In [1]:

```
import tensorflow as tf
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import cv2
import random
import os
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.layers import *
from tensorflow.keras.models import Sequential,Model

print(tf.__version__)
```

```
2.11.0
```

In [2]:

```
path = '/kaggle/input/animals10/raw-img'
# path = '/kaggle/input/animal-image-dataset-90-different-animals/animals/animals'
```

In [3]:

```
names = []
nums = []
data = {'Name of class':[],'Number of samples':[]}

for i in os.listdir(path):
    nums.append(len(os.listdir(path+'/'+i)))
    names.append(i)

num_classes = len(names)

data['Name of class']+=names
data['Number of samples']+=nums

df = pd.DataFrame(data)
df
```

Out[3]:

	Name of class	Number of samples
0	cavallo	2623
1	pecora	1820
2	elefante	1446
3	gatto	1668
4	scoiattolo	1862
5	gallina	3098
6	ragno	4821
7	mucca	1866
8	cane	4863
9	farfalla	2112

In [4]:

```
shape = (128,128)
```

In [5]:

```
# image_datagen = tf.keras.preprocessing.image.ImageDataGenerator(rescale = 1./255 , rota
tion_range=20,
#                                                                    width_shift_range=0.2
,
#                                                                    height_shift_range=0.
2,
#                                                                    horizontal_flip=True,
validation_split=0.2)
train_ds = tf.keras.utils.image_dataset_from_directory(
  path,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=shape,
  batch_size=32)

val_ds = tf.keras.utils.image_dataset_from_directory(
  path,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=shape,
  batch_size=32)

train_ds = train_ds.cache().prefetch(buffer_size=100)
val_ds = val_ds.cache().prefetch(buffer_size=100)
```

```
Found 26179 files belonging to 10 classes.
Using 20944 files for training.
Found 26179 files belonging to 10 classes.
Using 5235 files for validation.
```

In [6]:

```
translate = {"cane": "狗", "cavallo": "马", "elefante": "大象", "farfalla": "蝴蝶", "galli
na": "鸡", "gatto": "猫", "mucca": "牛", "pecora": "羊", "scoiattolo": "松鼠", "ragno": "
蜘蛛"}
# image_dataset_from_directory 函数会按字母排序label，所以这里也按字母排序
s = sorted(list(translate.keys()))
labels = [translate[x] for x in s]
print(labels)
```

```
['狗', '马', '大象', '蝴蝶', '鸡', '猫', '牛', '羊', '蜘蛛', '松鼠']
```

In [7]:

```
for image_batch, labels_batch in train_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
for image_batch, labels_batch in val_ds:
  print(image_batch.shape)
  print(labels_batch.shape)
  break
```

```
(32, 128, 128, 3)
(32,)
(32, 128, 128, 3)
(32,)
```

In [8]:

```
data_augmentation = keras.Sequential(
    [layers.RandomFlip("horizontal"), layers.RandomRotation(0.1),]
)

# mobilenet 需要输入归一化的像素
normalization_layer = layers.Rescaling(scale=1./127.5,offset=-1)
# normalization_layer = layers.Rescaling(scale=1./255)
```

In [9]:

```
data_process = keras.Sequential(
    [data_augmentation, normalization_layer]
)

train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
```

In [10]:

```
pretrained = tf.keras.applications.MobileNet(
    input_shape=(*shape, 3),
    alpha=0.25,
    depth_multiplier=1,
    dropout=0.001,
    include_top=True,
    weights="imagenet",
    input_tensor=None,
    pooling=None,
    classes=1000,
    classifier_activation="softmax",
)

pretrained.trainable = False



model = tf.keras.models.Sequential()

model.add(tf.keras.layers.InputLayer(input_shape=(*shape , 3)))
model.add(data_augmentation)
model.add(tf.keras.Model(inputs=pretrained.inputs, outputs=pretrained.layers[-5].output)
)

model.add(tf.keras.layers.Reshape((-1,)))
model.add(tf.keras.layers.Dropout(0.5))
model.add(tf.keras.layers.Flatten())
model.add(tf.keras.layers.Dense(num_classes))



model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/mobile
net/mobilenet_2_5_128_tf.h5
2108140/2108140 [==============================] - 1s 0us/step
```

In [11]:

```
# model = tf.keras.applications.MobileNet(
#     input_shape=(*shape , 3), weights=None, alpha=0.25 ,include_top=True ,classes=num_c
lasses
# )


# model.compile(optimizer='adam',
#               loss=tf.keras.losses.SparseCategoricalCrossentropy(),
#               metrics=['accuracy'])
```

In [12]:

```
model.summary()
history = model.fit(train_ds , validation_data = val_ds , epochs = 15)
```

```
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 128, 128, 3)       0

 model (Functional)          (None, 1, 1, 256)         218544
```

```
 reshape (Reshape)           (None, 256)               0

 dropout (Dropout)           (None, 256)               0

 flatten (Flatten)           (None, 256)               0

 dense (Dense)               (None, 10)                2570

=================================================================
Total params: 221,114
Trainable params: 2,570
Non-trainable params: 218,544
_________________________________________________________________
Epoch 1/15
655/655 [==============================] - 86s 125ms/step - loss: 1.7637 - accuracy: 0.47
70 - val_loss: 0.6575 - val_accuracy: 0.7780
Epoch 2/15
655/655 [==============================] - 66s 101ms/step - loss: 1.0318 - accuracy: 0.64
83 - val_loss: 0.5835 - val_accuracy: 0.8094
Epoch 3/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9473 - accuracy: 0.67
33 - val_loss: 0.5744 - val_accuracy: 0.8071
Epoch 4/15
655/655 [==============================] - 67s 103ms/step - loss: 0.9369 - accuracy: 0.67
56 - val_loss: 0.5625 - val_accuracy: 0.8185
Epoch 5/15
655/655 [==============================] - 66s 101ms/step - loss: 0.9372 - accuracy: 0.67
54 - val_loss: 0.5606 - val_accuracy: 0.8199
Epoch 6/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9309 - accuracy: 0.68
13 - val_loss: 0.5731 - val_accuracy: 0.8136
Epoch 7/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9400 - accuracy: 0.67
80 - val_loss: 0.5630 - val_accuracy: 0.8166
Epoch 8/15
655/655 [==============================] - 67s 103ms/step - loss: 0.9291 - accuracy: 0.68
18 - val_loss: 0.5580 - val_accuracy: 0.8164
Epoch 9/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9358 - accuracy: 0.67
85 - val_loss: 0.5619 - val_accuracy: 0.8149
Epoch 10/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9307 - accuracy: 0.68
10 - val_loss: 0.5615 - val_accuracy: 0.8174
Epoch 11/15
655/655 [==============================] - 66s 102ms/step - loss: 0.9370 - accuracy: 0.67
68 - val_loss: 0.5708 - val_accuracy: 0.8132
Epoch 12/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9356 - accuracy: 0.67
87 - val_loss: 0.5626 - val_accuracy: 0.8174
Epoch 13/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9270 - accuracy: 0.68
20 - val_loss: 0.5594 - val_accuracy: 0.8162
Epoch 14/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9347 - accuracy: 0.67
68 - val_loss: 0.5662 - val_accuracy: 0.8120
Epoch 15/15
655/655 [==============================] - 67s 102ms/step - loss: 0.9386 - accuracy: 0.67
50 - val_loss: 0.5593 - val_accuracy: 0.8166
```

In [13]:

```
# Unfreeze the base model
pretrained.trainable = True

# It's important to recompile your model after you make any changes
# to the `trainable` attribute of any inner layer, so that your changes
# are take into account
model.compile(optimizer=keras.optimizers.Adam(1e-5),  # Very low learning rate
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```python
# Train end-to-end. Be careful to stop before you overfit!
history = model.fit(train_ds , validation_data = val_ds , epochs = 80)
```

```
Epoch 1/80
655/655 [==============================] - 207s 300ms/step - loss: 1.0215 - accuracy: 0.6
520 - val_loss: 0.5900 - val_accuracy: 0.8008
Epoch 2/80
655/655 [==============================] - 196s 300ms/step - loss: 0.9634 - accuracy: 0.6
679 - val_loss: 0.5580 - val_accuracy: 0.8120
Epoch 3/80
655/655 [==============================] - 196s 300ms/step - loss: 0.9016 - accuracy: 0.6
926 - val_loss: 0.5395 - val_accuracy: 0.8180
Epoch 4/80
655/655 [==============================] - 197s 300ms/step - loss: 0.8614 - accuracy: 0.7
034 - val_loss: 0.5259 - val_accuracy: 0.8222
Epoch 5/80
655/655 [==============================] - 199s 303ms/step - loss: 0.8358 - accuracy: 0.7
117 - val_loss: 0.5133 - val_accuracy: 0.8269
Epoch 6/80
655/655 [==============================] - 198s 302ms/step - loss: 0.8068 - accuracy: 0.7
232 - val_loss: 0.5041 - val_accuracy: 0.8292
Epoch 7/80
655/655 [==============================] - 199s 303ms/step - loss: 0.7934 - accuracy: 0.7
258 - val_loss: 0.4943 - val_accuracy: 0.8329
Epoch 8/80
655/655 [==============================] - 199s 304ms/step - loss: 0.7761 - accuracy: 0.7
354 - val_loss: 0.4891 - val_accuracy: 0.8340
Epoch 9/80
655/655 [==============================] - 200s 306ms/step - loss: 0.7499 - accuracy: 0.7
425 - val_loss: 0.4833 - val_accuracy: 0.8369
Epoch 10/80
655/655 [==============================] - 199s 304ms/step - loss: 0.7383 - accuracy: 0.7
463 - val_loss: 0.4783 - val_accuracy: 0.8386
Epoch 11/80
655/655 [==============================] - 200s 306ms/step - loss: 0.7251 - accuracy: 0.7
560 - val_loss: 0.4742 - val_accuracy: 0.8395
Epoch 12/80
655/655 [==============================] - 201s 307ms/step - loss: 0.7224 - accuracy: 0.7
534 - val_loss: 0.4680 - val_accuracy: 0.8413
Epoch 13/80
655/655 [==============================] - 199s 304ms/step - loss: 0.7155 - accuracy: 0.7
559 - val_loss: 0.4636 - val_accuracy: 0.8432
Epoch 14/80
655/655 [==============================] - 199s 305ms/step - loss: 0.6927 - accuracy: 0.7
633 - val_loss: 0.4595 - val_accuracy: 0.8455
Epoch 15/80
655/655 [==============================] - 200s 305ms/step - loss: 0.6837 - accuracy: 0.7
655 - val_loss: 0.4553 - val_accuracy: 0.8470
Epoch 16/80
655/655 [==============================] - 200s 305ms/step - loss: 0.6684 - accuracy: 0.7
728 - val_loss: 0.4516 - val_accuracy: 0.8485
Epoch 17/80
655/655 [==============================] - 199s 305ms/step - loss: 0.6588 - accuracy: 0.7
758 - val_loss: 0.4469 - val_accuracy: 0.8489
Epoch 18/80
655/655 [==============================] - 201s 307ms/step - loss: 0.6523 - accuracy: 0.7
791 - val_loss: 0.4427 - val_accuracy: 0.8520
Epoch 19/80
655/655 [==============================] - 200s 305ms/step - loss: 0.6397 - accuracy: 0.7
810 - val_loss: 0.4411 - val_accuracy: 0.8506
Epoch 20/80
655/655 [==============================] - 201s 306ms/step - loss: 0.6413 - accuracy: 0.7
833 - val_loss: 0.4374 - val_accuracy: 0.8516
Epoch 21/80
655/655 [==============================] - 201s 307ms/step - loss: 0.6292 - accuracy: 0.7
835 - val_loss: 0.4337 - val_accuracy: 0.8537
Epoch 22/80
655/655 [==============================] - 201s 306ms/step - loss: 0.6217 - accuracy: 0.7
896 - val_loss: 0.4307 - val_accuracy: 0.8548
Epoch 23/80
655/655 [==============================] - 201s 307ms/step - loss: 0.6148 - accuracy: 0.7
```

```
913 - val_loss: 0.4277 - val_accuracy: 0.8560
Epoch 24/80
655/655 [==============================] - 199s 304ms/step - loss: 0.6051 - accuracy: 0.7
968 - val_loss: 0.4265 - val_accuracy: 0.8548
Epoch 25/80
655/655 [==============================] - 199s 304ms/step - loss: 0.6094 - accuracy: 0.7
898 - val_loss: 0.4213 - val_accuracy: 0.8567
Epoch 26/80
655/655 [==============================] - 200s 305ms/step - loss: 0.5869 - accuracy: 0.8
008 - val_loss: 0.4206 - val_accuracy: 0.8548
Epoch 27/80
655/655 [==============================] - 200s 305ms/step - loss: 0.5977 - accuracy: 0.7
965 - val_loss: 0.4186 - val_accuracy: 0.8564
Epoch 28/80
655/655 [==============================] - 199s 304ms/step - loss: 0.5756 - accuracy: 0.8
050 - val_loss: 0.4150 - val_accuracy: 0.8594
Epoch 29/80
655/655 [==============================] - 199s 303ms/step - loss: 0.5744 - accuracy: 0.8
046 - val_loss: 0.4121 - val_accuracy: 0.8596
Epoch 30/80
655/655 [==============================] - 201s 307ms/step - loss: 0.5760 - accuracy: 0.8
030 - val_loss: 0.4108 - val_accuracy: 0.8604
Epoch 31/80
655/655 [==============================] - 202s 308ms/step - loss: 0.5734 - accuracy: 0.8
052 - val_loss: 0.4080 - val_accuracy: 0.8609
Epoch 32/80
655/655 [==============================] - 201s 307ms/step - loss: 0.5624 - accuracy: 0.8
086 - val_loss: 0.4079 - val_accuracy: 0.8615
Epoch 33/80
655/655 [==============================] - 200s 304ms/step - loss: 0.5557 - accuracy: 0.8
123 - val_loss: 0.4055 - val_accuracy: 0.8627
Epoch 34/80
655/655 [==============================] - 202s 308ms/step - loss: 0.5471 - accuracy: 0.8
134 - val_loss: 0.4022 - val_accuracy: 0.8628
Epoch 35/80
655/655 [==============================] - 200s 306ms/step - loss: 0.5501 - accuracy: 0.8
139 - val_loss: 0.4024 - val_accuracy: 0.8638
Epoch 36/80
655/655 [==============================] - 200s 305ms/step - loss: 0.5435 - accuracy: 0.8
156 - val_loss: 0.4005 - val_accuracy: 0.8646
Epoch 37/80
655/655 [==============================] - 201s 306ms/step - loss: 0.5392 - accuracy: 0.8
172 - val_loss: 0.3996 - val_accuracy: 0.8649
Epoch 38/80
655/655 [==============================] - 202s 308ms/step - loss: 0.5317 - accuracy: 0.8
217 - val_loss: 0.3994 - val_accuracy: 0.8649
Epoch 39/80
655/655 [==============================] - 201s 307ms/step - loss: 0.5305 - accuracy: 0.8
211 - val_loss: 0.3958 - val_accuracy: 0.8659
Epoch 40/80
655/655 [==============================] - 201s 306ms/step - loss: 0.5262 - accuracy: 0.8
221 - val_loss: 0.3933 - val_accuracy: 0.8663
Epoch 41/80
655/655 [==============================] - 201s 307ms/step - loss: 0.5266 - accuracy: 0.8
216 - val_loss: 0.3937 - val_accuracy: 0.8661
Epoch 42/80
655/655 [==============================] - 201s 308ms/step - loss: 0.5194 - accuracy: 0.8
226 - val_loss: 0.3921 - val_accuracy: 0.8684
Epoch 43/80
655/655 [==============================] - 201s 307ms/step - loss: 0.5093 - accuracy: 0.8
275 - val_loss: 0.3888 - val_accuracy: 0.8682
Epoch 44/80
655/655 [==============================] - 203s 309ms/step - loss: 0.5136 - accuracy: 0.8
235 - val_loss: 0.3886 - val_accuracy: 0.8688
Epoch 45/80
655/655 [==============================] - 202s 308ms/step - loss: 0.5073 - accuracy: 0.8
279 - val_loss: 0.3877 - val_accuracy: 0.8695
Epoch 46/80
655/655 [==============================] - 201s 307ms/step - loss: 0.5069 - accuracy: 0.8
312 - val_loss: 0.3874 - val_accuracy: 0.8709
Epoch 47/80
655/655 [==============================] - 200s 305ms/step - loss: 0.5057 - accuracy: 0.8
```

```
274 - val_loss: 0.3861 - val_accuracy: 0.8688
Epoch 48/80
655/655 [==============================] - 201s 307ms/step - loss: 0.4974 - accuracy: 0.8
305 - val_loss: 0.3860 - val_accuracy: 0.8688
Epoch 49/80
655/655 [==============================] - 202s 308ms/step - loss: 0.4940 - accuracy: 0.8
304 - val_loss: 0.3831 - val_accuracy: 0.8720
Epoch 50/80
655/655 [==============================] - 202s 308ms/step - loss: 0.4920 - accuracy: 0.8
350 - val_loss: 0.3813 - val_accuracy: 0.8730
Epoch 51/80
655/655 [==============================] - 202s 308ms/step - loss: 0.4915 - accuracy: 0.8
316 - val_loss: 0.3812 - val_accuracy: 0.8730
Epoch 52/80
655/655 [==============================] - 203s 309ms/step - loss: 0.4869 - accuracy: 0.8
365 - val_loss: 0.3794 - val_accuracy: 0.8722
Epoch 53/80
655/655 [==============================] - 203s 311ms/step - loss: 0.4836 - accuracy: 0.8
335 - val_loss: 0.3794 - val_accuracy: 0.8732
Epoch 54/80
655/655 [==============================] - 201s 307ms/step - loss: 0.4792 - accuracy: 0.8
358 - val_loss: 0.3776 - val_accuracy: 0.8730
Epoch 55/80
655/655 [==============================] - 203s 310ms/step - loss: 0.4785 - accuracy: 0.8
393 - val_loss: 0.3775 - val_accuracy: 0.8732
Epoch 56/80
655/655 [==============================] - 203s 309ms/step - loss: 0.4703 - accuracy: 0.8
381 - val_loss: 0.3766 - val_accuracy: 0.8732
Epoch 57/80
655/655 [==============================] - 199s 303ms/step - loss: 0.4773 - accuracy: 0.8
382 - val_loss: 0.3757 - val_accuracy: 0.8743
Epoch 58/80
655/655 [==============================] - 193s 295ms/step - loss: 0.4558 - accuracy: 0.8
432 - val_loss: 0.3730 - val_accuracy: 0.8743
Epoch 59/80
655/655 [==============================] - 189s 289ms/step - loss: 0.4639 - accuracy: 0.8
429 - val_loss: 0.3721 - val_accuracy: 0.8760
Epoch 60/80
655/655 [==============================] - 188s 287ms/step - loss: 0.4678 - accuracy: 0.8
411 - val_loss: 0.3733 - val_accuracy: 0.8755
Epoch 61/80
655/655 [==============================] - 192s 293ms/step - loss: 0.4523 - accuracy: 0.8
476 - val_loss: 0.3719 - val_accuracy: 0.8749
Epoch 62/80
655/655 [==============================] - 191s 291ms/step - loss: 0.4547 - accuracy: 0.8
457 - val_loss: 0.3707 - val_accuracy: 0.8751
Epoch 63/80
655/655 [==============================] - 195s 298ms/step - loss: 0.4480 - accuracy: 0.8
478 - val_loss: 0.3723 - val_accuracy: 0.8762
Epoch 64/80
655/655 [==============================] - 206s 314ms/step - loss: 0.4568 - accuracy: 0.8
440 - val_loss: 0.3723 - val_accuracy: 0.8756
Epoch 65/80
655/655 [==============================] - 205s 312ms/step - loss: 0.4525 - accuracy: 0.8
470 - val_loss: 0.3701 - val_accuracy: 0.8756
Epoch 66/80
655/655 [==============================] - 203s 310ms/step - loss: 0.4472 - accuracy: 0.8
464 - val_loss: 0.3699 - val_accuracy: 0.8760
Epoch 67/80
655/655 [==============================] - 203s 310ms/step - loss: 0.4406 - accuracy: 0.8
508 - val_loss: 0.3682 - val_accuracy: 0.8768
Epoch 68/80
655/655 [==============================] - 194s 297ms/step - loss: 0.4413 - accuracy: 0.8
498 - val_loss: 0.3672 - val_accuracy: 0.8766
Epoch 69/80
655/655 [==============================] - 202s 308ms/step - loss: 0.4359 - accuracy: 0.8
512 - val_loss: 0.3665 - val_accuracy: 0.8772
Epoch 70/80
655/655 [==============================] - 204s 312ms/step - loss: 0.4408 - accuracy: 0.8
503 - val_loss: 0.3646 - val_accuracy: 0.8781
Epoch 71/80
655/655 [==============================] - 192s 293ms/step - loss: 0.4413 - accuracy: 0.8
```

```
519 - val_loss: 0.3673 - val_accuracy: 0.8779
Epoch 72/80
655/655 [==============================] - 190s 290ms/step - loss: 0.4344 - accuracy: 0.8
536 - val_loss: 0.3637 - val_accuracy: 0.8774
Epoch 73/80
655/655 [==============================] - 191s 292ms/step - loss: 0.4299 - accuracy: 0.8
562 - val_loss: 0.3664 - val_accuracy: 0.8777
Epoch 74/80
655/655 [==============================] - 190s 290ms/step - loss: 0.4276 - accuracy: 0.8
551 - val_loss: 0.3636 - val_accuracy: 0.8777
Epoch 75/80
655/655 [==============================] - 191s 292ms/step - loss: 0.4259 - accuracy: 0.8
542 - val_loss: 0.3638 - val_accuracy: 0.8776
Epoch 76/80
655/655 [==============================] - 192s 293ms/step - loss: 0.4234 - accuracy: 0.8
547 - val_loss: 0.3605 - val_accuracy: 0.8791
Epoch 77/80
655/655 [==============================] - 197s 300ms/step - loss: 0.4244 - accuracy: 0.8
557 - val_loss: 0.3626 - val_accuracy: 0.8783
Epoch 78/80
655/655 [==============================] - 196s 299ms/step - loss: 0.4160 - accuracy: 0.8
592 - val_loss: 0.3631 - val_accuracy: 0.8791
Epoch 79/80
655/655 [==============================] - 197s 301ms/step - loss: 0.4132 - accuracy: 0.8
610 - val_loss: 0.3613 - val_accuracy: 0.8810
Epoch 80/80
655/655 [==============================] - 197s 301ms/step - loss: 0.4070 - accuracy: 0.8
618 - val_loss: 0.3598 - val_accuracy: 0.8819
```

In [14]:

```
converter = tf.lite.TFLiteConverter.from_keras_model(model)

def representative_dataset():
  for _ in range(10000):
    yield [np.random.uniform(-1, 1, size=(1, *shape, 3)).astype(np.float32)]


# Set the optimization flag.
converter.optimizations = [tf.lite.Optimize.DEFAULT]
# Enforce integer only quantization
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
# Provide a representative dataset to ensure we quantize correctly.
converter.representative_dataset = representative_dataset
converter.inference_input_type = tf.int8
converter.inference_output_type = tf.int8

model_tflite = converter.convert()



# 将 TensorFlow Lite 模型保存到磁盘中
with open('my_model.tflite', 'wb') as f:
    f.write(model_tflite)
```

```
/opt/conda/lib/python3.7/site-packages/tensorflow/lite/python/convert.py:765: UserWarning
: Statistics for quantized inputs were expected, but not specified; continuing anyway.
  warnings.warn("Statistics for quantized inputs were expected, but not "
fully_quantize: 0, inference_type: 6, input_inference_type: INT8, output_inference_type:
INT8
```

In [15]:

```
interpreter = tf.lite.Interpreter(model_path="my_model.tflite")

interpreter.allocate_tensors()
input_details = interpreter.get_input_details()[0]
output_details = interpreter.get_output_details()[0]
```

```
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
```

In [16]:

```
input_details
```

Out[16]:

```
{'name': 'serving_default_input_2:0',
 'index': 0,
 'shape': array([  1, 128, 128,   3], dtype=int32),
 'shape_signature': array([ -1, 128, 128,   3], dtype=int32),
 'dtype': numpy.int8,
 'quantization': (0.007843137718737125, -1),
 'quantization_parameters': {'scales': array([0.00784314], dtype=float32),
  'zero_points': array([-1], dtype=int32),
  'quantized_dimension': 0},
 'sparsity_parameters': {}}
```

In [17]:

```
output_details
```

Out[17]:

```
{'name': 'StatefulPartitionedCall:0',
 'index': 93,
 'shape': array([ 1, 10], dtype=int32),
 'shape_signature': array([-1, 10], dtype=int32),
 'dtype': numpy.int8,
 'quantization': (0.03112613409757614, 36),
 'quantization_parameters': {'scales': array([0.03112613], dtype=float32),
  'zero_points': array([36], dtype=int32),
  'quantized_dimension': 0},
 'sparsity_parameters': {}}
```

In [18]:

```
img=cv2.imread("/kaggle/input/animals10/raw-img/gallina/10.jpeg")

img = cv2.resize(img,shape)

img = cv2.cvtColor(img,cv2.COLOR_RGB2BGR)
plt.imshow(img)
img = (img.copy().astype("int32")-128).astype("int8")


print(img.shape)
# img = img.astype("float32")/255.0
# img = img.reshape(( 3, 352, 352))
# img = np.transpose(img, (2, 0, 1))
# print(img.shape)


interpreter.set_tensor(input_details["index"], [img])
interpreter.invoke()
result = interpreter.get_tensor(output_details["index"])
print(result)
m = np.argmax(result)
labels[m]
```

```
(128, 128, 3)
[[ -19  -28 -128 -128  127 -128 -128   61 -128 -128]]
```

Out[18]:

'鸡'

In [ ]: