

convnext-curriculum

February 27, 2026

1 Galaxy Zoo 2: ConvNeXt Curriculum Learning

This notebook demonstrates curriculum learning for the Galaxy Zoo 2 dataset using PyTorch and ConvNeXt. We reuse functions and classes from the original ConvNeXt notebook and implement a staged training process where the model is first trained on high-confidence examples, then gradually exposed to more ambiguous data.

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.

```
[1]: 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
```

1.2 2. Define Image Preprocessing and Tensor Saving Functions

Reuse the image preprocessing and tensor saving functions from the original notebook.

```
[2]: norm = transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.
    ↪225])
def process_image(img_path_output_size):
    img_path, output_dir, size = img_path_output_size
    basename = os.path.splitext(os.path.basename(img_path))[0]
    out_path = os.path.join(output_dir, f"{basename}.pt")
```

```

    img = Image.open(img_path).convert('RGB')
    width, height = img.size
    left, top, right, bottom = 20, 20, width - 20, height - 20
    img_cropped = img.crop((left, top, right, bottom))
    img_resized = img_cropped.resize(size, Image.LANCZOS)
    tensor = transforms.ToTensor()(img_resized)
    tensor = norm(tensor)
    torch.save(tensor, out_path)

def save_tensor_images_threaded(input_dir, output_dir, size=(224, 224), ↴
    num_workers=4):
    os.makedirs(output_dir, exist_ok=True)
    image_files = glob.glob(os.path.join(input_dir, '*.jpg'))
    print(f"Found {len(image_files)} images.")
    args = [(img_path, output_dir, size) for img_path in image_files]
    with concurrent.futures.ThreadPoolExecutor(max_workers=num_workers) as ↴
        executor:
        results = list(executor.map(process_image, args))
    for res in results[:20]:
        print(res)
    print(f"Finished saving tensors for {len(image_files)} images.")

```

```

[3]: 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)

```

```

[3]: 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]
    )

```

])

1.3 3. Define GalaxyZooTensorDataset Class

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

```
[4]: 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
```

1.4 4. Prepare Curriculum Learning Data Splits

We will split the training data into curriculum stages based on label confidence (maximum probability per sample). High-confidence samples will be used first, followed by medium and low-confidence

samples.

```
[5]: # 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)

# Compute confidence (max probability) for each sample
confidences = dataset.labels.max(axis=1)

# Define thresholds for curriculum stages
high_thresh = 0.94
med_thresh = 0.85

high_conf_idx = np.where(confidences >= high_thresh)[0]
med_conf_idx = np.where((confidences < high_thresh) & (confidences >= med_thresh))[0]
low_conf_idx = np.where(confidences < med_thresh)[0]

print(f"High confidence: {len(high_conf_idx)} samples")
print(f"Medium confidence: {len(med_conf_idx)} samples")
print(f"Low confidence: {len(low_conf_idx)} samples")
```

```
High confidence: 20107 samples
Medium confidence: 21556 samples
Low confidence: 19915 samples
```

1.5 5. Create DataLoaders for Curriculum Stages

Create DataLoaders for each curriculum stage using the corresponding subset of the training data.

```
[6]: batch_size = 512

high_conf_loader = DataLoader(Subset(dataset, high_conf_idx), batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True)
med_conf_loader = DataLoader(Subset(dataset, np.concatenate([high_conf_idx, med_conf_idx])), batch_size=batch_size, shuffle=True, num_workers=0, pin_memory=True)
full_loader = DataLoader(dataset, batch_size=batch_size, shuffle=True, num_workers=4, pin_memory=True)

# For validation, split a portion from the full dataset
from torch.utils.data import random_split
total = len(dataset)
val_size = int(0.2 * total)
train_size = total - val_size
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])
```

```
val_loader = DataLoader(val_dataset, batch_size=batch_size, shuffle=False, num_workers=4, pin_memory=True)
```

1.6 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.

```
[ ]: from torchvision.models import convnext_small, ConvNeXt_Small_Weights
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)
```

1.7 7. Set Up Loss Function and Optimizer

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

```
[ ]: criterion = nn.BCEWithLogitsLoss()

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

```
    weight_decay=1e-4,  
)
```

1.8 8. Train Model with Curriculum Learning Loop

Train the model sequentially on high-confidence, then medium, then all data. Track and print training loss for each stage.

```
[8]: def train_one_stage(model, loader, optimizer, criterion, device, num_epochs=3, ↵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
```

1.9 9. Save Model Checkpoints

```
[9]: 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}")
```

```
[10]: train_losses = []
```

```
[13]: print("Training on high-confidence samples...")
stage_name = "High_Confidence_2_layers"
high_losses = train_one_stage(convnext, high_conf_loader, optimizer, criterion,
                             device, num_epochs=5, stage_name=stage_name)
save_checkpoint(convnext, optimizer, train_losses,
               f"curriculum_checkpoint_{stage_name}.pth")
train_losses.extend(high_losses)
```

Training on high-confidence samples...

```
High_Confidence_2_layers Epoch 1/5 Batch 30/40 Loss=0.2506 RunningAvg=0.3088
High_Confidence_2_layers Epoch 1/5: Loss=0.2934, Time=177.72s
High_Confidence_2_layers Epoch 2/5 Batch 30/40 Loss=0.2336 RunningAvg=0.2334
High_Confidence_2_layers Epoch 2/5: Loss=0.2325, Time=174.37s
High_Confidence_2_layers Epoch 3/5 Batch 30/40 Loss=0.2227 RunningAvg=0.2241
High_Confidence_2_layers Epoch 3/5: Loss=0.2237, Time=175.42s
High_Confidence_2_layers Epoch 4/5 Batch 30/40 Loss=0.2258 RunningAvg=0.2205
High_Confidence_2_layers Epoch 4/5: Loss=0.2201, Time=174.97s
High_Confidence_2_layers Epoch 5/5 Batch 30/40 Loss=0.2152 RunningAvg=0.2179
High_Confidence_2_layers Epoch 5/5: Loss=0.2171, Time=175.49s
Checkpoint saved as curriculum_checkpoint_High_Confidence_2_layers.pth
```

```
[14]: print("Training on high+medium-confidence samples...")
stage_name = "Medium_Confidence_2_layers"
med_losses = train_one_stage(convnext, med_conf_loader, optimizer, criterion,
                            device, num_epochs=5, stage_name=stage_name)
save_checkpoint(convnext, optimizer, train_losses,
               f"curriculum_checkpoint_{stage_name}.pth")
train_losses.extend(med_losses)
```

Training on high+medium-confidence samples...

```
Medium_Confidence_2_layers Epoch 1/5 Batch 30/82 Loss=0.2317 RunningAvg=0.2342
```

```

Medium_Confidence_2_layers Epoch 1/5 Batch 60/82 Loss=0.2283 RunningAvg=0.2342
Medium_Confidence_2_layers Epoch 1/5: Loss=0.2340, Time=364.33s
Medium_Confidence_2_layers Epoch 2/5 Batch 30/82 Loss=0.2293 RunningAvg=0.2323
Medium_Confidence_2_layers Epoch 2/5 Batch 60/82 Loss=0.2304 RunningAvg=0.2324
Medium_Confidence_2_layers Epoch 2/5: Loss=0.2322, Time=363.03s
Medium_Confidence_2_layers Epoch 3/5 Batch 30/82 Loss=0.2351 RunningAvg=0.2319
Medium_Confidence_2_layers Epoch 3/5 Batch 60/82 Loss=0.2302 RunningAvg=0.2308
Medium_Confidence_2_layers Epoch 3/5: Loss=0.2307, Time=363.43s
Medium_Confidence_2_layers Epoch 4/5 Batch 30/82 Loss=0.2266 RunningAvg=0.2296
Medium_Confidence_2_layers Epoch 4/5 Batch 60/82 Loss=0.2360 RunningAvg=0.2293
Medium_Confidence_2_layers Epoch 4/5: Loss=0.2293, Time=363.34s
Medium_Confidence_2_layers Epoch 5/5 Batch 30/82 Loss=0.2291 RunningAvg=0.2287
Medium_Confidence_2_layers Epoch 5/5 Batch 60/82 Loss=0.2232 RunningAvg=0.2286
Medium_Confidence_2_layers Epoch 5/5: Loss=0.2287, Time=363.30s
Checkpoint saved as curriculum_checkpoint_Medium_Confidence_2_layers.pth

```

```
[15]: print("Training on all samples...")
stage_name = "All_data_2_layers"
full_losses = train_one_stage(convnext, full_loader, optimizer, criterion, ▾
    device, num_epochs=10, stage_name=stage_name)
save_checkpoint(convnext, optimizer, train_losses, ▾
    f"curriculum_checkpoint_{stage_name}.pth")
train_losses.extend(full_losses)
```

Training on all samples..

```

All_data_2_layers Epoch 1/10 Batch 30/121 Loss=0.2431 RunningAvg=0.2470
All_data_2_layers Epoch 1/10 Batch 60/121 Loss=0.2483 RunningAvg=0.2467
All_data_2_layers Epoch 1/10 Batch 90/121 Loss=0.2468 RunningAvg=0.2465
All_data_2_layers Epoch 1/10 Batch 120/121 Loss=0.2513 RunningAvg=0.2465
All_data_2_layers Epoch 1/10: Loss=0.2465, Time=343.20s
All_data_2_layers Epoch 2/10 Batch 30/121 Loss=0.2460 RunningAvg=0.2454
All_data_2_layers Epoch 2/10 Batch 60/121 Loss=0.2397 RunningAvg=0.2453
All_data_2_layers Epoch 2/10 Batch 90/121 Loss=0.2391 RunningAvg=0.2454
All_data_2_layers Epoch 2/10 Batch 120/121 Loss=0.2490 RunningAvg=0.2455
All_data_2_layers Epoch 2/10: Loss=0.2455, Time=340.73s
All_data_2_layers Epoch 3/10 Batch 30/121 Loss=0.2490 RunningAvg=0.2443
All_data_2_layers Epoch 3/10 Batch 60/121 Loss=0.2450 RunningAvg=0.2449
All_data_2_layers Epoch 3/10 Batch 90/121 Loss=0.2416 RunningAvg=0.2451
All_data_2_layers Epoch 3/10 Batch 120/121 Loss=0.2463 RunningAvg=0.2448
All_data_2_layers Epoch 3/10: Loss=0.2448, Time=340.18s
All_data_2_layers Epoch 4/10 Batch 30/121 Loss=0.2415 RunningAvg=0.2437
All_data_2_layers Epoch 4/10 Batch 60/121 Loss=0.2462 RunningAvg=0.2441
All_data_2_layers Epoch 4/10 Batch 90/121 Loss=0.2468 RunningAvg=0.2441
All_data_2_layers Epoch 4/10 Batch 120/121 Loss=0.2452 RunningAvg=0.2441
All_data_2_layers Epoch 4/10: Loss=0.2442, Time=340.82s
All_data_2_layers Epoch 5/10 Batch 30/121 Loss=0.2528 RunningAvg=0.2438
All_data_2_layers Epoch 5/10 Batch 60/121 Loss=0.2472 RunningAvg=0.2437
All_data_2_layers Epoch 5/10 Batch 90/121 Loss=0.2426 RunningAvg=0.2436

```

```

All_data_2_layers Epoch 5/10 Batch 120/121 Loss=0.2394 RunningAvg=0.2437
All_data_2_layers Epoch 5/10: Loss=0.2437, Time=340.98s
All_data_2_layers Epoch 6/10 Batch 30/121 Loss=0.2452 RunningAvg=0.2444
All_data_2_layers Epoch 6/10 Batch 60/121 Loss=0.2438 RunningAvg=0.2439
All_data_2_layers Epoch 6/10 Batch 90/121 Loss=0.2418 RunningAvg=0.2435
All_data_2_layers Epoch 6/10 Batch 120/121 Loss=0.2432 RunningAvg=0.2433
All_data_2_layers Epoch 6/10: Loss=0.2433, Time=340.80s
All_data_2_layers Epoch 7/10 Batch 30/121 Loss=0.2443 RunningAvg=0.2427
All_data_2_layers Epoch 7/10 Batch 60/121 Loss=0.2388 RunningAvg=0.2430
All_data_2_layers Epoch 7/10 Batch 90/121 Loss=0.2436 RunningAvg=0.2428
All_data_2_layers Epoch 7/10 Batch 120/121 Loss=0.2414 RunningAvg=0.2429
All_data_2_layers Epoch 7/10: Loss=0.2429, Time=341.24s
All_data_2_layers Epoch 8/10 Batch 30/121 Loss=0.2460 RunningAvg=0.2425
All_data_2_layers Epoch 8/10 Batch 60/121 Loss=0.2409 RunningAvg=0.2426
All_data_2_layers Epoch 8/10 Batch 90/121 Loss=0.2447 RunningAvg=0.2425
All_data_2_layers Epoch 8/10 Batch 120/121 Loss=0.2372 RunningAvg=0.2424
All_data_2_layers Epoch 8/10: Loss=0.2424, Time=341.25s
All_data_2_layers Epoch 9/10 Batch 30/121 Loss=0.2405 RunningAvg=0.2413
All_data_2_layers Epoch 9/10 Batch 60/121 Loss=0.2434 RunningAvg=0.2418
All_data_2_layers Epoch 9/10 Batch 90/121 Loss=0.2426 RunningAvg=0.2421
All_data_2_layers Epoch 9/10 Batch 120/121 Loss=0.2460 RunningAvg=0.2421
All_data_2_layers Epoch 9/10: Loss=0.2421, Time=340.51s
All_data_2_layers Epoch 10/10 Batch 30/121 Loss=0.2435 RunningAvg=0.2425
All_data_2_layers Epoch 10/10 Batch 60/121 Loss=0.2436 RunningAvg=0.2423
All_data_2_layers Epoch 10/10 Batch 90/121 Loss=0.2402 RunningAvg=0.2416
All_data_2_layers Epoch 10/10 Batch 120/121 Loss=0.2387 RunningAvg=0.2417
All_data_2_layers Epoch 10/10: Loss=0.2417, Time=340.38s
Checkpoint saved as curriculum_checkpoint_All_data_2_layers.pth

```

1.10 10. Save Training Loss for Each Curriculum Stage

```
[16]: loss_df = pd.DataFrame({'epoch': list(range(1, len(train_losses)+1)), 'loss': train_losses})
loss_df.to_csv('curriculum_train_losses_2_layers.csv', index=False)
print('Training losses saved to curriculum_train_losses_2_layers.csv')
```

Training losses saved to curriculum_train_losses_2_layers.csv

1.11 11. Evaluate on test dataset and save submission

```
[ ]: 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_curriculum_2_layer.csv', index=False)

```