Question 2

In [ ]:
```
# Checking internet is enabled
import socket
try:
    socket.setdefaulttimeout(1)
    socket.socket(socket.AF_INET, socket.SOCK_STREAM).connect(('1.1.1.1', 53))
except socket.error as ex:
    raise Exception("No internet.")
```

In [ ]:
```
# Ensuring the latest version of any required libraries
import os
%pip install -Uqq fastai
```

Downloading Images of Animals

In [ ]:
```
# Installing duckduckgo_search
%pip install -Uqq duckduckgo_search
```

In [ ]:
```
from duckduckgo_search import ddg_images
from fastcore.all import *
from fastai.vision.all import *

def search_images(term, max_images=250):
    return L(ddg_images(term, max_results=max_images)).itemgot('image')
```

Downloading and displaying dog URL and image (for demonstration):

In [ ]:
```
urls = search_images('dog photos', max_images=1)
urls[0]
```

In [ ]:
```
from fastdownload import download_url
dest = 'dog.jpg'
download_url(urls[0], dest, show_progress=False)

from fastai.vision.all import *
im = Image.open(dest)
im.to_thumb(256, 256)
```

Downloading and displaying lion image:

In [ ]:
```
download_url(search_images('lion photos', max_images=1)[0], 'lion.jpg', show_progress=False)
Image.open('lion.jpg').to_thumb(256, 256)
```

Downloading and displaying whale image:

In [ ]:
```
download_url(search_images('whale photos', max_images=1)[0], 'whale.jpg', show_progress=False)
Image.open('whale.jpg').to_thumb(256, 256)
```

Retrieving and storing photo examples for each of the 10 chosen animals (dog, cat, lion, turtle, bear, giraffe, panda, whale, goat and chicken), with each group of photos saved to a different folder:

In [ ]:
```
searches = 'dog','cat','lion','turtle','flamingo','giraffe','panda','whale','goat','chicken'
path = Path('animals')
from time import sleep

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

Removing PNG images:

In [ ]:
```
searches = 'dog','cat','lion','turtle','flamingo','giraffe','panda','whale','goat','chicken'
path = Path('animals')

for animal in searches:
    for file in os.listdir(path/animal):
        if file.endswith('.png'):
             os.remove(path/animal/file)
    resize_images(path/animal, max_size=400, dest=path/animal)
```

Training the Model

Removing any photos that did not download correctly:

In [ ]:
```
failed = verify_images(get_image_files(path))
failed.map(Path.unlink)
len(failed)
```

Creating `DataLoaders` and viewing batch:

In [ ]:
```
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)

dls.show_batch(max_n=20)
```

Training and fine-tuning model:

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

Using/Checking the Model

Image classification of original dog, lion and whale photos:

In [ ]:
```
is_animal,_,probs = learn.predict(PILImage.create('dog.jpg'))
print(f"This is a: {is_animal}.")
print(f"Probability it's a dog: {probs[2]:.4f}")
```

In [ ]:
```
is_animal,_,probs = learn.predict(PILImage.create('lion.jpg'))
print(f"This is a: {is_animal}.")
print(f"Probability it's a lion: {probs[6]:.4f}")
```

In [ ]:
```
is_animal,_,probs = learn.predict(PILImage.create('whale.jpg'))
print(f"This is a: {is_animal}.")
print(f"Probability it's a whale: {probs[9]:.4f}")
```

Displaying the confusion matrix:

In [ ]:
```
interp = ClassificationInterpretation.from_learner(learn)
interp.plot_confusion_matrix(figsize=(12,12), dpi=60)
```

Printing cells of the confusion matrix with the most incorrect predictions:

In [ ]:
```
interp.most_confused(min_val=1)
```

Displaying top losses:

In [ ]:
```
interp.plot_top_losses(12, nrows=3)
```

Visualising data using t-SNE:

In [ ]:
```
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt

# TSNE embedding with 2 dimensions
tsne = TSNE(n_components=2)

# Storing image classifications
animals = ['dog','cat','lion','turtle','flamingo','giraffe','panda','whale','goat','chicken']
animal_classifications = []
for path in dls.valid_ds.items:
    animal_classifications.append(animals.index(str(path).split("/")[1]))

# Transforming classification predictions to the embedded space
predictions = learn.get_preds(ds_idx=1)[0]
tsne_result = tsne.fit_transform(predictions)

# Plotting embeddings
plt.scatter(tsne_result[:,0], tsne_result[:,1], c=animal_classifications, cmap="tab10", s=5)
plt.colorbar()
plt.title("t-SNE Graph")
```

Installing nbconvert for exporting notebook:

In [ ]:

```
%pip install nbconvert
```