How to Deploy Your Trained Model

Başak Esin Köktürk-Güzel, PhD
Paul Bodesheim, PhD

CA22129 - InsectAI - Using Image-based AI for Insect Monitoring & Conservation (InsectAI)

September 29, 2025

Contents

1 Aim of This Documentation

The aim of this documentation is to provide a practical, step-by-step guide for deploying trained machine learning models on different platforms. Within the scope of the InsectAI project, our main goal is to make AI-based insect monitoring tools accessible not only to computer scientists but also to researchers from other disciplines, such as zoologists and ecologists.

Many experts in biodiversity and ecology can benefit from machine learning models, yet they often face difficulties in using technical deployment tools. This guide addresses that gap by showing how to deploy trained models on three widely used platforms:

- **Google Colab**: A free, cloud-based environment for quick prototyping, testing, and sharing of models without the need for local installation.
- **Hugging Face**: A model hub that allows publishing models with ready-to-use inference APIs, making them easily accessible for a broad community.
- **Google Cloud**: A scalable and production-ready solution for serving models to larger user bases with robust infrastructure.

Feature	Google Colab	Hugging Face Spaces	Google Cloud
Purpose	Experimentation, prototyping	Model sharing, demo	Production, scalable services
Ease of Use	Very easy	Easy	Moderate to difficult
Persistence	Temporary sessions	Persistent hosting	Persistent and reliable
Scalability	None	Limited	High (auto-scaling)
GPU/TPU	Free but limited	Pro plan only	Wide, pay-as-you-go
Cost	Free / Pro option	Free / Pro option	Pay-as-you-go
Best For	Education, testing	Showcasing models	Enterprise deployment

Table 1: Comparison of Google Colab, Hugging Face Spaces, and Google Cloud for deployment

By following this documentation, users (including zoologists, ecologists, and other non-computer scientists) will be able to deploy and interact with machine learning models more effectively, bridging the gap between AI research and practical applications in biodiversity monitoring.

2 Google Colab

Google Colaboratory (Colab) is a free, cloud-based environment provided by Google for running Python code in Jupyter Notebooks. It is widely adopted by the machine learning and data science community because:

- It provides free access to CPUs, GPUs, and TPUs without requiring a local installation.
- Notebooks can be shared easily, similar to Google Docs.
- Users can directly connect to their Google Drive to store and access files.

Colab is particularly valuable for researchers from non-computer science backgrounds, as it removes the need for complex local setup and allows experiments to be run directly from any web browser. In this project, we provide ready-to-use Colab notebooks for training and testing image classification models using standard backbones such as EfficientNetB0 [**?**], MobileNetV2 [**?**], ResNet50 [**?**], and InceptionV3 [**?**].

2.1 Activating Google Colab from Google Drive

If you have never used Colab before, it first needs to be activated in Google Drive:

1. Open Google Drive in your browser and log in with your Google account.
2. Click the **New** button on the left-hand side.

3. Navigate to **More** → **Google Colaboratory**.

4. If you do not see "Google Colaboratory" in the list:

   - Click on **Connect more apps**.
   - Search for "Colaboratory".
   - Install and connect it to your Google Drive.

5. After setup, new Colab notebooks can be created directly from Google Drive.

2.2 Provided Colab Notebooks

As part of our InsectAI COST Action Short-Term Scientific Mission (STSM), we prepared two Google Colab notebooks that allow users to train and test image classification models without writing any code. Both notebooks follow the same workflow and differ only in the chosen deep-learning framework.

2.2.1 Training your model

Two framework options are available: PyTorch and TensorFlow/Keras. Both versions perform the same tasks, so users can select whichever framework they prefer.

- **PyTorch**: hands-on, widely used in research; ideal for flexible experimentation.
- **TensorFlow/Keras**: production-oriented; convenient for exporting to TFLite/TF Serving.

The notebooks provide a no-code workflow where users only need to upload their dataset (organized in class-specific folders).

Key features:

- Choice of backbone models (EfficientNet, MobileNet, ResNet50, InceptionV3).
- Adjustable training parameters through simple selection menus.
- Automatic export of the trained model and a `class_names.txt` file required for testing.

A block diagram of the training workflow is shown in Figure 1. Each code cell in the notebook corresponds to a specific step, which we briefly explain below.

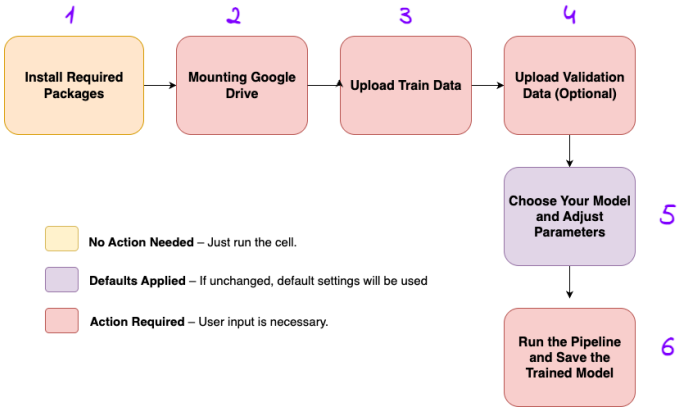


Figure 1: Block diagram of the Colab training workflow.

1. **Install Required Packages** – Installs the necessary Python libraries (e.g., TensorFlow, Keras, matplotlib). No user action is required, just run the cell.

2. **Mount Google Drive** – Connects Google Drive for accessing and storing datasets as well as experiment results. User input is required for authentication.

3. **Upload Train Data** – Upload the training dataset (images should be organized in separate folders according to their classes).

4. **Upload Validation Data (Optional)** – An additional dataset can be uploaded for model validation. This step is not mandatory.

5. **Choose Your Model and Adjust Parameters** – Select the backbone model and set parameters such as learning rate, batch size, and number of epochs. Default values are provided if not modified.

6. **Run the Pipeline and Save the Trained Model** – Executes the complete pipeline, trains the selected model, and saves the final trained model together with class labels for later use.

To run the notebook, open it in Colab and click **Runtime** → **Run All** to execute all cells sequentially, or run them individually using the **Run** button (triangle icon) shown in Figure 2.

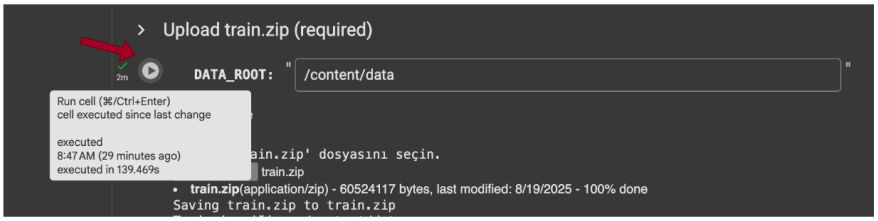


Figure 2: Running individual cells in Colab.

2.2.2 Testing your model

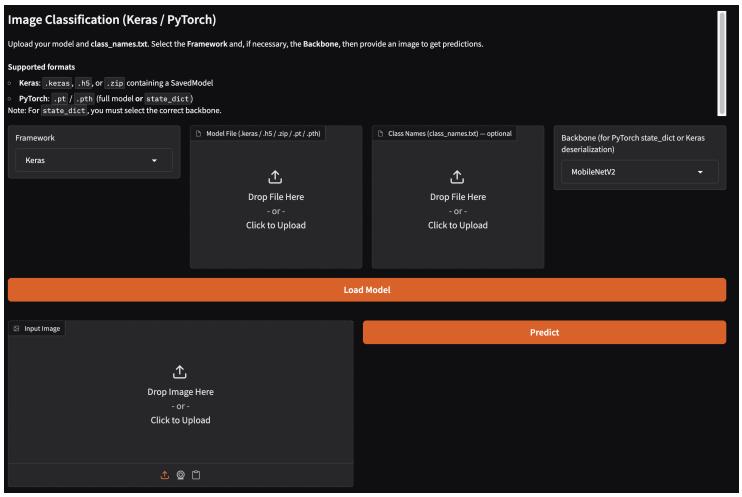
Similar to the training step, we now provide a **single Colab notebook** that supports both `PyTorch` and `TensorFlow/Keras` frameworks for model testing. The notebook uses `Gradio` to create an interactive web interface, enabling users to upload trained models and test them directly, without writing additional code. Simply open the notebook in Colab and run it to launch the interface.

Key features:

- Unified Gradio-based interface for uploading an image and receiving predictions.
- Easy loading of trained models (`.keras`, `.h5`, `SavedModel`, `.pt`, `.pth`) and the optional `class_names.txt` file.
- Automatic preprocessing for common backbones (EfficientNet, MobileNetV2, ResNet50, InceptionV3).
- Visualization of prediction results (top-5 predictions, confidence scores, or positive/negative outputs for binary models).

By using this unified notebook, researchers can train models on their own datasets and immediately test them in a browser-based interface, without installing any software locally.

At the end of execution, the notebook provides a clickable user interface (UI) link to launch the interactive testing environment. If you click on this link, you will see an interface similar to the example below. From here, you can easily select the framework, upload your model file and the optional `class_names.txt` file. Afterwards, simply upload a test image to obtain the prediction results.



For an overview of the entire workflow, an animated demonstration of training and testing on Colab is also available (note: the GIF requires a PDF reader such as Adobe Acrobat or Foxit Reader to play).

Training your model:

A GIF illustration of the training process will be placed here.

Testing your model:

A GIF illustration of the testing process will be placed here.

3 Hugging Face

Hugging Face is an open-source AI platform that provides tools, models, and datasets for natural language processing, computer vision, and other machine learning tasks. It is best known for the Transformers library, which offers state-of-the-art pretrained models that researchers and developers can easily adapt for their own projects. In addition, the Hugging Face Hub allows users to share, discover, and collaborate on models and datasets, while Gradio and Streamlit make it simple to deploy interactive machine learning applications through user-friendly frameworks.

In this tutorial, we will present two different paths to make your model available to users through Hugging Face.

- **Path 1:** Train your model in Google Colab (either with `model_train_with_keras.ipynb` or `model_train_with_pytorch.ipynb`) and deploy it on Hugging Face Spaces.
- **Path 2:** Use Hugging Face AutoTrain to train your model directly through a web interface and then deploy it on Spaces as well.

3.1 Path 1: Train in Colab and Deploy on Hugging Face Spaces

This path fine-tunes an image classifier on your own dataset and publishes it on the Hugging Face Hub. No deep coding is required; simply organize your images into folders named by class and run the provided Colab notebook.

3.1.1 Prepare Your Dataset (Folder Layout)

Organize your dataset as:

```
data/
  train/
    classA/  img001.jpg, img002.jpg, ...
    classB/  ...
  validation/
    classA/  ...
    classB/  ...
```

Tip. Folder names become class labels. Use `train`/`validation` (and optionally `test`).

3.1.2 Choose Your Training Notebook (Keras or PyTorch)

- `model_train_with_keras.ipynb` Trains with TensorFlow/Keras backbones (EfficientNetB0, MobileNetV2, ResNet50, InceptionV3). Exports to:
  - `model.keras` (recommended single-file format), or
  - `savedmodel/` directory (with `saved_model.pb` and `variables/`).

- `model_train_with_pytorch.ipynb` Trains with PyTorch backbones (ResNet50, MobileNetV2, EfficientNetB0, InceptionV3). Exports to:
  - `model.pt` or `model.pth` (full model via `torch.save(model, ...)`), or
  - `state_dict.pth` (weights only via `torch.save(model.state_dict(), ...)`).

3.1.3 Set Up via the Hugging Face Web Interface (No CLI)

Everything (account, repositories, access tokens) is managed from the web UI—no command-line tools are required.

Create an Account

1. Go to huggingface.co and click **Sign up**.
2. Verify your email address. After logging in, you will see your profile avatar in the top-right.

3.1.4 Prepare a Model Repository (on the Hub)

1. Click your avatar → **New model**.
2. Choose an **Owner** (your username or an organization).
3. Set a clear **Model name** (e.g., `insectai-image-classifier`).
4. Select a **License** (e.g., `apache-2.0`) and set **Visibility** (**Public** recommended).
5. Click **Create model**.

Upload trained model and files.

- For **Keras**: upload `model.keras` (or `savedmodel/` folder).
- For **PyTorch**: upload `model.pt` / `model.pth` (or `state_dict.pth`).
- Always include:
  - `class_names.txt` (one label per line, generated during training).
  - Optional `README.md` (short description, dataset, usage).

Notes

- The built-in *Inference Widget* may not auto-activate for plain Keras or PyTorch files. For a working demo, create a **Space (Gradio)** and load the model there.
- Keep `class_names.txt` in the same repo so that human-readable labels appear. Otherwise, predictions show as `class_0`, `class_1`, etc.

3.1.5 Enable the Inference Widget (Zero-Code Testing)

1. On your model page, open the **Model** tab.
2. If a supported file format is present, the **Inference widget** appears.
3. Upload a sample image and confirm top-k predictions.

3.1.6 Create a Web App on Spaces (All via UI)

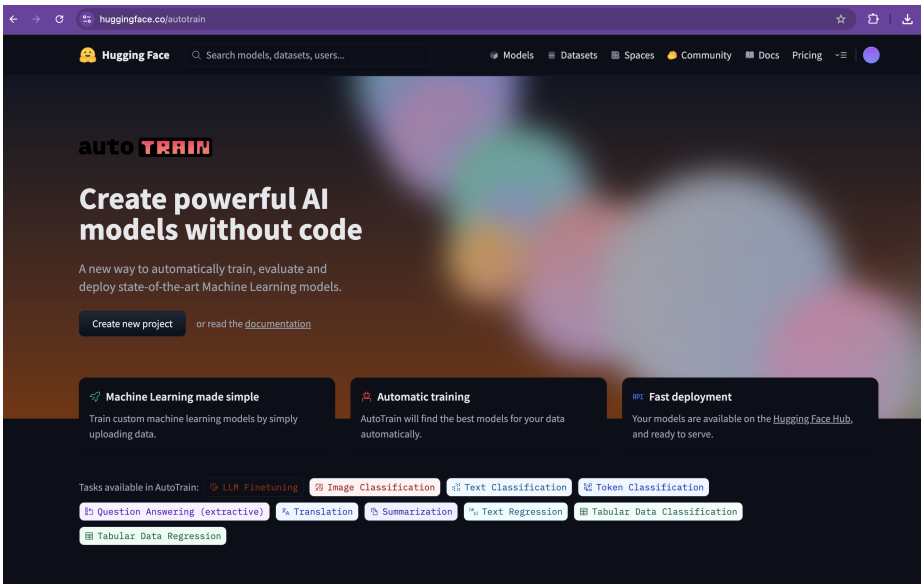
1. From the top bar, choose **Spaces** $\rightarrow$ **Create new Space**.
2. Set **SDK** = **Gradio**, pick a **Name**, set **Visibility**, then click **Create Space**.
3. Inside the Space, add `app.py` via **Files** $\rightarrow$ **Add file** $\rightarrow$ **Create new file**. Ready-to-use template files for Hugging Face Spaces are available for both frameworks: **Keras** (`app.py`,and **PyTorch** (app.py).
4. Add `requirements.txt` similarly. Again Ready-to-use template files for requirements.txt is available for Keras and PyTorch. The Space will build and launch automatically.
5. If your model repo is **Private**, add the Space as a **Collaborator** or make it **Public**.

3.2 Path 2: Use Hugging Face AutoTrain and Deploy on Spaces

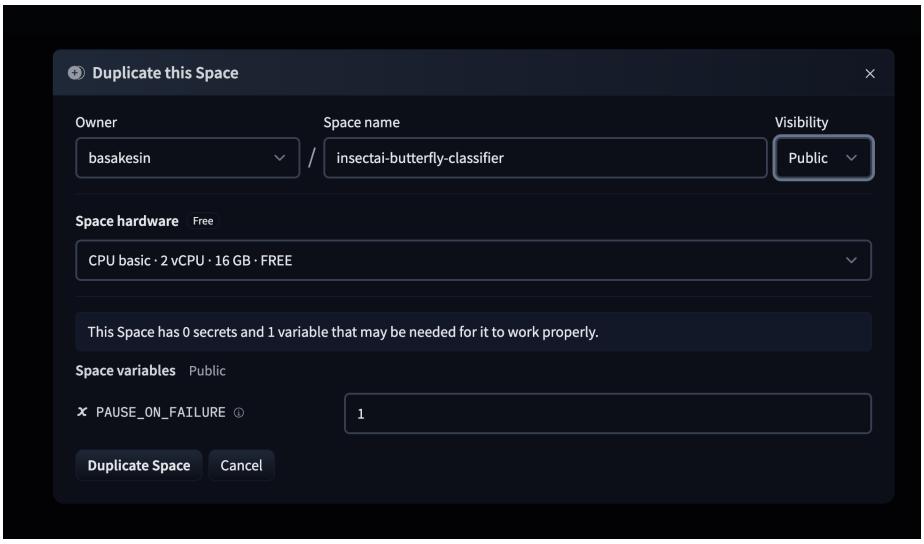
This alternative path uses Hugging Face AutoTrain to handle training and packaging automatically. It is simpler but gives less manual control.

3.2.1 AutoTrain (Web UI)

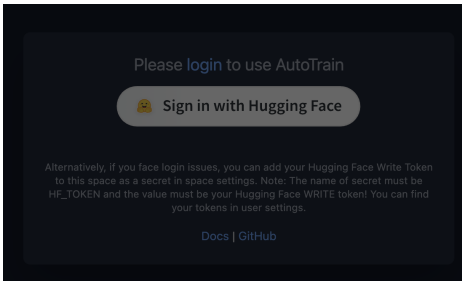
1. On the Hugging Face Hub, navigate to **AutoTrain** and click on **Create New Project**.



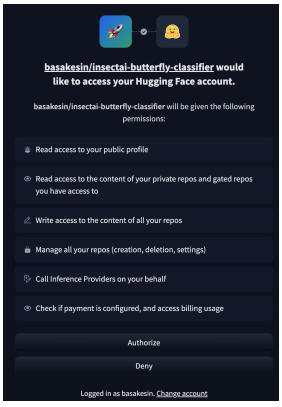
2. You are essentially duplicating the Autotrain Space. Please provide a clear and unique name for your Space, such as *insectai-butterfly-classifier*. If you set the visibility to **Private** by default, you must assign a role of **Collaborator** when deploying your model to Spaces. If there are no restrictions, we recommend setting it to **Public**.



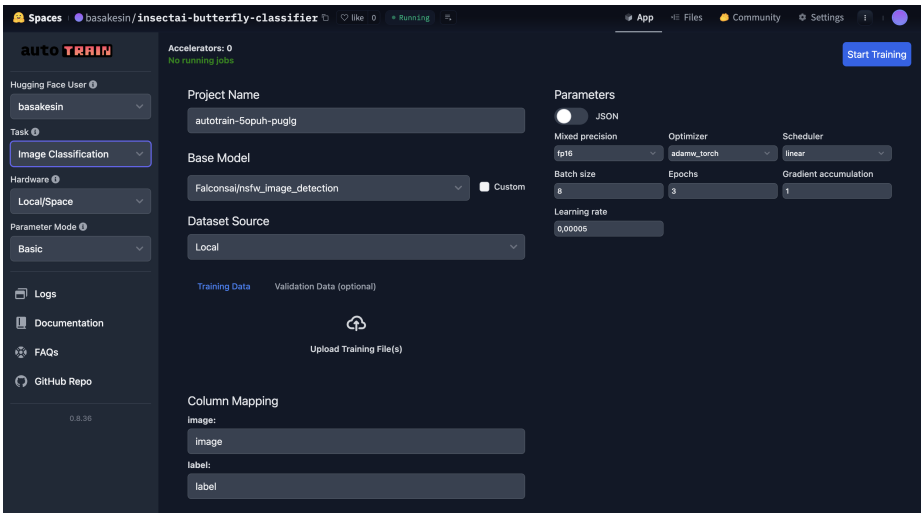
3. After duplicating the space, please login your Hugging Face profile.



4. Proceed by selecting **Authorize**. This step will grant the required permissions for the Space to access your Hugging Face account, including reading your profile, managing repositories, and running inference on your behalf.



5. Select the task: **Image Classification**.

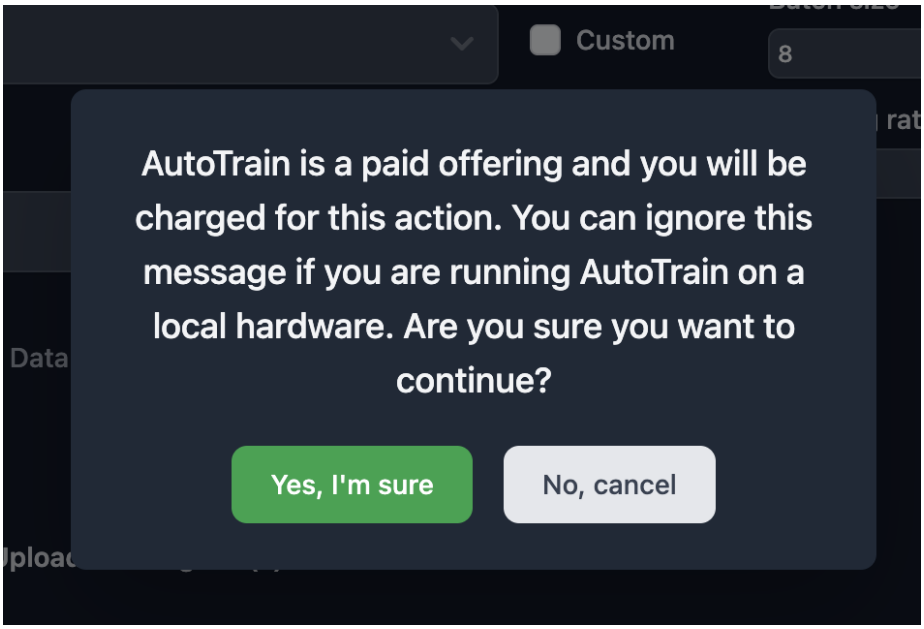


6. You can freely change the project name (e.g., in the example above it appears as `autotrain-5opuh-pglug`). This name will also become the model repository name on your Hugging Face profile. Next, choose a Base Model. If you are unsure which one to select, we recommend starting with `microsoft/resnet50` or `google/efficientnet-b0`.

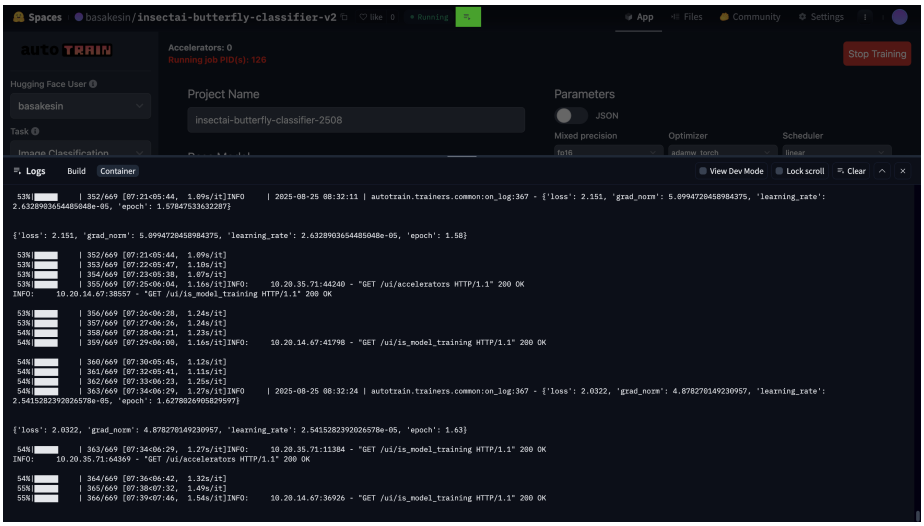
7. Upload your dataset (folder-per-class) or choose a Hub dataset. To upload your dataset zip the class folders inside train directly, not the train folder itself. When extracted, the archive should display classA/, classB/, etc., at the top level instead of train/classA/.

8. After that, you may adjust the parameters as needed, or simply leave them at their default values. Then, click the **Start Training** button.

9. A warning message will appear, indicating that **AutoTrain is a paid service** when used on Hugging Face's cloud infrastructure. If you are running AutoTrain on your own local hardware, you can safely ignore this message. Click **Yes, I'm sure** to proceed.



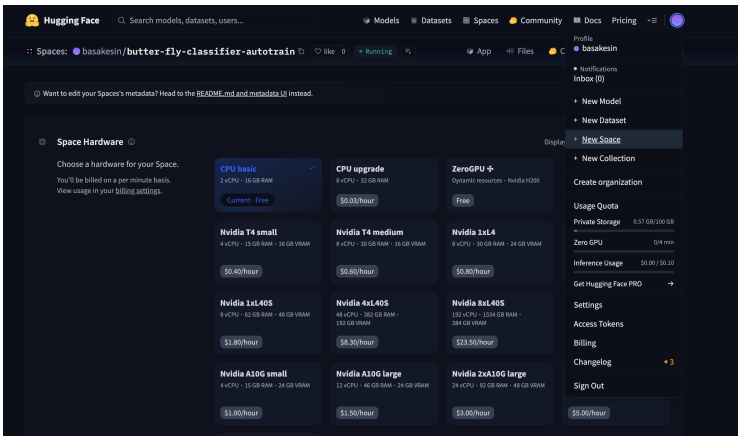
10. You can view all training progress and logs by clicking the log icon next to the `"Running"` status.



Deploying AutoTrain Models on Hugging Face Spaces

After completing training with **AutoTrain**, the resulting model is automatically pushed to your Hugging Face Hub account. To make it accessible through a user-friendly web interface, you can directly create a Space from this trained model:

1. On the Hugging Face Hub, you can create a new Space from your profile menu. Click your profile icon and select `+ New Space`.



2. Give your Space a descriptive name (e.g., `insectai-butterfly-classifier-demo`).
3. Select `Gradio` as the SDK.
4. Choose `Public` (recommended) or `Private` visibility.
5. After creating the Space, you will see the `Files` section at the top right. Click it, then select `+ Contribute` → `Create a new file`. Name the file `app.py` and copy-paste the example code provided below.

```python
import os
import gradio as gr
from transformers import pipeline

MODEL_ID = "basakesin/butterfly-classifier-auto-train"
HF_TOKEN = os.getenv("HF_TOKEN")  # if the model is private, the token comes from Space secrets

pipe = pipeline("image-classification", model=MODEL_ID, token=HF_TOKEN)

def predict(img, top_k=5):
    preds = pipe(img, top_k=top_k)  # [{'label': '...', 'score': 0.97}, ...]
    # Convert to dictionary for Gradio Label
    return {p["label"]: float(p["score"]) for p in preds}

demo = gr.Interface(
    fn=predict,
    inputs=[gr.Image(type="pil"), gr.Slider(1, 10, value=5, step=1, label="Top-K")],
    outputs=gr.Label(num_top_classes=5),  # by default shows top 5 predictions
    title="Butterfly Classifier",
    description="Upload an image to get top-k predictions."
)

if __name__ == "__main__":
    demo.launch()  # if share=True is set, it provides a public link
```

6. Change the `MODEL_ID` in the code to match your model path or the name of your AutoTrain project. If your model is private, you must also provide

the `HF_TOKEN`. To create an access token, click your profile icon on the Hugging Face Hub, navigate to `Access Tokens`, and then select `+ Create a new token`.

7. You can also modify the `title` and `description` fields in the code to customize the interface.

8. In the same way, create a file named `requirements.txt` to specify the necessary dependencies. Please copy and paste the following lines into the `requirements.txt` file:

```
gradio>=4.44.0
transformers>=4.41.0
torch
Pillow
safetensors
```

4 Google Cloud

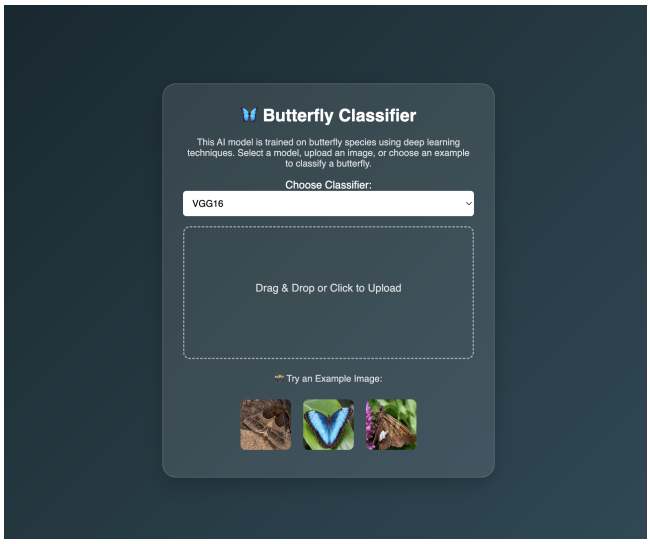
Deploying the trained model as a Flask application on Google Cloud provides several practical advantages compared to using Google Colab or Hugging Face. On Google Cloud, the system runs continuously without the session interruptions that are typical in Colab, and it can be scaled to serve multiple users in parallel, which is not possible with the limited resources of Hugging Face free tiers. The deployment also allows full customization of the interface and backend, as well as integration with other services such as databases and authentication systems, making it suitable for professional and industrial use. However, this approach requires additional coding and configuration effort, and it comes with financial costs since Google Cloud resources are billed, whereas Colab and Hugging Face offer limited free usage. Overall, while Colab and Hugging Face are suitable for rapid prototyping and demonstration, a Flask deployment on Google Cloud is more robust for continuous, large-scale, and secure applications.

4.1 Creating a Demo Web App –

There are many different ways to serve a trained model; however, for simplicity, we prepared a minimalist demo application for you. We also provide the complete source code of this demo, so that you can easily adapt it to your own project. By simply changing the model file, example images, and the project title, you can deploy your own application without additional effort.

We prepared a demo web page (last accessed on 01.09.2025) for a butterfly classifier using the Kaggle dataset[1]. The complete source code of this web application will be provided. You may freely modify the titles, images, models, and interface elements according to your requirements.

[1] `https://www.kaggle.com/datasets/gpiosenka/butterfly-images40-species`



4.1.1 File Structure of the InsectAI Flask Application

The project is organized into the following structure:

```
InsectAI Flask App/
|
|-- static/                          # Static resources (CSS, images, etc.)
|   `-- Example_Images/              # Example input images for testing
|       |-- demo_img1.jpg
|       |-- demo_img2.jpg
|       `-- demo_img3.jpg
|
|-- templates/                       # HTML templates for the web interface
|   `-- index.html                   # Main upload and results page
|
|-- uploads/                         # Folder for user-uploaded test images
|
|-- app.py                           # Main Flask application (routes and logic)
|
|-- model1.h5   # Pre-trained Keras model 1 ( e.g. InceptionV3)
|-- model2.h5   # Pre-trained Keras model 2 (e.g. MobileNetV2)
|-- model3.h5         # Pre-trained Keras model 3 (e.g. VGG16)
|
|-- Dockerfile                       # Instructions to build a Docker image
|-- requirements.txt                 # Python package dependencies
```

Explanation of files and folders:

- `static/`
  Contains static resources such as CSS or example images. The subfolder `Example_Images/` includes butterfly images used for demonstration. Users can replace or add their own sample images here.

- `templates/index.html`
  Defines the layout of the web interface. This is where the file upload form, prediction results, and page design are specified. Changing text or design elements can be done in this file.

- `uploads/`
  Temporary folder where images uploaded by users through the web interface are stored before being processed by the model.

- `app.py`
  The core of the Flask application. It loads one of the trained models, defines the prediction logic, and connects the backend with the web interface. To switch models, modify the loading path inside this file.

- `butterfly_classifier_InceptionV3.h5`, `butterfly_classifier_MobileNetV2.h5`, `butterfly_classifier_VGG16.h5`
  Pre-trained Keras models stored in HDF5 format. Replacing these files with new models (same input/output structure) allows the application to classify different datasets.

- `Dockerfile`
  Contains the build instructions for creating a Docker image of the application. It installs dependencies, copies project files, and starts the Flask server. By modifying this file, you can adapt the runtime environment (e.g., change Python version). **Note:** If you do not want to write the Dockerfile manually, you can generate a valid template by giving `app.py` to any Large Language Model (e.g., ChatGPT, Gemini, DeepSeek).

- `requirements.txt`
  Lists all required Python packages. When deploying, these packages are installed automatically. Update this file if you add new libraries to the project. **Note:** Similarly, the content of `requirements.txt` can be generated by providing `app.py` to an LLM (e.g., ChatGPT, Gemini, DeepSeek).

This structure makes it straightforward to test the demo, adapt it for other datasets, or deploy it to cloud services such as Google Cloud Run.

4.1.2 Deploying The Web App to Google Cloud

4.1.3 Deploying The Web App to Google Cloud

Before deploying, make sure you have:

- A Google Cloud account (Sign up here)
- A Google Cloud project (you can create one in the Google Cloud Console)
- Billing enabled on your project (required for Cloud Run, though there is a free tier)
- Google Cloud SDK (`gcloud`) installed on your computer (Install instructions)

Step 1: Authenticate Google Cloud

```
gcloud auth login
```

Step 2: Set your project

```
gcloud config set project PROJECT-ID
```

Step 3: Build the Docker image

```
gcloud builds submit --tag gcr.io/PROJECT-ID/butterfly-classifier
```

Step 4: Deploy to Cloud Run

```
gcloud run deploy butterfly-classifier \
  --image gcr.io/PROJECT-ID/butterfly-classifier \
  --platform managed \
  --region us-central1 \
  --allow-unauthenticated
```

After deployment, Cloud Run will print a service URL. Open this link in your browser to use the application. At this point, your model is fully deployed as a scalable web service accessible from anywhere.

Notes and Tips:

- Replace `PROJECT-ID` with the ID of your Google Cloud project.
- You can restrict access by removing the `--allow-unauthenticated` flag, which requires users to authenticate with Google accounts.
- By default, the free tier of Cloud Run includes up to 2 million requests per month, making it cost-effective for small demos.
- For production use, you may integrate the service with a database, enable logging and monitoring, or configure domain mapping with HTTPS.