Transfer Learning

Transfer learning

- Transfer learning is a machine learning technique that allows a model trained on one task to be repurposed for a new, related task.
- The key advantage of transfer learning is its ability to leverage existing knowledge, reducing the amount of data and computational resources required to train a model from scratch on the new task.
- This makes transfer learning particularly useful in scenarios where data is scarce or when training a brand-new model is prohibitively expensive.

Involves two steps:

1. **Pre-training phase**: In this phase, the model is trained on a large dataset known as the source task. This task is related, to some extent, to the target task (the new task) but is not exactly the same. The purpose of this step is to allow the model to learn some generalizable features or patterns.

2. **Fine-tuning phase**: After pre-training, the model is fine-tuned on the target task's dataset. This step often includes modifying parts of the model (such as the output layer) to fit the specific requirements of the new task and performing a limited number of training iterations on the new dataset to refine the model parameters.

Widely applicable across various fields:

Transfer learning is widely applicable across various fields, including but not limited to natural language processing, computer vision, and speech recognition.

- In natural language processing, a model pre-trained on a large corpus of text can be fine-tuned for specific tasks such as sentiment analysis (情感分析) or question answering (Q/A).

- In computer vision, a model trained on a broad set of image data can be fine-tuned for specific image recognition tasks, such as facial recognition (臉部辨識) or object classification (物品分類).

TorchVision

- TorchVision is an accompanying package for PyTorch, specifically designed for the computer vision domain.
- It provides a range of tools and pre-trained models to help researchers and developers achieve quick results in tasks like image classification, object detection, image transformations, and other visual tasks.

https://pytorch.org/vision/stable/datasets.html#built-in-datasets

The main features of TorchVision can be categorized into several parts:

1. **Datasets**: TorchVision offers a collection of common datasets such as ImageNet, CIFAR10, MNIST, etc., making it convenient for users to train and test models.
2. **Models**: It includes a series of pre-trained models like VGG, ResNet, Inception, etc. These models can be used directly for inference or as pre-trained models for further training.
3. **Transforms**: TorchVision provides various methods for image transformation, such as scaling, cropping, rotating, and color transformation, which helps in data augmentation and preprocessing.
4. **Utils**: This includes tools for image reading and saving, as well as tools that make it easier to export models to other platforms.

```
import torchvision
from torchvision import models
```

```
print(dir(models))
```

```
['AlexNet', 'AlexNet_Weights', 'ConvNeXt', 'ConvNeXt_Base_Weights',
'ConvNeXt_Large_Weights', 'ConvNeXt_Small_Weights', 'ConvNeXt_Tiny_
Weights', 'DenseNet', 'DenseNet121_Weights', 'DenseNet161_Weights',
'DenseNet169_Weights', 'DenseNet201_Weights', 'EfficientNet', 'EfficientNet_
B0_Weights', 'EfficientNet_B1_Weights',

....

'regnet_y_8gf', 'resnet', 'resnet101', 'resnet152', 'resnet18',
'resnet34', 'resnet50', 'resnext101_32x8d', 'resnext101_64x4d',
'resnext50_32x4d', 'segmentation', 'shufflenet_v2_x0_5', 'shufflenet_
v2_x1_0', 'shufflenet_v2_x1_5', 'shufflenet_v2_x2_0', 'shufflenetv2',
'squeezenet', 'squeezenet1_0', 'squeezenet1_1', 'swin_b', 'swin_s',
'swin_t', 'swin_transformer', 'vgg', 'vgg11', 'vgg11_bn', 'vgg13',
'vgg13_bn', 'vgg16', 'vgg16_bn', 'vgg19', 'vgg19_bn', 'video', 'vision_
transformer', 'vit_b_16', 'vit_b_32', 'vit_h_14', 'vit_l_16', 'vit_l_32',
'wide_resnet101_2', 'wide_resnet50_2']
```

The following classification models are available, with or without pre-trained weights:

- AlexNet
- ConvNeXt
- DenseNet
- EfficientNet
- EfficientNetV2
- GoogLeNet
- Inception V3
- MaxVit
- MNASNet
- MobileNet V2
- MobileNet V3
- RegNet
- ResNet
- ResNeXt
- ShuffleNet V2
- SqueezeNet
- SwinTransformer
- VGG
- VisionTransformer
- Wide ResNet

https://pytorch.org/vision/stable/models.html

Here is an example of how to use the pre-trained image classification models:

```
from torchvision.io import read_image
from torchvision.models import resnet50, ResNet50_Weights

img = read_image("test/assets/encode_jpeg/grace_hopper_517x606.jpg")

# Step 1: Initialize model with the best available weights
weights = ResNet50_Weights.DEFAULT
model = resnet50(weights=weights)
model.eval()

# Step 2: Initialize the inference transforms
preprocess = weights.transforms()

# Step 3: Apply inference preprocessing transforms
batch = preprocess(img).unsqueeze(0)

# Step 4: Use the model and print the predicted category
prediction = model(batch).squeeze(0).softmax(0)
class_id = prediction.argmax().item()
score = prediction[class_id].item()
category_name = weights.meta["categories"][class_id]
print(f"{category_name}: {100 * score:.1f}%")
```

https://pytorch.org/vision/stable/models.html

Correct answer is within the top five highest-scoring categories predicted by the model.

Table of all available classification weights

Accuracies are reported on ImageNet-1K using single crops:

Weight	Acc@1 (Accuracy at 1)	Acc@5 (Accuracy at 5)	Params (Total number of trainable parameters in the model)	GFLOPS	Recipe
AlexNet_Weights.IMAGENET1K_V1	56.522	79.066	61.1M	0.71	link
ConvNeXt_Base_Weights.IMAGENET1K_V1	84.062	96.87	88.6M	15.36	link
ConvNeXt_Large_Weights.IMAGENET1K_V1	84.414	96.976	197.8M	34.36	link
ConvNeXt_Small_Weights.IMAGENET1K_V1	83.616	96.65	50.2M	8.68	link
ConvNeXt_Tiny_Weights.IMAGENET1K_V1	82.52	96.146	28.6M	4.46	link
DenseNet121_Weights.IMAGENET1K_V1	74.434	91.972	8.0M	2.83	link
DenseNet161_Weights.IMAGENET1K_V1	77.138	93.56	28.7M	7.73	link
DenseNet169_Weights.IMAGENET1K_V1	75.6	92.806	14.1M	3.36	link
DenseNet201_Weights.IMAGENET1K_V1	76.896	93.37	20.0M	4.29	link

GFLOPS: Number of floating-point operations required for the model to perform one forward inference.

Recipe: The specific training process or settings used to achieve these performance metrics

https://pytorch.org/vision/stable/models.html

IMAGENET

14,197,122 images, 21841 synsets indexed

Home Download Challenges About

Not logged in. Login I Signup

Download

Download ImageNet Data

The most highly-used subset of ImageNet is the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012-2017 image classification and localization dataset. This dataset spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. This subset is available on Kaggle.

For access to the full ImageNet dataset and other commonly used subsets, please login or request access. In doing so, you will need to agree to our terms of access.

https://www.image-net.org

imagenet_classes.csv

A	B
0	tench
1	goldfish
2	great_white_shark
3	tiger_shark
4	hammerhead
5	electric_ray
6	stingray
7	cock
8	hen
9	ostrich
10	brambling
11	goldfinch
12	house_finch
13	junco
14	indigo_bunting

989	hip
990	buckeye
991	coral_fungus
992	agaric
993	gyromitra
994	stinkhorn
995	earthstar
996	hen-of-the-woods
997	bolete
998	ear
999	toilet_tissue

Hugging Face
https://huggingface.co › datasets › imag...

imagenet-1k · Datasets at Hugging Face

This dataset spans 1000 object classes and contains 1,281,167 training images, 50,000 validation images and 100,000 test images. The version also has the patch ...

https://huggingface.co/datasets/imagenet-1k

ImageNet-21K Pretraining for the Masses

Tal Ridnik, Emanuel Ben-Baruch, Asaf Noy, Lihi Zelnik-Manor

ImageNet-1K serves as the primary dataset for pretraining deep learning models for computer vision tasks. ImageNet-21K dataset, which is bigger and more diverse, is used less frequently for pretraining, mainly due to its complexity, low accessibility, and underestimation of its added value. This paper aims to close this gap, and make high-quality efficient pretraining on ImageNet-21K available for everyone. Via a dedicated preprocessing stage, utilization of WordNet hierarchical structure, and a novel training scheme called semantic softmax, we show that various models significantly benefit from ImageNet-21K pretraining on numerous datasets and tasks, including small mobile-oriented models. We also show that we outperform previous ImageNet-21K pretraining schemes for prominent new models like ViT and Mixer. Our proposed pretraining pipeline is efficient, accessible, and leads to SoTA reproducible results, from a publicly available dataset. The training code and pretrained models are available at: this https URL

https://github.com/Alibaba-MIIL/ImageNet21K

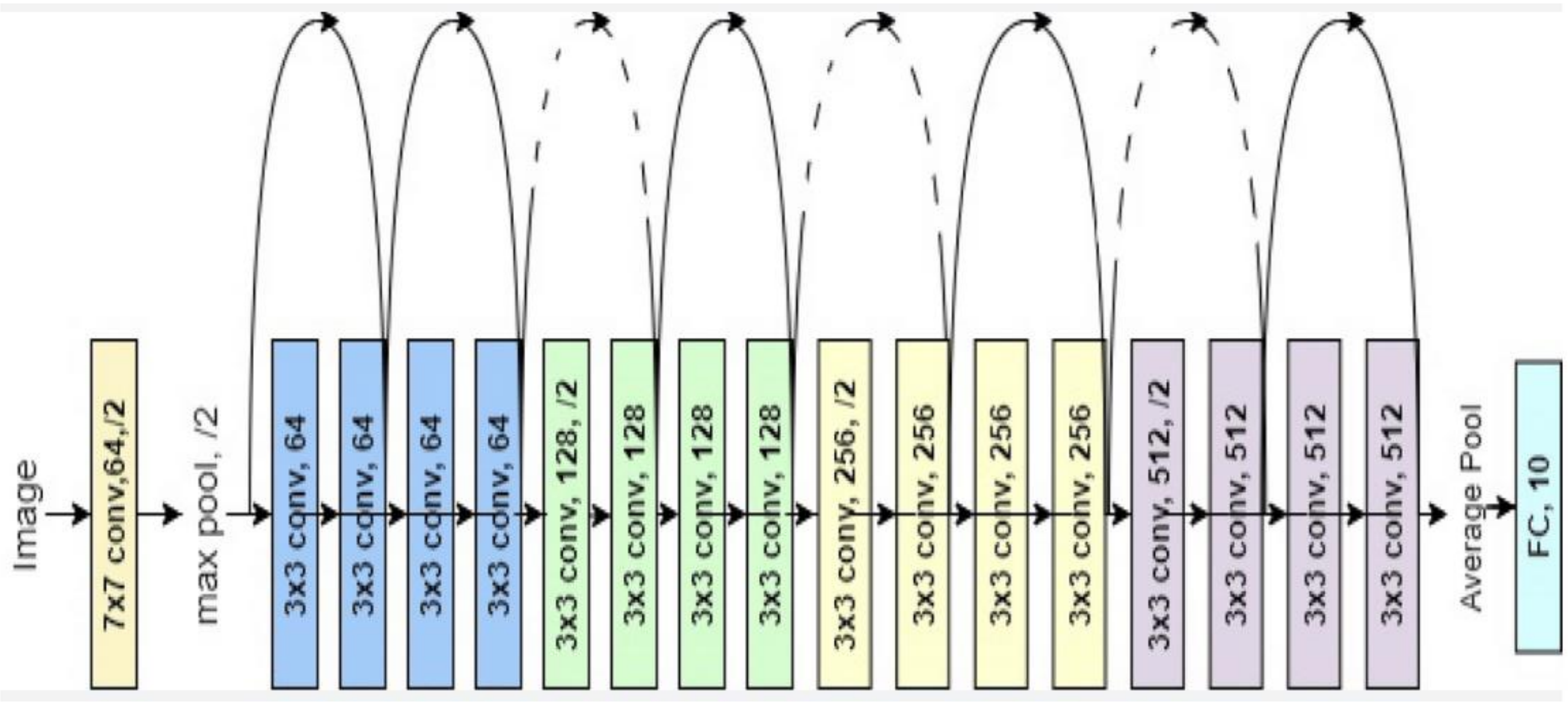
`Example: ResNet18_Weights`

Example:

18 layers

```
import torchvision.models as models
from torchvision.models.resnet import ResNet18_Weights

resnet=models.resnet18(weights='ResNet18_Weights.DEFAULT')
```



https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html

```python
img=Image.open("data/dog.jpg")
#img.show()
img
```

```python
preprocess = transforms.Compose([
    transforms.Resize(256), #Resize the image to 256 pixels on the shortest side while maintaining aspect ratio.
    transforms.CenterCrop(244), #Crop the center of the image to make it 244 pixels in both height and width.
    transforms.ToTensor() #Convert the image to a PyTorch tensor. This also scales the image's pixel values to [0, 1].
])
```

```python
# Apply preprocessing transformations to the input image 'img'
# This typically includes operations such as resizing, normalization, etc.,
# to match the input specifications of the model (e.g., ResNet-18).
img2 = preprocess(img)
print(img2.shape)
```

```
torch.Size([3, 244, 244])
```

```python
# Add an extra dimension to 'img2' at the first position (dimension index 0).
# This operation transforms 'img2' from a 3D tensor (C x H x W) to a 4D tensor (1 x C x H x W),
# simulating a batch of size 1. PyTorch models typically expect inputs in batches,
# so this step is necessary for a single image.
# 'torch' is the main namespace of PyTorch; 'unsqueeze' is a function that adds a dimension of size one.
img3 = torch.unsqueeze(img2, 0)
print(img3.shape)
```

```
torch.Size([1, 3, 244, 244])
```

```python
resnet=models.resnet18(weights='ResNet18_Weights.DEFAULT')
```

```
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, 119MB/s]
```

```python
# Sets the ResNet model to evaluation mode.
resnet.eval()
out=resnet(img3)
```

```python
out_numpy=out.detach().numpy()  # 轉為NumPy
out_class=np.argmax(out_numpy, axis=1)  # 找出最大值的索引
print(out_class)
```

```
[208]
```

```python
df=pd.read_csv("data/imagenet_classes.csv",header=None)
print(df.head())
```

```
   0                 1
0  0             tench
1  1          goldfish
2  2  great_white_shark
3  3       tiger_shark
4  4        hammerhead
```

```python
label = df.iloc[out_class].values
print(label)
```

```
[[208 ' Labrador_retriever']]
```

```python
score = torch.nn.functional.softmax(out, dim=1)[0] * 100
print(score.shape)
```

```
torch.Size([1000])
```

```python
print(f"score:{score[out_class].item():.2f}")
```

```
score:91.69
```

Evaluation mode:

•**Disables Dropout**: In evaluation mode, all the dropout layers in the model are deactivated. Dropout layers randomly drop out (set to zero) a fraction of input units during training to prevent overfitting. However, during inference, you want to use the full capacity of the model without randomly dropping out nodes, so dropout is turned off.

•**Freezes Batch Normalization Layers**: Batch normalization layers normalize the input or the activations of the previous layer to have zero mean and unit variance. This is done differently during training and inference. During training, batch statistics (mean and variance of the current batch) are used for normalization, and these statistics are also updated to compute running estimates that are used during inference. In evaluation mode, these running estimates are used instead of batch statistics to ensure consistency in the outputs, as the model is no longer being updated.

把 BatchNormalization 固定、停用Dropout，用訓練好的值

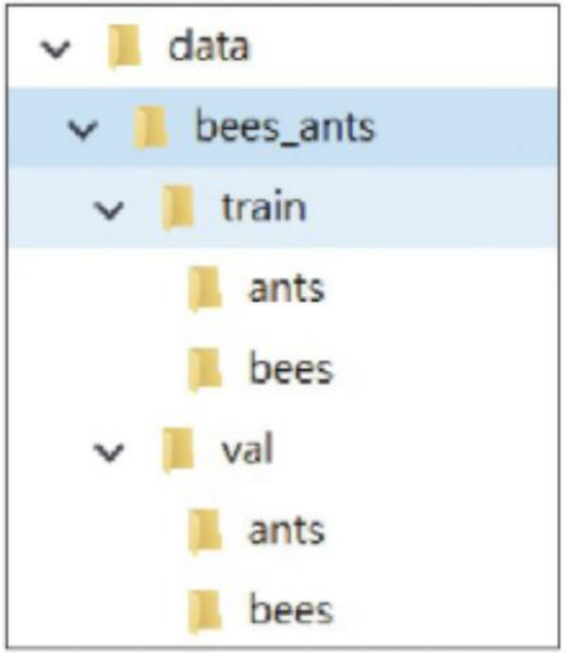
ImageFolder

ImageFolder

ImageFolder is a class in PyTorch's torchvision.datasets module for handling image datasets that are stored in a directory structure with each folder named after the class it represents.

Inside each folder are the images that belong to that class.

ImageFolder automatically maps this folder structure to a dataset with labels, making it ideal for training classification models.

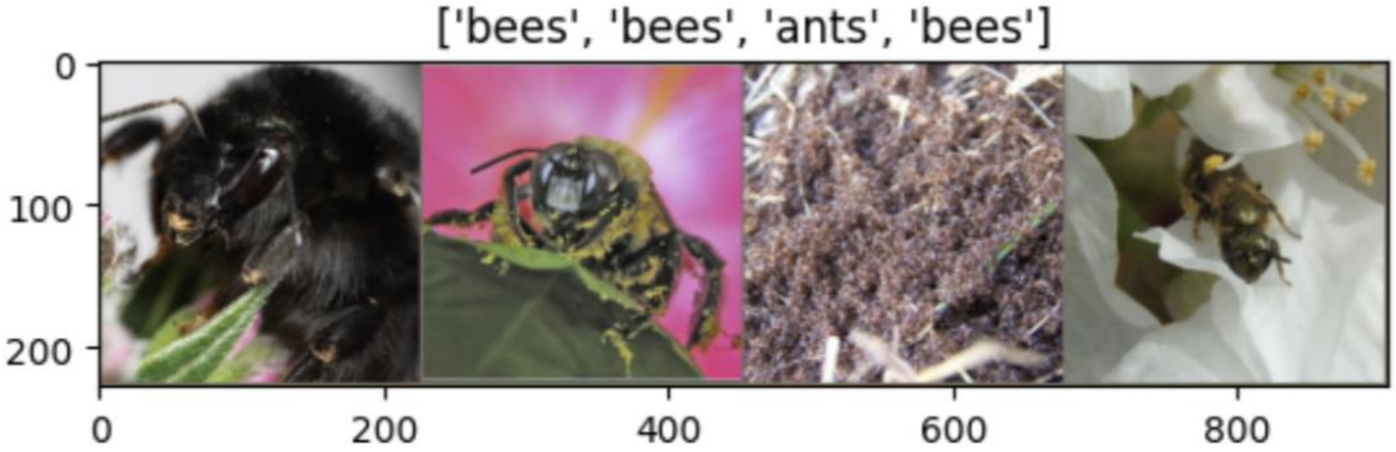



```
train_dataset=datasets.ImageFolder(
    root="data/bees_ants/train",
    transform=train_transforms
)
```

```
val_dataset=datasets.ImageFolder(
    root="data/bees_ants/val",
    transform=val_transforms
)
```

Exercise:



ex09_bee.ipynb

```python
# Define a sequence of transformations to be applied to training data

train_transforms=transforms.Compose([

    # Randomly crop the image to size 224x224
    transforms.RandomResizedCrop(224),

    # Randomly flip the image horizontally
    transforms.RandomHorizontalFlip(),

    # Convert the image to a PyTorch tensor
    transforms.ToTensor(),

    # Normalize the image with mean [0.485, 0.456, 0.406] and standard deviation [0.229, 0.224, 0.225]
    transforms.Normalize(
        [0.485,0.456,0.406],
        [0.229,0.224,0.225]
    )
])
```

```
train_dataset=datasets.ImageFolder(
    root="data/bees_ants/train",
    transform=train_transforms
)
```

```
val_dataset=datasets.ImageFolder(
    root="data/bees_ants/val",
    transform=val_transforms

)
```

```
train_loader=torch.utils.data.DataLoader(  # Create a DataLoader object for training data
    train_dataset,  # Use the train_dataset as the source of data
    batch_size=4,   # Set the batch size to 4, meaning each iteration will process 4 samples
    shuffle=True,   # Shuffle the data at the beginning of each epoch to introduce randomness
    num_workers=4   # Use 4 worker processes to load data in parallel, which can speed up the process
)

```

```
val_loader=torch.utils.data.DataLoader(
    val_dataset,
    batch_size=4,
    shuffle=True,
    num_workers=4
)
```

```
model=models.resnet18(weights='ResNet18_Weights.DEFAULT')
```

```
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, 132MB/s]
```

```
print(model.fc)
```

```
Linear(in_features=512, out_features=1000, bias=True)
```

```
model.fc=nn.Linear(model.fc.in_features,2)
print(model.fc)
```

```
Linear(in_features=512, out_features=2, bias=True)
```

```
device = 'cuda' if torch.cuda.is_available() else 'cpu'
model=model.to(device)
print(device)
```

```
cpu
```

Decays the learning rate of each parameter group by gamma every step_size epochs.

```
criterion=nn.CrossEntropyLoss()
optimizer=optim.SGD(model.parameters(),lr=0.001,momentum=0.9)
```

```
from torch.optim.lr_scheduler import StepLR
exp_lr_scheduler=StepLR(optimizer,step_size=7,gamma=0.1)
```

Assuming the original learning rate is set to 0.001,
during training, the learning rate will change by a factor of 0.1 every 7 epochs.
After 7 epochs, the learning rate becomes 0.0001,
and after another 7 epochs, the learning rate becomes 0.00001.

```
from torch.optim.lr_scheduler import StepLR

# Create a StepLR scheduler
scheduler = StepLR(optimizer, step_size=5, gamma=0.1)

# Within the training loop
for epoch in range(epochs):
    # Train the model
    ...

    # Update the learning rate
    scheduler.step()
```

StepLR() decreases the learning rate by a factor after a specified number of epochs.

Create an instance of StepLR() and pass it the optimizer and the step size, among other parameters.

Call scheduler.step() after each epoch.

https://pytorch.org/docs/stable/generated/torch.optim.lr_scheduler.StepLR.html

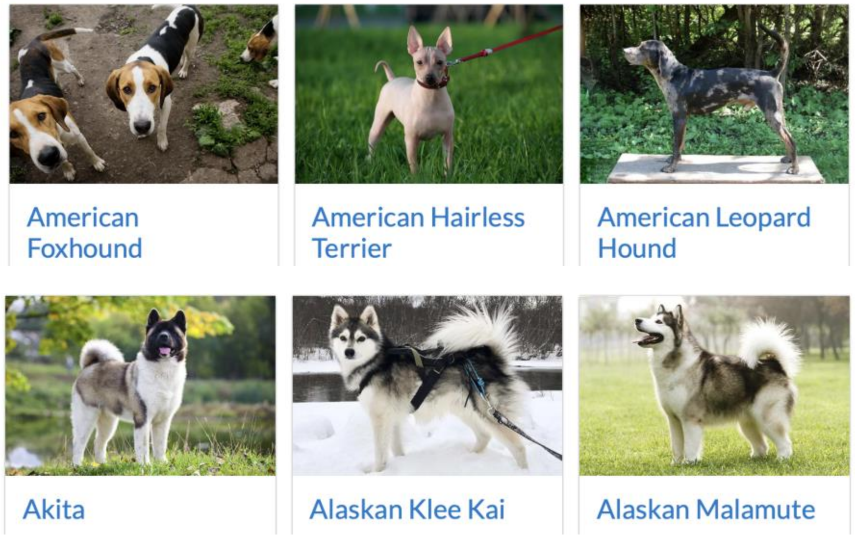
Exercise:

Create a breed identifier, choose an animal, collect images, adjust the network, so that the identifier can distinguish between different breeds (at least 8 breeds).

Submission requirements:

1. source code transferlearning.py
2. PDF documents
    1. Explaining the steps.
        1. Image collecting.
        2. Image processing.
        3. Build/test the model.
    2. Show the outputs.
3. Upload to e-learning before 4/26 14:10

https://www.akc.org/dog-breeds/



Bird breeds

來自網路來源


Canary


Cockatiel


Budgerigar


Lovebird


Parrot


Pionus parrots


Columbidae


Finch


Macaw

Fish breeds

來自網路來源


Betta


Goldfish


Danios


Angelfish


Guppy


Harlequin rasbora


Neon Tetra


Tetra


Barb

Guinea pig breeds

來自網路來源


Abyssinian


American Guinea pig


English Crested Guinea …


Guinea pig


Teddy guinea pig


Silkie guinea pig


Texel guinea pig


Peruvian guinea pig


Merino

Snake species

來自網路來源


Brown snakes


Racer


Burmese python


Corn snake


Rat snakes


Garter snake


Boa constrictor


Ball python


Jamaican tree snake