convnext-normal

February 27, 2026

1 Galaxy Zoo 2: ConvNeXt Transfer Learning

This notebook demonstrates transfer learning for the Galaxy Zoo 2 dataset using PyTorch and ConvNeXt.

1.1 1. Import Required Libraries and Utilities

Import all necessary libraries, including torch, torchvision, numpy, pandas, matplotlib, and any utility functions/classes reused from the original notebook.

[14]:
```
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torchvision.models as models
from torch.utils.data import DataLoader, Dataset, Subset
import os
from PIL import Image
from torchvision import transforms
import glob
import concurrent.futures
from torchvision.models import ConvNeXt_Small_Weights
import time
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
```

[15]:
```
import zipfile

def unzip_file(zip_path, extract_to):
    with zipfile.ZipFile(zip_path, 'r') as zip_ref:
        zip_ref.extractall(extract_to)

base_path = "/kaggle/input/competitions/galaxy-zoo-the-galaxy-challenge"
output_path = "/kaggle/working/galaxy_zoo"

os.makedirs(output_path, exist_ok=True)
```

```
unzip_file(f"{base_path}/images_training_rev1.zip", output_path)
unzip_file(f"{base_path}/images_test_rev1.zip", output_path)
unzip_file(f"{base_path}/training_solutions_rev1.zip", output_path)
```

[16]:
```
from torchvision import transforms

train_transform = transforms.Compose([
    transforms.CenterCrop(384),    # replaces manual crop
    transforms.Resize((224, 224)),
    transforms.ToTensor(),
    transforms.Normalize(
        mean=[0.485, 0.456, 0.406],
        std=[0.229, 0.224, 0.225]
    )
])
```

1.2 3. Define GalaxyZooTensorDataset Class

Reuse the GalaxyZooTensorDataset class for loading preprocessed tensor images and labels.

[17]:
```
class GalaxyZooImageDataset(Dataset):
    def __init__(self, csv_file, image_dir, transform=None):
        self.image_dir = image_dir
        self.transform = transform

        if csv_file is not None:
            self.df = pd.read_csv(csv_file)
            self.ids = self.df.iloc[:, 0].values
            self.labels = self.df.iloc[:, 1:].values.astype(np.float32)
            self.has_labels = True
        else:
            self.df = None
            self.ids = [
                os.path.splitext(f)[0]
                for f in os.listdir(image_dir)
                if f.endswith(".jpg")
            ]
            self.labels = None
            self.has_labels = False

    def __len__(self):
        return len(self.ids)

    def __getitem__(self, idx):
        galaxy_id = self.ids[idx]
        img_path = os.path.join(self.image_dir, f"{galaxy_id}.jpg")
        image = Image.open(img_path).convert("RGB")

        if self.transform:
            image = self.transform(image)

        if self.has_labels:
            label = torch.tensor(self.labels[idx])
            return image, label
        else:
            return image, galaxy_id
```

1.3 4. Prepare Data

[18]:
```
# Load training solutions and dataset
csv_file = '/kaggle/working/galaxy_zoo/training_solutions_rev1.csv'
image_dir = '/kaggle/working/galaxy_zoo/images_training_rev1'
dataset = GalaxyZooImageDataset(csv_file, image_dir, transform=train_transform)
```

1.4 5. Create DataLoader

[19]:
```
batch_size = 512

full_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True,␣
 ↪num_workers=4, pin_memory=True)
```

1.5 6. Load and Modify ConvNeXt Model

Load the ConvNeXt model, freeze all layers except the classifier, and modify the final layer to match the number of classes.

[20]:
```
convnext = models.convnext_small(weights=ConvNeXt_Small_Weights.IMAGENET1K_V1)
num_classes = dataset.labels.shape[1]

# Freeze everything first
for param in convnext.parameters():
    param.requires_grad = False

# Replace final classifier layer
in_features = convnext.classifier[2].in_features
convnext.classifier[2] = nn.Linear(in_features, num_classes)

# Unfreeze last two feature stages and classifier, collect params for optimizer
backbone_params = []
classifier_params = []

for stage in convnext.features[-2:]:
    for param in stage.parameters():
        param.requires_grad = True
        backbone_params.append(param)

for param in convnext.classifier.parameters():
    param.requires_grad = True
    classifier_params.append(param)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
convnext = convnext.to(device)
```

1.6 7. Set Up Loss Function and Optimizer

Set up the BCEWithLogitsLoss and Adam optimizer, filtering parameters as in the original notebook.

[22]:
```
criterion = nn.BCEWithLogitsLoss()
optimizer = torch.optim.AdamW(
    [
        {"params": backbone_params, "lr": 1e-4},
        {"params": classifier_params, "lr": 1e-3},
    ],
    weight_decay=1e-4,
)
```

1.7 8. Train Model

[23]:
```
def train_one_stage(model, loader, optimizer, criterion, device, num_epochs=20,␣
 ↪stage_name="Stage"):
    model.train()
    losses = []

    for epoch in range(num_epochs):
        running_loss = 0.0
        epoch_start = time.time()

        for batch_idx, (images, labels) in enumerate(loader):
            images = images.to(device, non_blocking=True)
            labels = labels.to(device, non_blocking=True)

            optimizer.zero_grad(set_to_none=True)
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()

            running_loss += loss.item() * images.size(0)
            # Debug every 30 batches
            if (batch_idx + 1) % 30 == 0:
                current_avg = running_loss / ((batch_idx + 1) * loader.
↪batch_size)
                print(
                    f"{stage_name} Epoch {epoch+1}/{num_epochs} "
                    f"Batch {batch_idx+1}/{len(loader)} "
                    f"Loss={loss.item():.4f} "
                    f"RunningAvg={current_avg:.4f}"
                )

        avg_loss = running_loss / len(loader.dataset)
        losses.append(avg_loss)

        print(
            f"{stage_name} Epoch {epoch+1}/{num_epochs}: "
            f"Loss={avg_loss:.4f}, "
            f"Time={time.time()-epoch_start:.2f}s"
        )

    return losses
```

1.8 9. Save Model Checkpoints

[24]:
```
def save_checkpoint(model, optimizer, train_losses, filename):
    checkpoint = {
        'model_state_dict': model.state_dict(),
        'optimizer_state_dict': optimizer.state_dict(),
        'train_losses': train_losses
    }
    torch.save(checkpoint, filename)
    print(f"Checkpoint saved as {filename}")
```

[25]:
```
train_losses = []
```

[26]:
```
stage_name = "2_layer"
losses = train_one_stage(convnext, full_loader, optimizer, criterion, device,␣
↪num_epochs=20, stage_name=stage_name)
save_checkpoint(convnext, optimizer, losses, f"checkpoint_{stage_name}.pth")
train_losses.extend(losses)
```

```
2_layer Epoch 1/20 Batch 30/121 Loss=0.2855 RunningAvg=0.3302
2_layer Epoch 1/20 Batch 60/121 Loss=0.2605 RunningAvg=0.3014
2_layer Epoch 1/20 Batch 90/121 Loss=0.2613 RunningAvg=0.2889
2_layer Epoch 1/20 Batch 120/121 Loss=0.2595 RunningAvg=0.2814
2_layer Epoch 1/20: Loss=0.2814, Time=341.34s
2_layer Epoch 2/20 Batch 30/121 Loss=0.2586 RunningAvg=0.2572
2_layer Epoch 2/20 Batch 60/121 Loss=0.2545 RunningAvg=0.2556
```

```
2_layer Epoch 2/20 Batch 90/121 Loss=0.2563 RunningAvg=0.2551
2_layer Epoch 2/20 Batch 120/121 Loss=0.2488 RunningAvg=0.2542
2_layer Epoch 2/20: Loss=0.2542, Time=340.98s
2_layer Epoch 3/20 Batch 30/121 Loss=0.2526 RunningAvg=0.2512
2_layer Epoch 3/20 Batch 60/121 Loss=0.2480 RunningAvg=0.2508
2_layer Epoch 3/20 Batch 90/121 Loss=0.2512 RunningAvg=0.2506
2_layer Epoch 3/20 Batch 120/121 Loss=0.2481 RunningAvg=0.2504
2_layer Epoch 3/20: Loss=0.2504, Time=341.21s
2_layer Epoch 4/20 Batch 30/121 Loss=0.2502 RunningAvg=0.2492
2_layer Epoch 4/20 Batch 60/121 Loss=0.2496 RunningAvg=0.2488
2_layer Epoch 4/20 Batch 90/121 Loss=0.2479 RunningAvg=0.2487
2_layer Epoch 4/20 Batch 120/121 Loss=0.2515 RunningAvg=0.2484
2_layer Epoch 4/20: Loss=0.2484, Time=341.12s
2_layer Epoch 5/20 Batch 30/121 Loss=0.2442 RunningAvg=0.2472
2_layer Epoch 5/20 Batch 60/121 Loss=0.2454 RunningAvg=0.2478
2_layer Epoch 5/20 Batch 90/121 Loss=0.2381 RunningAvg=0.2474
2_layer Epoch 5/20 Batch 120/121 Loss=0.2425 RunningAvg=0.2473
2_layer Epoch 5/20: Loss=0.2473, Time=341.10s
2_layer Epoch 6/20 Batch 30/121 Loss=0.2508 RunningAvg=0.2462
2_layer Epoch 6/20 Batch 60/121 Loss=0.2406 RunningAvg=0.2459
2_layer Epoch 6/20 Batch 90/121 Loss=0.2446 RunningAvg=0.2462
2_layer Epoch 6/20 Batch 120/121 Loss=0.2423 RunningAvg=0.2461
2_layer Epoch 6/20: Loss=0.2462, Time=341.48s
2_layer Epoch 7/20 Batch 30/121 Loss=0.2522 RunningAvg=0.2452
2_layer Epoch 7/20 Batch 60/121 Loss=0.2459 RunningAvg=0.2456
2_layer Epoch 7/20 Batch 90/121 Loss=0.2364 RunningAvg=0.2454
2_layer Epoch 7/20 Batch 120/121 Loss=0.2459 RunningAvg=0.2453
2_layer Epoch 7/20: Loss=0.2453, Time=341.27s
2_layer Epoch 8/20 Batch 30/121 Loss=0.2449 RunningAvg=0.2452
2_layer Epoch 8/20 Batch 60/121 Loss=0.2462 RunningAvg=0.2447
2_layer Epoch 8/20 Batch 90/121 Loss=0.2408 RunningAvg=0.2445
2_layer Epoch 8/20 Batch 120/121 Loss=0.2445 RunningAvg=0.2447
2_layer Epoch 8/20: Loss=0.2447, Time=341.10s
2_layer Epoch 9/20 Batch 30/121 Loss=0.2378 RunningAvg=0.2442
2_layer Epoch 9/20 Batch 60/121 Loss=0.2437 RunningAvg=0.2442
2_layer Epoch 9/20 Batch 90/121 Loss=0.2415 RunningAvg=0.2440
2_layer Epoch 9/20 Batch 120/121 Loss=0.2471 RunningAvg=0.2440
2_layer Epoch 9/20: Loss=0.2441, Time=341.62s
2_layer Epoch 10/20 Batch 30/121 Loss=0.2455 RunningAvg=0.2429
2_layer Epoch 10/20 Batch 60/121 Loss=0.2428 RunningAvg=0.2434
2_layer Epoch 10/20 Batch 90/121 Loss=0.2470 RunningAvg=0.2432
2_layer Epoch 10/20 Batch 120/121 Loss=0.2441 RunningAvg=0.2436
2_layer Epoch 10/20: Loss=0.2436, Time=340.31s
2_layer Epoch 11/20 Batch 30/121 Loss=0.2433 RunningAvg=0.2420
2_layer Epoch 11/20 Batch 60/121 Loss=0.2416 RunningAvg=0.2426
2_layer Epoch 11/20 Batch 90/121 Loss=0.2453 RunningAvg=0.2428
2_layer Epoch 11/20 Batch 120/121 Loss=0.2453 RunningAvg=0.2431
2_layer Epoch 11/20: Loss=0.2431, Time=340.40s
```

```
2_layer Epoch 12/20 Batch 30/121 Loss=0.2419 RunningAvg=0.2427
2_layer Epoch 12/20 Batch 60/121 Loss=0.2413 RunningAvg=0.2427
2_layer Epoch 12/20 Batch 90/121 Loss=0.2453 RunningAvg=0.2427
2_layer Epoch 12/20 Batch 120/121 Loss=0.2498 RunningAvg=0.2427
2_layer Epoch 12/20: Loss=0.2427, Time=340.95s
2_layer Epoch 13/20 Batch 30/121 Loss=0.2431 RunningAvg=0.2418
2_layer Epoch 13/20 Batch 60/121 Loss=0.2377 RunningAvg=0.2425
2_layer Epoch 13/20 Batch 90/121 Loss=0.2418 RunningAvg=0.2423
2_layer Epoch 13/20 Batch 120/121 Loss=0.2471 RunningAvg=0.2423
2_layer Epoch 13/20: Loss=0.2423, Time=341.40s
2_layer Epoch 14/20 Batch 30/121 Loss=0.2398 RunningAvg=0.2409
2_layer Epoch 14/20 Batch 60/121 Loss=0.2459 RunningAvg=0.2416
2_layer Epoch 14/20 Batch 90/121 Loss=0.2400 RunningAvg=0.2418
2_layer Epoch 14/20 Batch 120/121 Loss=0.2386 RunningAvg=0.2419
2_layer Epoch 14/20: Loss=0.2419, Time=341.27s
2_layer Epoch 15/20 Batch 30/121 Loss=0.2427 RunningAvg=0.2414
2_layer Epoch 15/20 Batch 60/121 Loss=0.2433 RunningAvg=0.2415
2_layer Epoch 15/20 Batch 90/121 Loss=0.2461 RunningAvg=0.2418
2_layer Epoch 15/20 Batch 120/121 Loss=0.2418 RunningAvg=0.2415
2_layer Epoch 15/20: Loss=0.2415, Time=341.59s
2_layer Epoch 16/20 Batch 30/121 Loss=0.2450 RunningAvg=0.2414
2_layer Epoch 16/20 Batch 60/121 Loss=0.2357 RunningAvg=0.2408
2_layer Epoch 16/20 Batch 90/121 Loss=0.2457 RunningAvg=0.2407
2_layer Epoch 16/20 Batch 120/121 Loss=0.2370 RunningAvg=0.2412
2_layer Epoch 16/20: Loss=0.2412, Time=341.80s
2_layer Epoch 17/20 Batch 30/121 Loss=0.2417 RunningAvg=0.2406
2_layer Epoch 17/20 Batch 60/121 Loss=0.2469 RunningAvg=0.2408
2_layer Epoch 17/20 Batch 90/121 Loss=0.2407 RunningAvg=0.2411
2_layer Epoch 17/20 Batch 120/121 Loss=0.2405 RunningAvg=0.2409
2_layer Epoch 17/20: Loss=0.2409, Time=340.85s
2_layer Epoch 18/20 Batch 30/121 Loss=0.2427 RunningAvg=0.2407
2_layer Epoch 18/20 Batch 60/121 Loss=0.2438 RunningAvg=0.2404
2_layer Epoch 18/20 Batch 90/121 Loss=0.2436 RunningAvg=0.2403
2_layer Epoch 18/20 Batch 120/121 Loss=0.2470 RunningAvg=0.2405
2_layer Epoch 18/20: Loss=0.2405, Time=341.17s
2_layer Epoch 19/20 Batch 30/121 Loss=0.2420 RunningAvg=0.2403
2_layer Epoch 19/20 Batch 60/121 Loss=0.2369 RunningAvg=0.2400
2_layer Epoch 19/20 Batch 90/121 Loss=0.2421 RunningAvg=0.2402
2_layer Epoch 19/20 Batch 120/121 Loss=0.2385 RunningAvg=0.2403
2_layer Epoch 19/20: Loss=0.2403, Time=340.90s
2_layer Epoch 20/20 Batch 30/121 Loss=0.2410 RunningAvg=0.2400
2_layer Epoch 20/20 Batch 60/121 Loss=0.2403 RunningAvg=0.2398
2_layer Epoch 20/20 Batch 90/121 Loss=0.2413 RunningAvg=0.2399
2_layer Epoch 20/20 Batch 120/121 Loss=0.2421 RunningAvg=0.2400
2_layer Epoch 20/20: Loss=0.2400, Time=341.03s
Checkpoint saved as checkpoint_2_layer.pth
```

1.9 10. Save Training Loss

[27]:
```
loss_df = pd.DataFrame({'epoch': list(range(1, len(losses)+1)), 'loss': losses})
loss_df.to_csv('train_losses_2_layer.csv', index=False)
print('Training losses saved to train_losses_2_layer.csv')
```

```
Training losses saved to train_losses_2_layer.csv
```

1.10 11. Evaluate on test dataset and save submission

[28]:
```
test_dataset = GalaxyZooImageDataset(
    csv_file=None,
    image_dir='/kaggle/working/galaxy_zoo/images_test_rev1',
    transform=train_transform
)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False,␣
 ↪num_workers=4, pin_memory=True)

convnext.eval()
all_predictions = []
all_galaxy_ids = []

with torch.no_grad():
    for images, galaxy_ids in test_loader:
        images = images.to(device, non_blocking=True)
        outputs = convnext(images)
        probs = torch.sigmoid(outputs)
        all_predictions.append(probs.cpu().numpy())
        all_galaxy_ids.extend(galaxy_ids)

predictions = np.concatenate(all_predictions, axis=0)

columns = ['GalaxyId']
questions = {1: 3, 2: 2, 3: 2, 4: 2, 5: 4, 6: 2, 7: 3, 8: 7, 9: 3, 10: 3, 11: 6}
for q, count in questions.items():
    for i in range(1, count + 1):
        columns.append(f'Class{q}.{i}')

submission_df = pd.DataFrame(predictions, columns=columns[1:])
submission_df.insert(0, 'GalaxyId', all_galaxy_ids)
print(submission_df.head())
submission_df.to_csv('submission_2_layer.csv', index=False)
```

```
  GalaxyId  Class1.1  Class1.2  Class1.3  Class2.1  Class2.2  Class3.1  \
0   330879  0.813227  0.131761  0.049507  0.007043  0.120720  0.005384
1   488101  0.068476  0.926639  0.006073  0.043621  0.892778  0.059002
2   250283  0.416989  0.568959  0.006695  0.540014  0.097956  0.026330
3   674086  0.687720  0.266575  0.054557  0.008842  0.232478  0.013878
4   978686  0.523805  0.434150  0.035505  0.012733  0.379234  0.073247
```

```
   Class3.2  Class4.1  Class4.2  …  Class9.3  Class10.1  Class10.2  \
0  0.115147  0.006579  0.118242  …  0.000440   0.002269   0.002022
1  0.848857  0.847676  0.066704  …  0.016094   0.640775   0.301475
2  0.073546  0.020657  0.091587  …  0.045186   0.010986   0.004896
3  0.299039  0.032044  0.254406  …  0.000747   0.022740   0.011810
4  0.257033  0.036286  0.373198  …  0.001039   0.015428   0.016188

   Class10.3  Class11.1  Class11.2  Class11.3  Class11.4  Class11.5  Class11.6
0   0.001970   0.000581   0.001117   0.000121   0.000090   0.000548   0.004091
1   0.030069   0.018315   0.166348   0.114499   0.064391   0.131802   0.429292
2   0.005475   0.000820   0.006657   0.000172   0.000181   0.000552   0.013598
3   0.004322   0.008367   0.005242   0.000989   0.000310   0.001793   0.035874
4   0.006899   0.007676   0.013576   0.000762   0.000359   0.000890   0.016691

[5 rows x 38 columns]
```