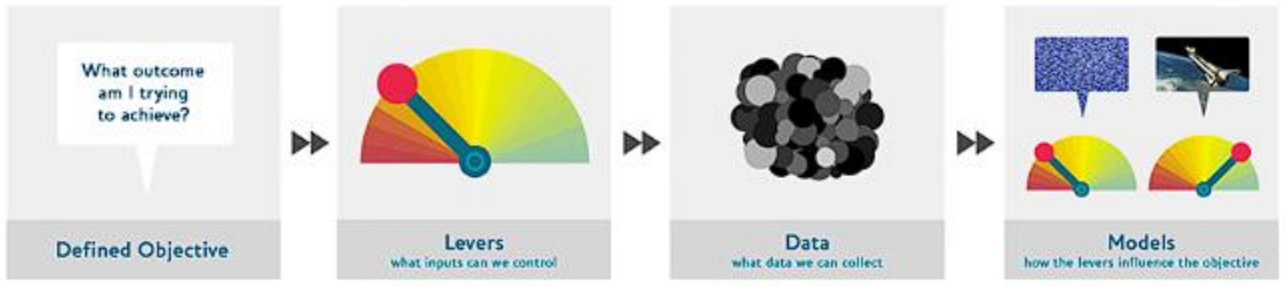
End to End deep learning project with Deployment

1. Think about a personal problem /project and work on it rather than waiting to study the theories
2. Iterate through the process to improve later. Always do this from start to end.
3. Your goals are what you obtain at the end of each iteration.
4. Its better to solve popular solvable problems rather than unique problems to ensure that we never get lost at point of failure.
5. Popular Deep learning domains:
   1. Computer Vision : Object Recognition
   2. Text NLP: DL can be used to translate text from one language to another, summarize long documents into something that can be digested more quickly, find all mentions of a concept of interest.
   3. Combine Text and Image
   4. Tabular data
   5. Recommendation system


What outcome am I trying to achieve?
Defined Objective
Levers
what inputs can we control
Data
what data we can collect
Models
how the levers influence the objective

The Drive Train Approach by Fastai

Project: ‘Bird or Not’ predictor

Objective:
Create a DL image classification model to identify whether the image uploaded is a bird or not and and deploy it online to be accessible to public.

What needs to be done:
1. We need a working model to classify a bird image.
2. A method to make the model accessible to general public (website).

Link to the model: Bird classifier model .pkl

Ml Production - a Hugging Face Space by mdkhalid

STEP 1: A working model

Kaggle notebook for creating a model (free GPU)

Substeps:

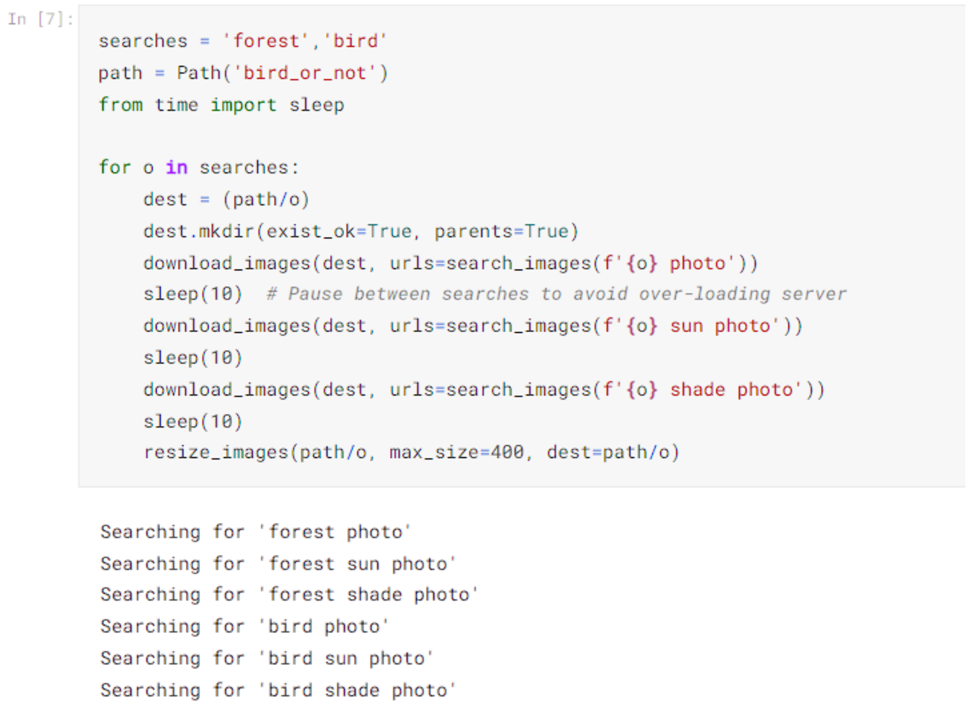
1. Use DuckDuckGo to search for images of bird and forest.
2. Fine-tune a pretrained neural network to recognize these two groups
3. Try running this model on a picture of a bird and see if it works.

Libraries: Fastai (high level Deep learning library build on top of Pytorch deep learning framework by Meta)

```
from duckduckgo_search import ddg_images
from fastcore.all import *
def search_images(term, max_images=30):
    print(f"Searching for '{term}'")
    return L(ddg_images(term, max_results=max_images)).itemgot('image')
```

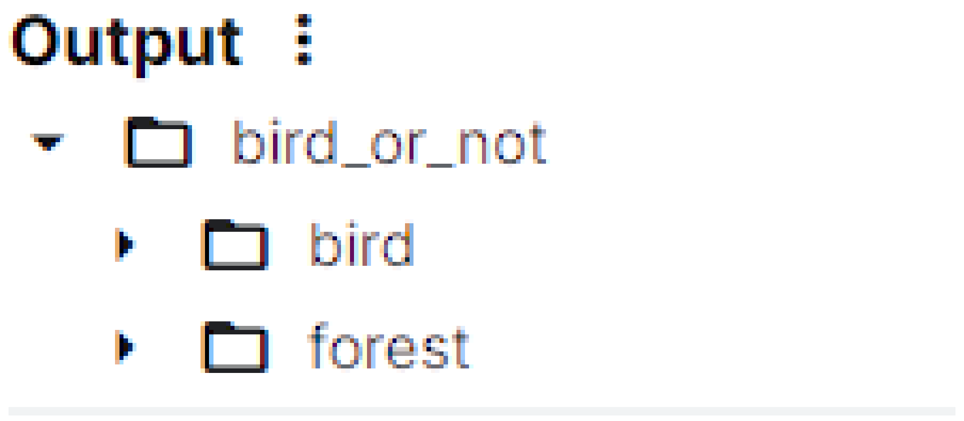
Data collection and storage

- Finding data and creating a database



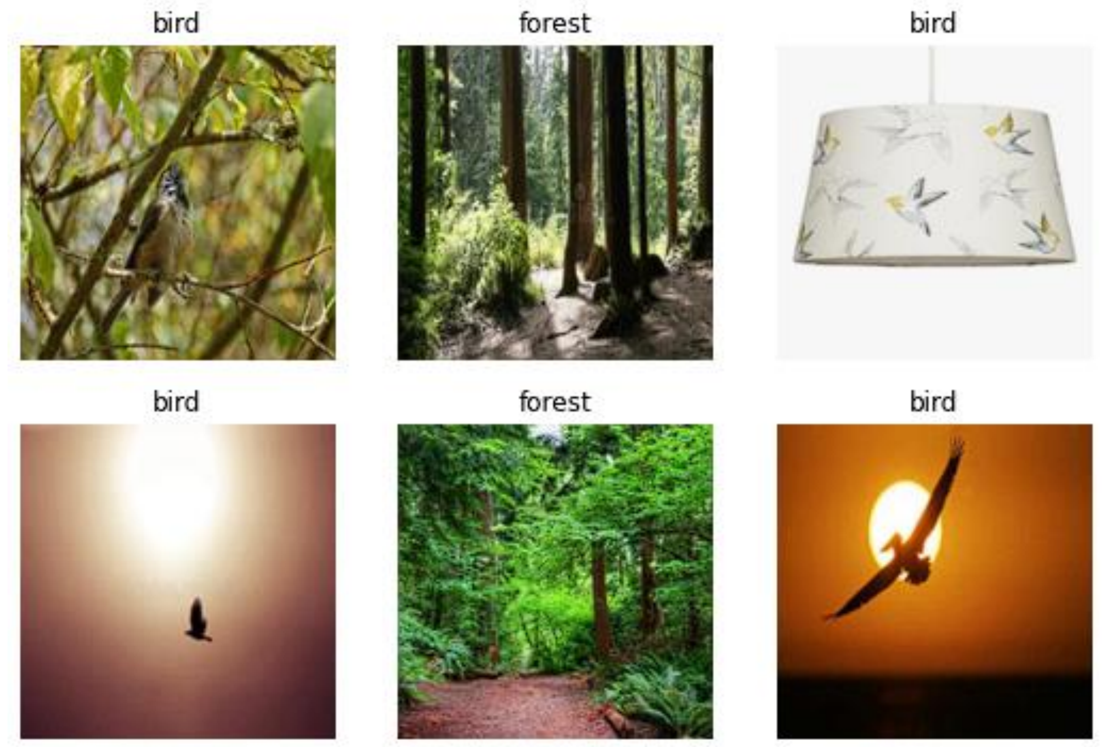
```
In [7]:
searches = 'forest','bird'
path = Path('bird_or_not')
from time import sleep

for o in searches:
    dest = (path/o)
    dest.mkdir(exist_ok=True, parents=True)
    download_images(dest, urls=search_images(f'{o} photo'))
    sleep(10)  # Pause between searches to avoid over-loading server
    download_images(dest, urls=search_images(f'{o} sun photo'))
    sleep(10)
    download_images(dest, urls=search_images(f'{o} shade photo'))
    sleep(10)
    resize_images(path/o, max_size=400, dest=path/o)
```

```
Searching for 'forest photo'
Searching for 'forest sun photo'
Searching for 'forest shade photo'
Searching for 'bird photo'
Searching for 'bird sun photo'
Searching for 'bird shade photo'
```



Creating a train and test dataset

- Datablocks from Dataloaders by fastai help in creating the training and validation data



```
dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    splitter=RandomSplitter(valid_pct=0.2, seed=42),
    get_y=parent_label,
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path, bs=32)

dls.show_batch(max_n=6)
```

Here what each of the `DataBlock` parameters means:

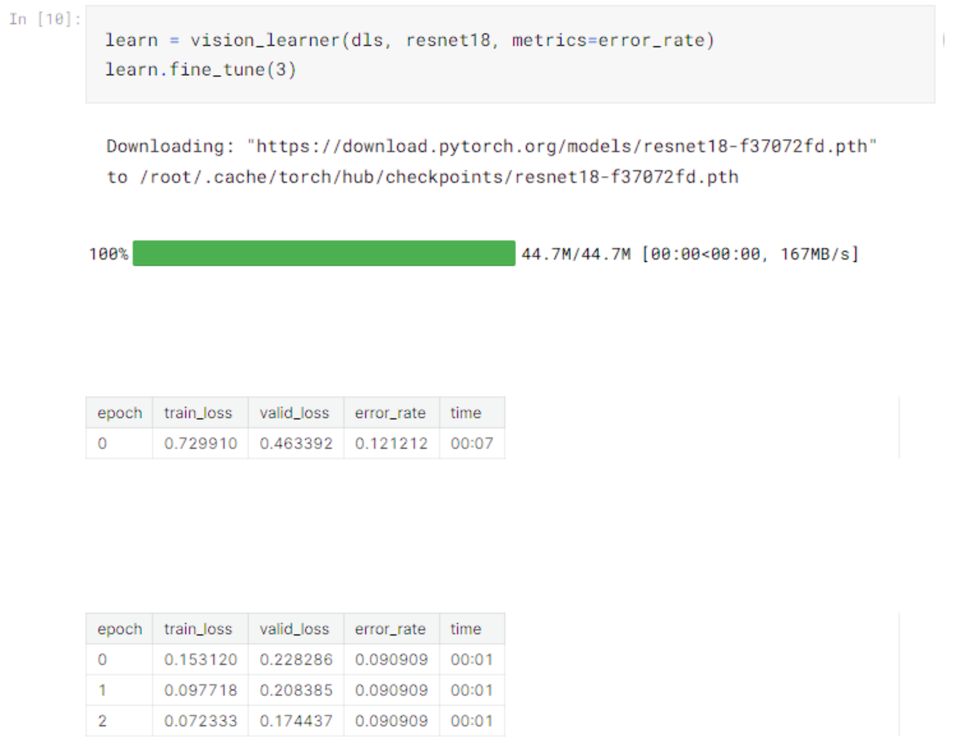
1. The inputs to our model are images, and the outputs are categories (in this case, "bird" or "forest")
2. To find all the inputs to our model, run the `get_image_files` function (which returns a list of all image files in a path).
3. Split the data into training and validation sets randomly, using 20% of the data for the validation set.
4. The labels (y values) is the name of the `parent` of each file (i.e. the name of the folder they're in, which will be *bird* or *forest*).
5. Before training, resize each image to 192x192 pixels by "squishing" it (as opposed to cropping it).

Learning from pretrained model resnet

Now we're ready to train our model. The fastest widely used computer vision model is `resnet18`. You can train this in a few minutes, even on a CPU! (On a GPU, it generally takes under 10 seconds...)

```
learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

"Fine-tuning" a model refers to training an already trained model aka pretrained model with our data.



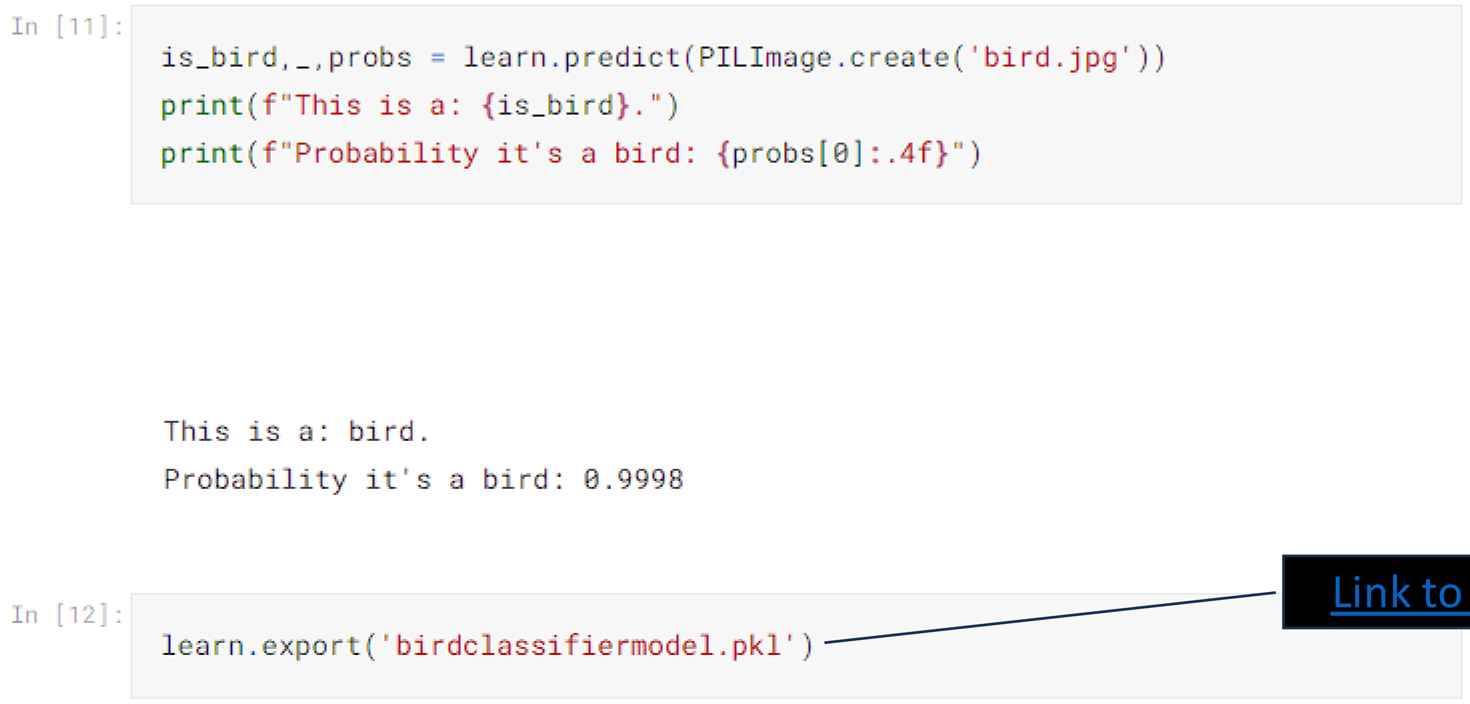
```
In [10]:
learn = vision_learner(dls, resnet18, metrics=error_rate)
learn.fine_tune(3)
```

```
Downloading: "https://download.pytorch.org/models/resnet18-f37072fd.pth" to /root/.cache/torch/hub/checkpoints/resnet18-f37072fd.pth

100% 44.7M/44.7M [00:00<00:00, 167MB/s]
```

epoch	train_loss	valid_loss	error_rate	time
0	0.729910	0.463392	0.121212	00:07

epoch	train_loss	valid_loss	error_rate	time
0	0.153120	0.228286	0.090909	00:01
1	0.097718	0.208385	0.090909	00:01
2	0.072333	0.174437	0.090909	00:01

Exporting the model after testing a sample data



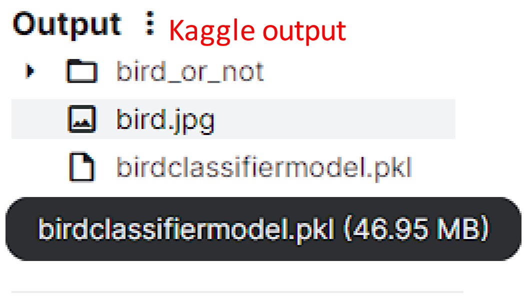
```
In [11]:
is_bird,_,probs = learn.predict(PILImage.create('bird.jpg'))
print(f"This is a: {is_bird}.")
print(f"Probability it's a bird: {probs[0]:.4f}")
```

```
This is a: bird.
Probability it's a bird: 0.9998
```

```
In [12]:
learn.export('birdclassifiermodel.pkl')
```

Link to the model: Bird classifier model .pkl

Kaggle output



Recap

What needs to be done:

~~1. We need a working model to classify a bird image.~~

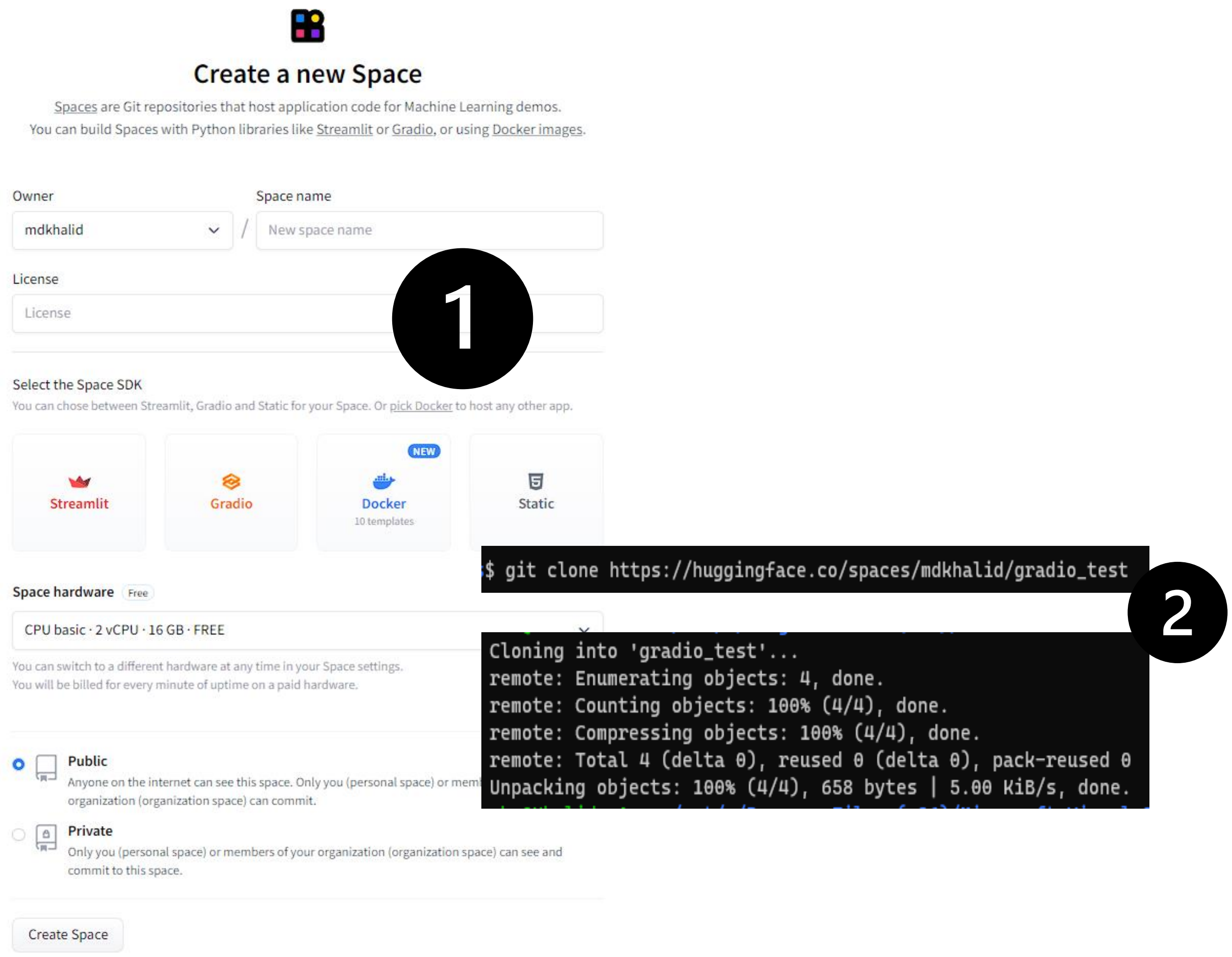
2. A method to make the model accessible to general public (website).

Link to the model: Bird classifier model .pkl

Ml Production - a Hugging Face Space by mdkhalid

Step 2: Create a web interface for public to use the model

- Link: [creating a Huggingface space]()
- Step 1: create a hugging face space , apache license, select gradio interface and make it public. We will Use gradio interface to host our model on Hugging face spaces for free
- Step 2: clone the space using terminal
- Step 3: download the classifier model pkl file into the cloned repo.
- Step 4: Add app.py file with code specified on gradio to communicate with your model



Create a new Space

Spaces are Git repositories that host application code for Machine Learning demos.
You can build Spaces with Python libraries like Streamlit or Gradio, or using Docker images.

Owner: mdkhalid / Space name: New space name

License: License

Select the Space SDK
You can chose between Streamlit, Gradio and Static for your Space. Or pick Docker to host any other app.

Streamlit | Gradio | Docker (NEW, 10 templates) | Static

Space hardware (Free)
CPU basic · 2 vCPU · 16 GB · FREE
You can switch to a different hardware at any time in your Space settings.
You will be billed for every minute of uptime on a paid hardware.

Public
Anyone on the internet can see this space. Only you (personal space) or mem
organization (organization space) can commit.

Private
Only you (personal space) or members of your organization (organization space) can see and commit to this space.

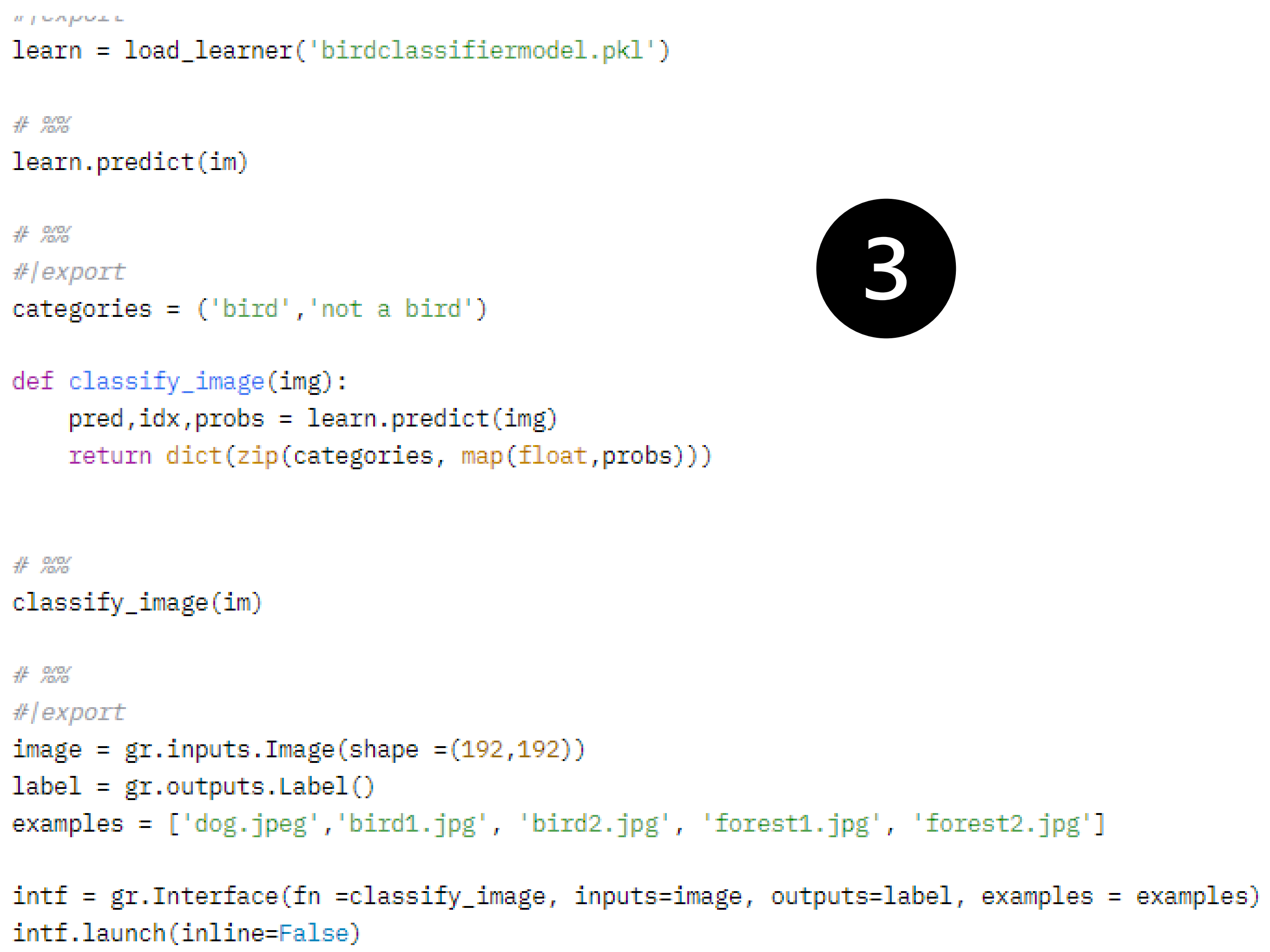
Create Space

```
$ git clone https://huggingface.co/spaces/mdkhalid/gradio_test
```

```
Cloning into 'gradio_test'...
remote: Enumerating objects: 4, done.
remote: Counting objects: 100% (4/4), done.
remote: Compressing objects: 100% (4/4), done.
remote: Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (4/4), 658 bytes | 5.00 KiB/s, done.
```

App.py file

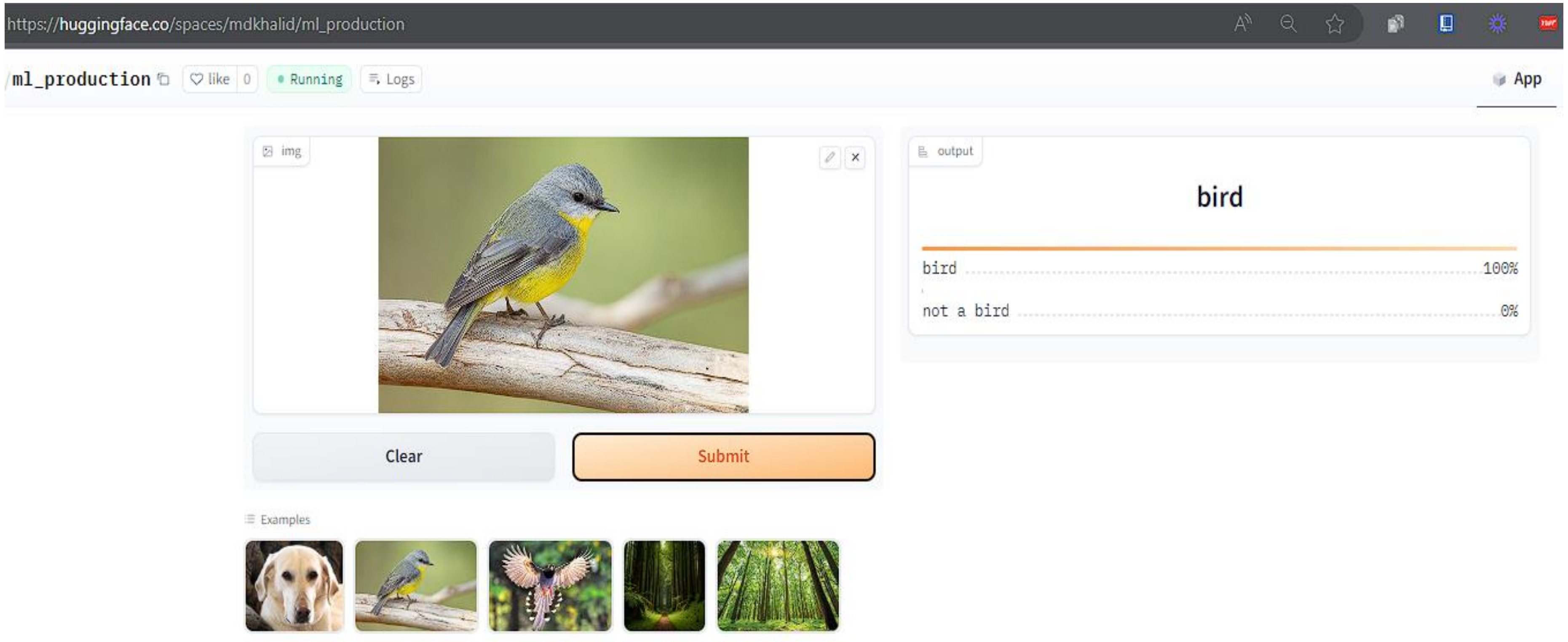
Gradio is an interface to take input pass it through a function and display output. It is easy to learn. Therefore it is preferred in this particular case. It is designed to read the content of app.py file therefore file must always be named in this fashion. We have modified it to show few examples also by placing some untested images in the root folder.



```
#|export
learn = load_learner('birdclassifiermodel.pkl')

# %%
learn.predict(im)

# %%
#|export
categories = ('bird','not a bird')

def classify_image(img):
    pred,idx,probs = learn.predict(img)
    return dict(zip(categories, map(float,probs)))


# %%
classify_image(im)

# %%
#|export
image = gr.inputs.Image(shape =(192,192))
label = gr.outputs.Label()
examples = ['dog.jpeg','bird1.jpg', 'bird2.jpg', 'forest1.jpg', 'forest2.jpg']

intf = gr.Interface(fn =classify_image, inputs=image, outputs=label, examples = examples)
intf.launch(inline=False)
```

3

Result


https://huggingface.co/spaces/mdkhalid/ml_production

```
bird

bird ........................................ 100%
not a bird .................................. 0%
```

Link to the web interface of the working model : Ml Production - a Hugging Face Space by mdkhalid

Key take aways

- Data collection needs to be streamlined. This helps in designing the labels
- Finetuning a pretrained model is faster and more effective than creating a whole new model.
- Fastai has incorporated best finetune practices so not many cycles are needed to get a decent model.
- Huggingface spaces offer good start to anyone unaware with web design therefore good for launching initial ML/DL projects.