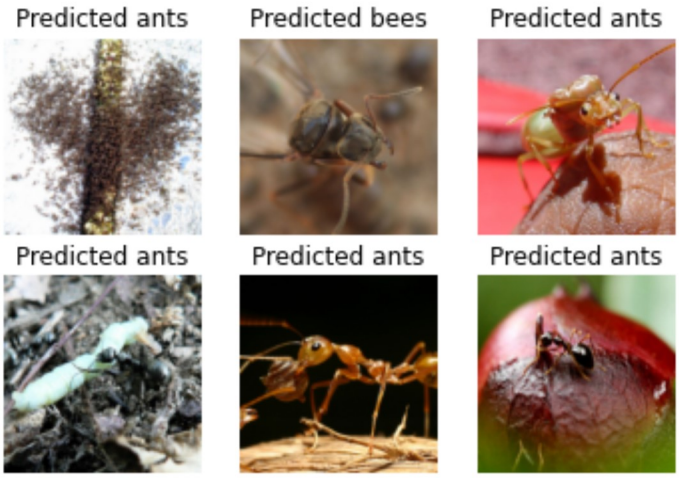
```
Epoch: 6 / 30
train Loss: 0.3811 Acc: 0.8648

val Loss: 0.3813 Acc: 0.8889

Epoch: 7 / 30
train Loss: 0.4489 Acc: 0.7951

val Loss: 0.2856 Acc: 0.9150

Epoch: 8 / 30
train Loss: 0.3303 Acc: 0.8525

val Loss: 0.2610 Acc: 0.9281

Epoch: 9 / 30
train Loss: 0.4298 Acc: 0.8361

val Loss: 0.2819 Acc: 0.9020

Epoch: 10 / 30
train Loss: 0.3481 Acc: 0.8320

val Loss: 0.2554 Acc: 0.9216

Epoch: 11 / 30
train Loss: 0.2995 Acc: 0.8852

val Loss: 0.2408 Acc: 0.9281

Epoch: 12 / 30
train Loss: 0.3145 Acc: 0.8566

val Loss: 0.2455 Acc: 0.9216

Epoch: 13 / 30
train Loss: 0.3038 Acc: 0.8361

val Loss: 0.2469 Acc: 0.9216

Epoch: 14 / 30
train Loss: 0.2442 Acc: 0.8975

val Loss: 0.2867 Acc: 0.8954

Epoch: 15 / 30
train Loss: 0.2736 Acc: 0.8648

val Loss: 0.2582 Acc: 0.9020

Epoch: 16 / 30
train Loss: 0.3095 Acc: 0.8525

val Loss: 0.2468 Acc: 0.9216

Epoch: 17 / 30
train Loss: 0.2819 Acc: 0.8730

val Loss: 0.2432 Acc: 0.9281
```

```
Epoch: 18 / 30
train Loss: 0.2765 Acc: 0.8607

val Loss: 0.2443 Acc: 0.9216

Epoch: 19 / 30
train Loss: 0.2397 Acc: 0.8852

val Loss: 0.2364 Acc: 0.9281

Epoch: 20 / 30
train Loss: 0.3606 Acc: 0.8279

val Loss: 0.2317 Acc: 0.9346

Epoch: 21 / 30
train Loss: 0.3057 Acc: 0.8770

val Loss: 0.2963 Acc: 0.9216

Epoch: 22 / 30
train Loss: 0.2689 Acc: 0.8811

val Loss: 0.2381 Acc: 0.9216

Epoch: 23 / 30
train Loss: 0.2571 Acc: 0.9016

val Loss: 0.2515 Acc: 0.9216

Epoch: 24 / 30
train Loss: 0.2707 Acc: 0.8934

val Loss: 0.2342 Acc: 0.9216

Epoch: 25 / 30
train Loss: 0.3202 Acc: 0.8648

val Loss: 0.2300 Acc: 0.9281

Epoch: 26 / 30
train Loss: 0.2434 Acc: 0.9057

val Loss: 0.2707 Acc: 0.9150

Epoch: 27 / 30
train Loss: 0.3011 Acc: 0.8689

val Loss: 0.2687 Acc: 0.9020

Epoch: 28 / 30
train Loss: 0.2325 Acc: 0.9098

val Loss: 0.2501 Acc: 0.9216

Epoch: 29 / 30
train Loss: 0.2538 Acc: 0.9057
```

```
val Loss: 0.2400 Acc: 0.9281

Training complete in 2m 34s
```

In [ ]:
```
visualise_predictions(model_ft, 6, 2, 3)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```



ConvNet as a Fixed Feature Extractor

- Here we will freeze all the layers of the network, except for the final layer.
- We need to set `requires_grad = False` for all those layers, so that their loss is not computed during the backward pass.

In [ ]:
```
model_conv = models.resnet34(pretrained=True)

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

```
Downloading: "https://download.pytorch.org/models/resnet34-b627a593.pth" to C:\Users\
janma/.cache\torch\hub\checkpoints\resnet34-b627a593.pth
100%|████████| 83.3M/83.3M [02:23<00:00, 610kB/s]
```

In [ ]:
```
num_ftrs = model_conv.fc.in_features
model_conv.fc = nn.Linear(num_ftrs, len(category_names))

model_conv.to(device)
```

Out[ ]:
```
ResNet(
  (conv1): Conv2d(3, 64, kernel_size=(7, 7), stride=(2, 2), padding=(3, 3), bias=Fals
e)
  (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=Tr
```

```
ue)
  (relu): ReLU(inplace=True)
  (maxpool): MaxPool2d(kernel_size=3, stride=2, padding=1, dilation=1, ceil_mode=Fals
e)
  (layer1): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    )
    (1): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias
=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stat
s=True)
    )
  )
  (layer2): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bia
s=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
```

```
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
  )
  (layer3): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(128, 256, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (downsample): Sequential(
        (0): Conv2d(128, 256, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
```

```
    (2): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
    (3): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
    (4): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
    (5): BasicBlock(
      (conv1): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(256, 256, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
  )
  (layer4): Sequential(
    (0): BasicBlock(
      (conv1): Conv2d(256, 512, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (downsample): Sequential(
        (0): Conv2d(256, 512, kernel_size=(1, 1), stride=(2, 2), bias=False)
        (1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
```

```
ts=True)
      )
    )
    (1): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
    (2): BasicBlock(
      (conv1): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn1): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(512, 512, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bi
as=False)
      (bn2): BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_sta
ts=True)
    )
  )
  (avgpool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=512, out_features=2, bias=True)
```

In [ ]:

```
criterion = nn.CrossEntropyLoss()

optimizer = optim.SGD(model_conv.parameters(), 0.001, 0.9)

exp_lr_scheduler = lr_scheduler.StepLR(optimizer, 7, 0.01)
```

Train and Evaluate

In [ ]:

```
model_conv = train_model(model_conv, criterion, optimizer, exp_lr_scheduler)
```

```
Epoch: 0 / 25
d:\ProgramFiles\anaconda\envs\fastapi-ml\lib\site-packages\torch\nn\functional.py:71
8: UserWarning: Named tensors and all their associated APIs are an experimental featu
re and subject to change. Please do not use them for anything important until they ar
e released as stable. (Triggered internally at  ..\c10/core/TensorImpl.h:1156.)
  return torch.max_pool2d(input, kernel_size, stride, padding, dilation, ceil_mode)
train Loss: 0.6462 Acc: 0.6598

val Loss: 0.2131 Acc: 0.9412

Epoch: 1 / 25
train Loss: 0.4524 Acc: 0.7828

val Loss: 0.3041 Acc: 0.8693

Epoch: 2 / 25
train Loss: 0.4432 Acc: 0.7992
```

```
val Loss: 0.1619 Acc: 0.9216

Epoch: 3 / 25
train Loss: 0.3744 Acc: 0.8279

val Loss: 0.2635 Acc: 0.8824

Epoch: 4 / 25
train Loss: 0.4363 Acc: 0.8156

val Loss: 0.2326 Acc: 0.9085

Epoch: 5 / 25
train Loss: 0.4172 Acc: 0.8156

val Loss: 0.1654 Acc: 0.9281

Epoch: 6 / 25
train Loss: 0.2964 Acc: 0.8402

val Loss: 0.2046 Acc: 0.9150

Epoch: 7 / 25
train Loss: 0.3512 Acc: 0.8525

val Loss: 0.1634 Acc: 0.9346

Epoch: 8 / 25
train Loss: 0.2824 Acc: 0.8689

val Loss: 0.1702 Acc: 0.9281

Epoch: 9 / 25
train Loss: 0.2675 Acc: 0.8770

val Loss: 0.1388 Acc: 0.9412

Epoch: 10 / 25
train Loss: 0.3115 Acc: 0.8566

val Loss: 0.1635 Acc: 0.9412

Epoch: 11 / 25
train Loss: 0.3183 Acc: 0.8607

val Loss: 0.1524 Acc: 0.9412

Epoch: 12 / 25
train Loss: 0.2740 Acc: 0.8770

val Loss: 0.1470 Acc: 0.9216

Epoch: 13 / 25
train Loss: 0.2642 Acc: 0.8689

val Loss: 0.1840 Acc: 0.9281

Epoch: 14 / 25
```

```
train Loss: 0.3747 Acc: 0.8566

val Loss: 0.1483 Acc: 0.9346

Epoch: 15 / 25
train Loss: 0.3237 Acc: 0.8648

val Loss: 0.1414 Acc: 0.9412

Epoch: 16 / 25
train Loss: 0.3138 Acc: 0.8730

val Loss: 0.1485 Acc: 0.9542

Epoch: 17 / 25
train Loss: 0.2813 Acc: 0.8811

val Loss: 0.1400 Acc: 0.9412

Epoch: 18 / 25
train Loss: 0.2980 Acc: 0.8648

val Loss: 0.1458 Acc: 0.9542

Epoch: 19 / 25
train Loss: 0.2572 Acc: 0.8975

val Loss: 0.1490 Acc: 0.9346

Epoch: 20 / 25
train Loss: 0.3071 Acc: 0.8648

val Loss: 0.2091 Acc: 0.9085

Epoch: 21 / 25
train Loss: 0.3139 Acc: 0.8689

val Loss: 0.1493 Acc: 0.9412

Epoch: 22 / 25
train Loss: 0.2518 Acc: 0.9016

val Loss: 0.1556 Acc: 0.9412

Epoch: 23 / 25
train Loss: 0.2769 Acc: 0.9016

val Loss: 0.1498 Acc: 0.9412

Epoch: 24 / 25
train Loss: 0.3111 Acc: 0.8566

val Loss: 0.1468 Acc: 0.9542

Training complete in 3m 37s
Best val Acc: 0.954248
```

In [ ]:

```
visualise_predictions(model_conv, 10, 2, 5)
```

```
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
Clipping input data to the valid range for imshow with RGB data ([0..1] for floats or
[0..255] for integers).
```

In [ ]: