1. **Use iNaturalist + LichenPortal to build your own dataset**, focusing on your region (Texas) or genus (e.g., *Cladonia*, *Parmotrema*).

2. Clean it up by cross-checking expert-verified IDs.

3. Organize it into a public dataset (hosted on GitHub or Hugging Face Datasets).

4. Train and publish a PyTorch model or Gradio app for ID.

5. Share it—others would definitely be interested in contributing or using it.

Absolutely! Here's a step-by-step outline to help you get started using data from **LichenPortal** and **iNaturalist**, and training a model in **PyTorch** for lichen recognition:

**🔍 Step 1: Collect and Organize Image Data**

**From LichenPortal**

* Navigate to [LichenPortal](https://lichenportal.org/portal/collections/misc/collprofiles.php).
* Use the **Search Collections** feature to filter for specimens with images.
* You can often bulk download metadata with images. Look for an "Export CSV" or "Download Images" option (some images might be hosted externally).
* Note the taxonomic information (species/genus) and image URL.
  + Identifications.csv
    - coreid: specific to each file
    - scientificName: Species (Genus + epithet)
    - genus
    - specificEpithet
  + multimedia.csv
    - coreid
    - goodQualityAccessURI

**From iNaturalist**

* Use the iNaturalist API to download observations of lichens with photos and taxonomic labels.
  + Endpoint: https://api.inaturalist.org/v1/observations
  + Filter by taxon\_id (lichens), photos=true, and desired location/date parameters.
  + Download the image URLs and taxonomic labels.
* For bulk downloads, use [this iNaturalist tool](https://www.inaturalist.org/pages/developers) or the Python library pyinaturalist.

\*\*\*Issues with getting an API token

taxon\_id = 54743

quality\_grade:

location =

**🧹 Step 2: Prepare Your Dataset**

#### **From LichenPortal:**

You have two CSV files:

* taxa.csv: Includes species information (coreid, scientificName, etc.).
* media.csv: Includes image URLs and a coreid column linking to the taxa.

✅ **Join these two** on the coreid column to link image URLs to species names.

#### **From iNaturalist:**

* You already have scientific\_name and image\_url per row — no merge needed.

Create a root folder like:

/Users/eabowman/Dropbox/LichenProject/Dataset/

Inside it, for each species, create a subfolder:

Dataset/

├── Xanthoparmelia\_chlorochroa/

│ ├── img\_001.jpg

│ └── ...

├── Cladonia\_evansii/

│ └── ...

...

This is the format expected by most machine learning libraries (e.g. torchvision.datasets.ImageFolder).

Script to Download and organize images

import os

import pandas as pd

import requests

from tqdm import tqdm

# Set paths

output\_dir = "/Users/eabowman/Dropbox/LichenProject/Dataset"

os.makedirs(output\_dir, exist\_ok=True)

# --- LichenPortal ---

# Load and merge lichenportal data

taxa\_df = pd.read\_csv("taxa.csv") # path to your taxa CSV

media\_df = pd.read\_csv("media.csv") # path to your media CSV

lichen\_df = media\_df.merge(taxa\_df[['coreid', 'scientificName']], on='coreid')

# --- iNaturalist ---

inat\_df = pd.read\_csv("inaturalist\_data.csv") # your iNat CSV

# Combine both sources

combined = pd.DataFrame({

'scientific\_name': pd.concat([

lichen\_df['scientificName'],

inat\_df['scientific\_name']

], ignore\_index=True),

'image\_url': pd.concat([

lichen\_df['accessURI'], # or whatever column has the image URL

inat\_df['image\_url']

], ignore\_index=True)

})

# Clean species names for folder names

combined['clean\_name'] = combined['scientific\_name'].str.replace(r'\s+', '\_', regex=True)

# Download function

def download\_image(row):

folder = os.path.join(output\_dir, row['clean\_name'])

os.makedirs(folder, exist\_ok=True)

filename = os.path.join(folder, os.path.basename(row['image\_url']))

try:

response = requests.get(row['image\_url'], timeout=10)

response.raise\_for\_status()

with open(filename, 'wb') as f:

f.write(response.content)

except Exception as e:

print(f"Failed: {row['image\_url']} - {e}")

# Download images with progress bar

for \_, row in tqdm(combined.iterrows(), total=len(combined)):

download\_image(row)

**🧠 Step 3: Build a PyTorch Image Classifier**

Two options seem to be available for training the image classifier:

1. Simple CNN (Convolutional Neural Network) from scratch

#### Pros:

* Full control over architecture and training.
* Good for learning purposes.
* Lightweight and faster to train on small datasets.

#### Cons:

* Requires a large number of images per class to work well — which might be a problem if your dataset is imbalanced or sparse.
* Likely to underperform compared to transfer learning on complex image tasks like species ID.

#### Use this if:

You're experimenting or want to deeply understand CNNs and have a ton of labeled training images per species.

2. Transfer learning (e.g., ResNet, MobileNet, EfficientNet)

This involves using a model pre-trained on ImageNet (a huge dataset of millions of labeled images) and then fine-tuning it on your lichen dataset.

#### Pros:

* Leverages learned features from millions of images (shapes, textures, edges, etc.).
* Requires **less data per class** to achieve high accuracy.
* Much faster to train and often more accurate.
* Easy to extend for downstream tasks (like embeddings for clustering or biodiversity analysis).

#### Cons:

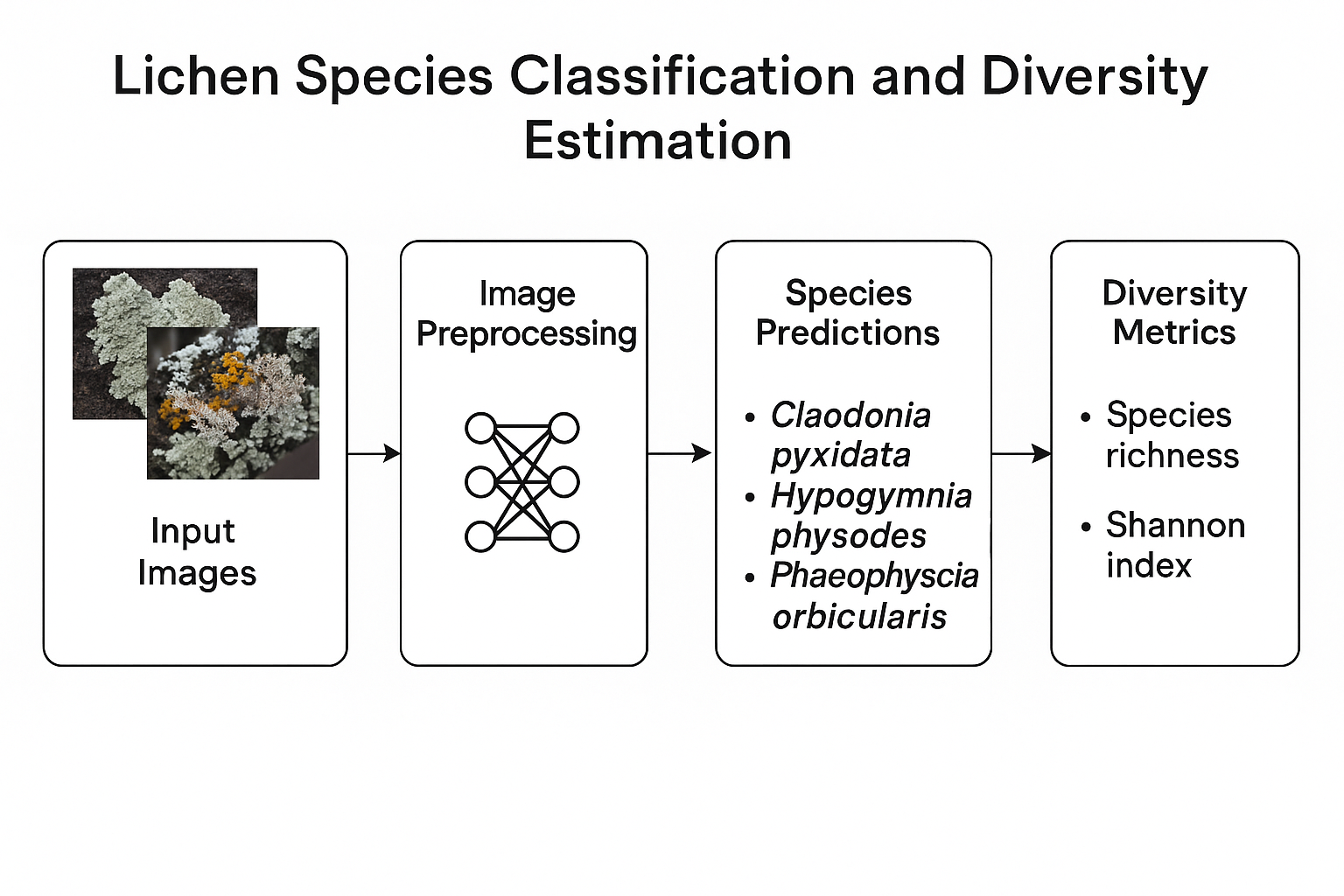
* Larger model size.
* Slightly more abstract (you don't design the architecture from scratch).

#### Use this if:

You want to get a robust lichen classifier quickly with high accuracy — **this is what I recommend for your project.**

Here’s how we’ll structure the pipeline:

1. **Train a transfer learning classifier** to predict lichen species from an image.
2. For each new photo, **predict species** for each image segment (or the image as a whole).
3. Tally species predictions per image to calculate:
   * **Species richness** = number of unique species.
   * **Diversity indices** = Shannon, Simpson, etc., based on prediction frequencies.



### Next steps (if you’re training a model):

1. **Define the model architecture**  
   Are you planning to use a pre-trained CNN like ResNet, or build your own?
2. **Set up loss function and optimizer**  
   Usually CrossEntropyLoss and Adam are a solid starting point.
3. **Train and validate the model**  
   Standard training loop with train() and eval() mode.
4. **Test and evaluate**  
   Accuracy, precision, recall — or whatever metrics suit your classification goal.
5. **Save the trained model**  
   So you can reload and use it later with torch.save() and torch.load().

### Where to find pre-trained models

1. **Torchvision**
   * Comes with popular models like resnet18, resnet50, densenet121, mobilenet\_v2, etc.
   * These are trained on ImageNet (1,000 general object classes).
   * Docs: https://pytorch.org/vision/stable/models.html
2. **Hugging Face**
   * Hosts lots of pre-trained vision models including ViT (Vision Transformer), ConvNeXt, Swin Transformer, etc.
   * Great for state-of-the-art models but a little heavier.
3. **TIMM (PyTorch Image Models)**
   * Created by Ross Wightman, a goldmine of pre-trained models.
   * You can install with pip install timm.
   * GitHub: <https://github.com/huggingface/pytorch-image-models>

### What to look for in a pre-trained model

If you're doing **transfer learning** (and you should! for now at least), here’s what you want:

| **Feature** | **Why it matters** |
| --- | --- |
| Trained on ImageNet | Good general-purpose features for natural images |
| Moderate size (e.g., ResNet18, MobileNetV2) | Avoid huge compute/memory requirements |
| Proven performance | Models like ResNet and DenseNet are well-tested |
| Customizable last layer | You’ll want to replace the final classification layer to output 1691 classes |

### Transfer learning vs. CNN from scratch

* **Transfer learning** is using a pre-trained CNN — so you're not doing both, you're doing one via the other.
* You load a pre-trained model, freeze some early layers (optional), and then fine-tune the rest to fit your lichen data.
* This is perfect for small- to medium-sized image datasets, especially with 1691 classes, which would be a heavy lift from scratch.

**📊 Step 4: Evaluate and Fine-Tune**

* Use a validation set to track performance during training.
* Use techniques like data augmentation, learning rate scheduling, and fine-tuning different layers.
* Save and load models using torch.save() and torch.load().

from torchvision import models

import torch.nn as nn

# Load pre-trained ResNet18

model = models.resnet18(pretrained=True)

# Freeze early layers (optional)

for param in model.parameters():

param.requires\_grad = False

# Replace final layer to match your number of classes

num\_features = model.fc.in\_features

model.fc = nn.Linear(num\_features, 1691) # Replace 1691 with your class count

# Now you can train this model

**What you're doing when you use resnet18(pretrained=True)**

* You're using a **Convolutional Neural Network (CNN)**: ResNet18 is a deep CNN architecture with convolutional layers, residual connections, etc.
* You're doing **transfer learning**: By using pretrained=True, you're starting with a model that already learned rich image features from **ImageNet**, and you're adapting (fine-tuning) it to your task.

**🔁 Step 5: Test and Deploy**

* Test your model on new field images.
* Consider deploying it as a web app using Gradio, Flask, or Streamlit for easy interaction.

Would you like a working Python script to get started with the API or training pipeline? Or maybe help building a starter dataset from iNaturalist?

Absolutely, I’d love to help you build this out! Here's a solid, practical **step-by-step plan** to guide your project—from raw lichen images to a shareable model and dataset that others could train or use. You’ll be laying groundwork for a valuable open resource.

## 🧭 Project Plan: Lichen Image Recognition Pipeline

### **🗂️ Phase 1: Dataset Construction**

#### ✅ 1. **Define Scope**

* Focus regionally (e.g., Texas lichens) or taxonomically (e.g., Cladonia, Usnea).
* Choose ~20–50 commonly observed species to start.
* Use iNaturalist and LichenPortal as primary sources.

#### ✅ 2. **Download Observations**

* Use the pyinaturalist API to download images + metadata (you've already started this!).
* Save image URLs, taxon names, observation dates, and locations.
* Limit to **research-grade** observations with **photos** and **community ID agreement**.

#### ✅ 3. **Filter and Clean**

* Remove duplicate or low-quality images.
* Manually verify a sample to catch misidentified images.
* Create a CSV like this:

| **species** | **image\_url** | **locality** |
| --- | --- | --- |
| Cladonia rangiferina | https://... | Travis County |
| Parmelia sulcata | https://... | Williamson Co. |

### **🧹 Phase 2: Dataset Preparation**

#### ✅ 4. **Download and Organize Images**

* Write a Python script to download and save images to species-labeled folders.
* Structure:
* data/
* ├── Cladonia\_rangiferina/
* │ ├── img1.jpg
* ├── Parmelia\_sulcata/
* │ ├── img1.jpg

#### ✅ 5. **Resize + Augment**

* Use torchvision.transforms to resize all images to a standard shape (e.g., 224×224).
* Optional: Apply light augmentations (rotation, brightness, zoom) to expand your training set.

### **🧠 Phase 3: Model Training**

#### ✅ 6. **Train Image Classifier**

* Use PyTorch + a pre-trained CNN (e.g., ResNet18 or EfficientNet).
* Load dataset with torchvision.datasets.ImageFolder.
* Fine-tune on your labeled images.
* Evaluate using a validation set (accuracy, confusion matrix, etc.).

#### ✅ 7. **Export Model**

* Save the trained model with .pt or .pth.
* Include metadata (class labels, training parameters).

### **🌐 Phase 4: Share + Deploy**

#### ✅ 8. **Host Dataset Publicly**

* Upload cleaned dataset to:
  + [Hugging Face Datasets](https://huggingface.co/datasets)
  + [Zenodo](https://zenodo.org/)
  + [GitHub](https://github.com/) (for metadata/scripts)

#### ✅ 9. **Deploy a Demo App**

* Use Gradio or Streamlit to create a simple web app:
  + Upload an image → get species prediction.
* Optional: Add a feedback loop for corrections (great for citizen science).

#### ✅ 10. **Publish + Invite Collaboration**

* Share your dataset and app with:
  + Fungal/lichen researchers
  + Biodiversity projects (e.g., iNaturalist forums)
  + Citizen science communities

## 🧰 Tools & Libraries You'll Use

| **Task** | **Tools** |
| --- | --- |
| Image collection | pyinaturalist, requests, pandas |
| Image processing | Pillow, torchvision |
| Model training | PyTorch, sklearn, matplotlib |
| App deployment | Gradio, Streamlit |
| Dataset hosting | GitHub, Zenodo, Hugging Face |

## 📝 Deliverables

1. Clean, labeled lichen image dataset.
2. Trained PyTorch model.
3. Jupyter notebook + scripts for training and downloading.
4. Optional: Web app for species prediction.
5. Optional: Dataset DOI if hosted on Zenodo.

Would you like me to start drafting one of the scripts for you—like image downloading or dataset prep? Or a Jupyter notebook template for training the model?

### **Purpose of** train\_model()

This function:

1. **Trains your model** over several epochs.
2. **Validates it** after each epoch to see how well it’s doing on unseen data.
3. **Keeps track of the best version** of your model (based on validation accuracy).
4. **Returns the trained model** with the best weights.

### 🧠 **What each part does**

#### since = time.time()

* Starts a timer to track how long training takes.

#### best\_model\_wts = copy.deepcopy(model.state\_dict())

* Saves a copy of the model’s starting weights.
* We'll replace this with the best weights we find.

#### for epoch in range(num\_epochs):

* Loops through the training process multiple times (each complete loop is one **epoch**).
* Each epoch helps the model learn a little more.

### 🔁 **For each epoch, we do two phases:**

* **train**: Learn from training data.
* **val**: Evaluate performance on validation data.

#### model.train() vs. model.eval()

* model.train(): Tells PyTorch to enable features like dropout and batch norm.
* model.eval(): Tells PyTorch to disable those features for evaluation.

### **Inside the training/validation loop:**

#### ```python

for inputs, labels in dataloaders[phase]:

- Goes through all batches of images and labels from your dataset.

#### ```python

inputs = inputs.to(device)

labels = labels.to(device)

Moves the data to the GPU (if you had one) or stays on CPU.

#### ```python

optimizer.zero\_grad()

- Clears old gradients before the new backward pass.

#### \*\*Forward pass:\*\*

```python

outputs = model(inputs)

\_, preds = torch.max(outputs, 1)

loss = criterion(outputs, labels)

 The model makes predictions.

 Calculates the loss (how wrong it was).

 Finds the class with the highest score (preds).

loss.backward()

optimizer.step()

 Tells the model how to improve based on the loss.

 Updates the model’s weights.

### **After each phase:**

* Calculates and prints:
  + Average loss (epoch\_loss)
  + Accuracy (epoch\_acc)

Best model tracking:

if phase == 'val' and epoch\_acc > best\_acc:

best\_acc = epoch\_acc

best\_model\_wts = copy.deepcopy(model.state\_dict())

* If this is the best accuracy so far on validation data, save it.

 Load the best weights back into the model.

 Return the trained model ready to use!