**An automated pipeline for supervised classification of petal color from citizen science photographs**

**The Problem:**

Researchers love studying flower color because it tells them a lot about ecology and evolution. But going out into the field to record this stuff? Pain. Massive pain. Why? Because flowers bloom quickly, the weather sucks, distances are huge, and money for fieldwork is tight. So how do you track flower color across big areas and times without losing your mind or your funding? That’s the pickle.

**What They Suggest (The Big Idea):**

Let’s get lazy-smart. Why not use the mountains of flower photos that regular folks (a.k.a. citizen scientists) are uploading on sites like iNaturalist? But wait, there's a twist. You can’t just scroll through 5000 blurry, shady, badly-lit photos and manually figure out what color the petals are. That’s madness. So, they built a **Python-based pipeline** that automatically looks at these images, finds the flower colors, and classifies them into neat categories. No human eyes needed.

**How They Did It (The Method):**

* **Pipeline Basics**: Four scripts written in Python, run in Jupyter Notebooks:
  1. **Downloader**: Grabs the images from iNaturalist.
  2. **Color Cluster Visualizer**: Figures out the major color blobs (clusters) in the flower parts using something called **k-means clustering** in **HSV color space** (think hue, saturation, value instead of basic RGB).
  3. **Data Collector**: Applies this clustering to ALL the images and collects stats on flower colors.
  4. **Classifier**: Sorts the images into user-defined color categories (like pink vs purple, or white vs blue).
* **Test Drive**: They tested this method on two plant species:
  1. **Geranium maculatum**: Flowers go from light pink to dark purple (continuous variation).
  2. **Linanthus parryae**: Flowers are either white or blue (binary – easy peasy).

**Results:**

* For **Geranium**, they processed 5000+ images. About 16% didn’t show flowers, and the rest got classified into Light, Medium, and Dark bins based on saturation (S) value.
* For **Linanthus**, out of 224 valid photos, the pipeline correctly picked out blue flowers with about **91.5% precision**.

**Conclusion (a.k.a. What’s the Big Deal?):**

* This method rocks for using messy, real-world photos from random people and turning them into useful, consistent data.
* It's simple, runs on normal laptops, and avoids all the complicated deep learning stuff that needs fancy GPUs and PhDs in machine learning.
* It’s flexible—you can tweak it for other flower species, or even other color-related research.
* But yeah, it's not perfect. It struggles if the flowers are tiny or if there’s other stuff in the photo that's the same color.

**A pipeline for the rapid collection of color data from photographs**

**The Problem:**

Scientists wanna know why flower colors vary across big landscapes – is it about climate, elevation, or just random? Problem is, getting this data traditionally takes a ton of time, money, and well, patience. Also, flower colors aren't just about looking pretty for pollinators; they might help with surviving droughts, UV rays, or fighting off pests. But most studies? They’re tiny in scope. To really crack this puzzle on a big scale, you need *lots* of flower pics.

**What They Suggest:**

Let’s crowdsource this. Use **iNaturalist** – you know, that place where plant nerds and nature lovers upload thousands of flower pics. But raw photos are chaotic. Lighting's different, angles vary, cameras suck sometimes. So, they built an **R Shiny app** (a user-friendly interface) to pull color data fast from these photos, especially focused on **Erysimum** (aka wallflowers, no, not the band). Then they looked at whether flower color had any real patterns tied to geography or climate.

**The Method:**

1. **Data Grab**: They snagged 4800+ wallflower pics from iNaturalist.
2. **The Tool**: R Shiny app where users can click on parts of each photo to grab color data, while keeping track of location/date.
3. **Color Science**: They focused on **hue** (color type) in the HSV color space, not just boring RGB.
4. **Spatial Analysis**:
   * Checked if flower colors are randomly spread or clustered (spoiler: they’re clustered).
   * Used stats magic (Moran’s I, PCA, regressions) to see if climate explains the color spread.
5. **Reality Check**: Compared raw pics to fancy calibrated ones (using ColorCheckers) to see if crappy iNaturalist photos are still useful (they are, mostly).

**Results (The Good Stuff):**

* **Colors aren’t random**: Wallflowers with yellow petals were everywhere, but orange ones? Mostly chillin' in specific spots like Arizona, New Mexico, and parts of California.
* **Climate Connection? Meh.**: There’s a weak link between color and climate. Spatial patterns were more about geography than temperature or rainfall.
* **Elevation? Now we’re talking.**: In some regions (like the Rockies), higher places had redder flowers. But this wasn’t true everywhere.
* **Raw Pics vs. Calibrated Pics**: Not much difference. Raw iNaturalist photos were “good enough” for big-picture trends.

**Conclusion (So What?):**

They built a fast, easy tool for anyone to scrape color data from photos. It works, and even though flower color isn’t super tied to climate across the whole range, there are juicy local patterns. Plus, they prove citizen science data, messy as it is, can still deliver solid insights – no need to be a data snob.

**GinJinn2: Object detection and segmentation for ecology and evolution**

**The Problem:**

In ecology and evolutionary research, dealing with image data is a slog. Think about trying to manually count seeds, bugs, or measure leaves in thousands of pics – yeah, it’s tedious and time-consuming. Machine learning, especially deep learning, could totally automate this, but it’s usually a techy nightmare for non-coders. Biologists aren’t trying to become data scientists overnight.

**What They Suggest:**

Enter **GinJinn2**, a super chill toolbox for **deep learning-based object detection and instance segmentation**. It’s built for biologists who don’t code but still want to flex with AI. This tool helps you spot, count, and segment stuff like seeds, insects, stomata, and leaves in images – all through a **command-line interface**, no need to touch hardcore code.

**How It Works (Methods):**

* **Backbone Tech**: It’s built on top of **Detectron2**, a serious deep learning framework.
* **Models**: Uses **Faster R-CNN** for bounding box detection and **Mask R-CNN** for pixel-perfect segmentation.
* **Dataset Handling**: Splits your data into training/validation/test sets, trains models, evaluates performance, and predicts on new images.
* **Pre/Post Processing**: You can crop, filter, merge datasets, and clean up image resolutions. Got small objects? Use sliding windows to zoom in on details.
* **No Coding Needed**: It’s all CLI-driven. You type commands, it does the AI magic.

**Example Uses (Real-World Flex):**

1. **Seed Counting**: They trained a model to detect seeds in dirty images (sand included). Nailed it with just **1% error**in seed proportions.
2. **Bug Detection**: On yellow sticky traps, detected bugs with around **7.2% error**. Decent, considering the crappy contrast and annotation quality.
3. **Stomata Segmentation**: Identified stomata (tiny leaf pores) with a **mean counting error of 2.34** per image. Not perfect, but workable.
4. **Leaf Extraction**: Pulled clean leaf outlines from herbarium specimens, ready for morphometric studies. Bounding box detection + segmentation = solid leaf masks.

**Results:**

* Seed counting? 🔥 98-99% accuracy.
* Insect traps? Solid but room for better data/annotations.
* Stomata? Could use more data, but it works.
* Leaves? Very usable, especially for detailed studies.

**Conclusion (The Big Takeaway):**

GinJinn2 makes deep learning accessible for non-techy scientists. It’s good for automating repetitive tasks with images – think counting, segmenting, detecting. It’s not perfect (only runs on Unix-like systems, needs an Nvidia GPU), but it’s powerful, flexible, and lets biologists focus on science, not software engineering.

**Image Harvest: an open-source platform for high-throughput plant image processing and analysis**

**The Problem:**

Plant scientists are drowning in data. Thanks to next-gen sequencing, we’ve got genotypes on lock, but phenotypes? Not so much. Measuring plant traits manually is slow, destructive, and expensive. Plus, plants change over time, so we need better, faster ways to collect and analyze images of plants to connect DNA to real-world traits (this is called the genotype-phenotype gap, and yeah, it’s a problem).

**What They Propose:**

They created **Image Harvest (IH)** – an **open-source, Python-based tool** for processing tons of plant images. It’s designed to run on powerful computing clusters (because your laptop isn’t cutting it), and it helps biologists extract useful data like plant size, shape, and growth traits from pictures. No heavy programming knowledge needed.

**Key Features:**

* Works with regular RGB images, fluorescence images, etc.
* Automates image analysis using **OpenCV** magic.
* Runs on your desktop for small jobs or on **Open Science Grid** for free supercomputing.
* Helps you measure things like **plant height, width, color**, and more from images.
* Extracts **digital traits** you can actually use in genetic studies (like GWAS).

**Methods:**

1. **Image Analysis Workflow**:
   * Remove background, clean up noise, crop plant from image.
   * Extract traits (e.g., projected shoot area, compactness, etc.).
   * Run this locally or on massive computing clusters for big datasets.
2. **Use Case – Rice Study**:
   * They imaged 376 rice genotypes, extracted traits using IH, and ran **genome-wide association studies (GWAS)**.
   * Found 3 major genetic regions linked to traits like plant density and growth habit.
   * IH’s results matched known biological pathways, proving it works.
3. **Performance**:
   * Processed 77,000+ images in ~30 hours on a cluster.
   * Tasks like trait extraction took the most time, but actual image processing was super fast.

**Results:**

* IH nailed the extraction of useful traits.
* Compared IH with other tools (PlantCV, LemnaGrid) – accuracy was on par, sometimes better.
* Found genes in rice related to important traits like growth habit and canopy density using IH-derived data.

**Conclusion:**

**Image Harvest** is a game-changer for plant biologists. It makes it possible to process huge image datasets and link plant traits to genetic data without needing a degree in computer science. Plus, it’s free, open-source, and runs on powerful computing grids if you don’t have fancy hardware.

**Step-by-Step Guide for Your Project (Let's Build This!):**

**STEP 1: Download + Filter Images**

* **Source**: GBIF (Global Biodiversity Information Facility)
* **Goal**: Get thousands of plant images, only keep ones with LEAVES.
* **Tasks**:
  1. Use GBIF API or scraping tool to grab images of specific species.
  2. Use **pre-trained deep learning model** (like **ResNet50**, **EfficientNetB0**) for:
     + **Binary Classification**: "Leaf" vs "No Leaf".
     + Dataset: You might need to **fine-tune** the model on some leaf/non-leaf images.
  3. Output: Clean folder with just **leaf images**.
* **Tools**:
  1. **Python**: Requests, BeautifulSoup for scraping (if needed).
  2. **PyTorch/TensorFlow**: For loading ResNet or EfficientNet.
  3. **Pre-trained models**: From torchvision or tf.keras.applications.

**STEP 2: Extract Features & Shapes**

* **2A: Deep Learning Feature Extraction**:
  + Use **ResNet/EfficientNet** to get **feature vectors** from the last layers (before classification).
  + Store these vectors (they represent the "learned" features of the leaves).
* **2B: Segmentation + Shape Extraction**:
  + **Segmentation**:
    - Use **Mask R-CNN** or **U-Net** to segment leaves from the background.
    - Goal: Get clean leaf **masks** (binary images where leaf = white, background = black).
    - Dataset: Try **LeafSnap** or build your own mini dataset to train.
  + **Shape Analysis**:
    - Apply **Fourier Descriptors** using **scikit-image** or **OpenCV**.
    - Other features: Aspect ratio, roundness, compactness.
* **Tools**:
  + **Detectron2**: For **Mask R-CNN**.
  + **Keras** or **PyTorch**: For **U-Net** (simpler if you want a lighter model).
  + **scikit-image / OpenCV**: For shape analysis.

**STEP 3: Evaluation + Statistical Analysis**

* **3A: Model Interpretation**:
  + Use **Grad-CAM** to see what parts of the image your deep learning model focused on.
  + Check if the model is truly "seeing" the leaf shape/morphology.
* **3B: Compare Deep vs Traditional Descriptors**:
  + Run **PCA** or **t-SNE** on deep features vs Fourier descriptors.
  + Evaluate which one clusters species better.
* **3C: Link to Environmental Data**:
  + Use GBIF’s metadata (location, climate zones).
  + Run **statistical tests** (like **ANOVA**, **correlation**) to see if leaf shapes vary by region/climate.
* **Tools**:
  + **Grad-CAM** libraries (e.g., tf-keras-vis or pytorch-grad-cam).
  + **scikit-learn**: PCA, t-SNE, clustering.
  + **Pandas + GeoPandas**: For environmental data.

**Algorithms/Functions Checklist:**

* **Image Filtering**: ResNet / EfficientNet (classification).
* **Segmentation**: Mask R-CNN / U-Net (leaf masks).
* **Feature Extraction**:
  + Deep: Feature vectors from ResNet/EfficientNet.
  + Traditional: Fourier Descriptors, Aspect Ratio, Roundness.
* **Visualization**: Grad-CAM.
* **Statistical Analysis**: Correlation, PCA, clustering.

**Questions to Ask Your Prof (Flex Those Brain Muscles):**

1. **Data Focus**: Which specific plant species are we targeting first?
2. **Model Training**: Do we have access to labeled datasets for leaf segmentation or do we need to create one?
3. **Computing Resources**: Do we have access to GPUs or high-throughput computing (e.g., cluster, cloud)?
4. **Environmental Metadata**: What’s the scope? Only geographical data, or specific climate variables too?
5. **Expected Output**: Are we focusing more on building a working pipeline, or on publishing insights from the leaf trait analysis?
6. **Deep Features vs Shape Descriptors**: Which one should we emphasize in our evaluation?
7. **Preferred Tools**: Any restrictions on tools? Should we stick to PyTorch, TensorFlow, or are both cool?

**Final Tips:**

* **Get clarity on the species + data volume** early.
* **Start small**: test with 100-200 images, build pipeline, then scale.
* Be ready to **fine-tune models** – GBIF images are messy.
* Document everything – these projects are heavy on steps.

Here’s a breakdown:

1. **Paper 1: APS3.11505** – **YES**
   * Relevant to **image filtering** and using citizen science images like GBIF.
   * Shows simple pipelines for color-based classification – useful when filtering images.
2. **Paper 2: APS3.11546** – **YES**
   * Relevant to **image processing** and **spatial/environmental analysis**.
   * Helps with **linking traits** (like leaf shape) to **biogeography**, similar to your final step.
3. **Paper 3: GinJinn2 (MEE)** – **YES**
   * Directly relevant for **leaf segmentation** using **Mask R-CNN**.
   * They did exactly what you need: **object detection + instance segmentation** for biological data.
4. **Paper 4: Image Harvest (JXB)** – **YES**
   * Relevant to **large-scale image processing pipelines**.
   * Helps with ideas on how to manage huge datasets efficiently.
5. **Frontiers Paper** – Haven’t read it here, but **likely YES** if it covers **deep learning in plant science**.