Question 2

In [ ]:
```
#NB: Kaggle requires phone verification to use the internet or a GPU. If you hav
#    This code is only here to check that your internet is enabled. It doesn't a
#    Here's a help thread on getting your phone number verified: https://www.kag

import socket,warnings
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("STOP: No internet. Click '>|' in top
```

In [ ]:
```
# It's a good idea to ensure you're running the latest version of any libraries
# `!pip install -Uqq <libraries>` upgrades to the latest version of <libraries>
# NB: You can safely ignore any warnings or errors pip spits out about running a
import os
iskaggle = os.environ.get('KAGGLE_KERNEL_RUN_TYPE', '')

if iskaggle:
    !pip install -Uqq fastai
```

Step 1: Download images of 10 different animals

Install necessary packages for the project

In [ ]:
```
# Skip this cell if you already have duckduckgo_search installed
!pip install -Uqq duckduckgo_search
!pip install -Uqq scikit-learn matplotlib
!pip install fastbook
```

Import all the libraries

In [ ]:
```
from duckduckgo_search import ddg_images
from fastcore.all import *
from fastbook import search_images_ddg
from fastai.vision.all import *
from fastai.vision.widgets import *
from fastdownload import download_url

from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
```

Using Duck Duck Go get 200 images of 10 animals (in total 2000) from the internet, then save them in categorised folders of each animal within the folder 'training'.

In [ ]:
```
#I got help from ChatGPT 3.5 to generate the comments for this code:
# List of animal names to search for
searches = ['dog', 'monkey', 'lion', 'cat', 'elephant', 'giraffe', 'panda', 'bir
# Path to the training directory
path = Path('training')
# Importing the sleep function from the time module
from time import sleep
```

```
# Function to resize images in a directory
def resize_images(p:Path, max_size=800, dest=Path('.'), exts = ['.jpg', '.jpeg',
    dest.mkdir(parents=True, exist_ok=True)  # Create the destination directory
    files = get_image_files(p)  # Get the list of image files in the directory
    files = [file for file in files if file.suffix.lower() in exts]  # Only take
    for file in files:
        try:
            img = Image.open(file)  # Open the image file
            img = img.resize((max_size, max_size), Image.ANTIALIAS)  # Resize th
            img.save(dest/file.name)  # Save the resized image to the destinatio
        except Exception as e:
            print(f"Could not resize image {file.name}: {e}")  # Handle and prin

# Iterate over each animal name in the searches list
for o in searches:
    dest = (path/o)  # Create a directory path for the current animal
    dest.mkdir(exist_ok=True, parents=True)  # Create the animal directory if it
    download_images(dest, urls=search_images_ddg(f'{o} animal'))  # Download ima
    sleep(10)  # Pause for 10 seconds to avoid overloading the server
    resize_images(o, max_size=400, dest=path/o)  # Call the resize_images functi
```

Remove images that did not download correctly.

In [ ]:
```
# Define the path to the training directory
path = Path('training')

# Verify images in the training directory
failed = verify_images(get_image_files(path))

# Remove the failed images
failed.map(Path.unlink)

# Get the count of failed images
num_failed = len(failed)
```

Out[ ]: 52

Create a new folder called 'test', and move 20 images of each animal into it for our test dataset.

In [ ]:
```
#I got help from ChatGPT 3.5 to generate the following code:
import shutil

# Define the path to the training directory
path = Path('training')

# Create the test directory path
test_path = path.parent/'test'
test_path.mkdir(exist_ok=True)

# Iterate over each animal in the searches list
for o in searches:
    # Get the list of images for the current animal in the training directory
    images = get_image_files(path/o)

    # Create the destination directory for the current animal in the test direct
    test_dest = test_path/o
    test_dest.mkdir(exist_ok=True)
```

```
    # Move the first 20 images from the training directory to the test directory
    for i in range(20):
        shutil.move(images[i], test_dest/images[i].name)
```

Step 2: Train our model

Count the number of pictures of each animal in the 'training' set:

In [ ]:
```
# Define the path to the training directory
path = Path('training')

# Define the list of animal search terms
searches = ['dog', 'monkey', 'lion', 'cat', 'elephant', 'giraffe', 'panda', 'bir

# Iterate over each search term
for search_term in searches:
    # Create the path to the folder for the current search term
    path_to_folder = path/search_term

    # Get the number of images in the folder for the current search term
    num_imgs = len(get_image_files(path_to_folder))

    # Print the number of images for the current search term
    print(f'Number of {search_term} images: {num_imgs}')

# Get the total number of images in the training directory
total_imgs = len(get_image_files(path))

# Print the total number of images
print(f'Total number of images: {total_imgs}')
```

```
Number of dog images: 94
Number of monkey images: 147
Number of lion images: 148
Number of cat images: 112
Number of elephant images: 147
Number of giraffe images: 160
Number of panda images: 160
Number of bird images: 98
Number of penguin images: 159
Number of crocodile images: 159
Total number of images: 1384
```

Count the number of pictures of each animal in the 'test' set:

In [ ]:
```
# Define the path to the test directory
path = Path('test')

# Define the list of animal search terms
searches = ['dog', 'monkey', 'lion', 'cat', 'elephant', 'giraffe', 'panda', 'bir

# Iterate over each search term
for search_term in searches:
    # Create the path to the folder for the current search term
    path_to_folder = path/search_term

    # Get the number of images in the folder for the current search term
```

```
    num_imgs = len(get_image_files(path_to_folder))

    # Print the number of images for the current search term
    print(f'Number of {search_term} images: {num_imgs}')

# Get the total number of images in the test directory
total_imgs = len(get_image_files(path))

# Print the total number of images
print(f'Total number of images: {total_imgs}')
```

```
Number of dog images: 20
Number of monkey images: 20
Number of lion images: 20
Number of cat images: 20
Number of elephant images: 20
Number of giraffe images: 20
Number of panda images: 20
Number of bird images: 20
Number of penguin images: 20
Number of crocodile images: 20
Total number of images: 200
```

To train a model, we'll need `DataLoaders`, which is an object that contains a *training set* (the images used to create a model) and a *validation set* (the images used to check the accuracy of a model -- not used during training). In `fastai` we can create that easily using a `DataBlock`, and view sample images from it.

We setup the data loading and preprocessing for the training set by defining the blocks, image retrieval, splitter, label extraction and item transforms. The data loader is made.

In [ ]:
```
# Define the path to the training directory
path = Path('training')

# Create a DataBlock for loading and preprocessing the data
dls = DataBlock(
    # Specify the blocks for the inputs (images) and targets (categories)
    blocks=(ImageBlock, CategoryBlock),

    # Function to get the image files
    get_items=get_image_files,

    # Splitter to randomly split the data into training and validation sets
    splitter=RandomSplitter(valid_pct=0.2, seed=42),

    # Function to get the labels (categories) from the parent directory of each
    get_y=parent_label,

    # Item transformations to resize the images
    item_tfms=[Resize(192, method='squish')]
).dataloaders(path)



#dls.show_batch(max_n=10)
```

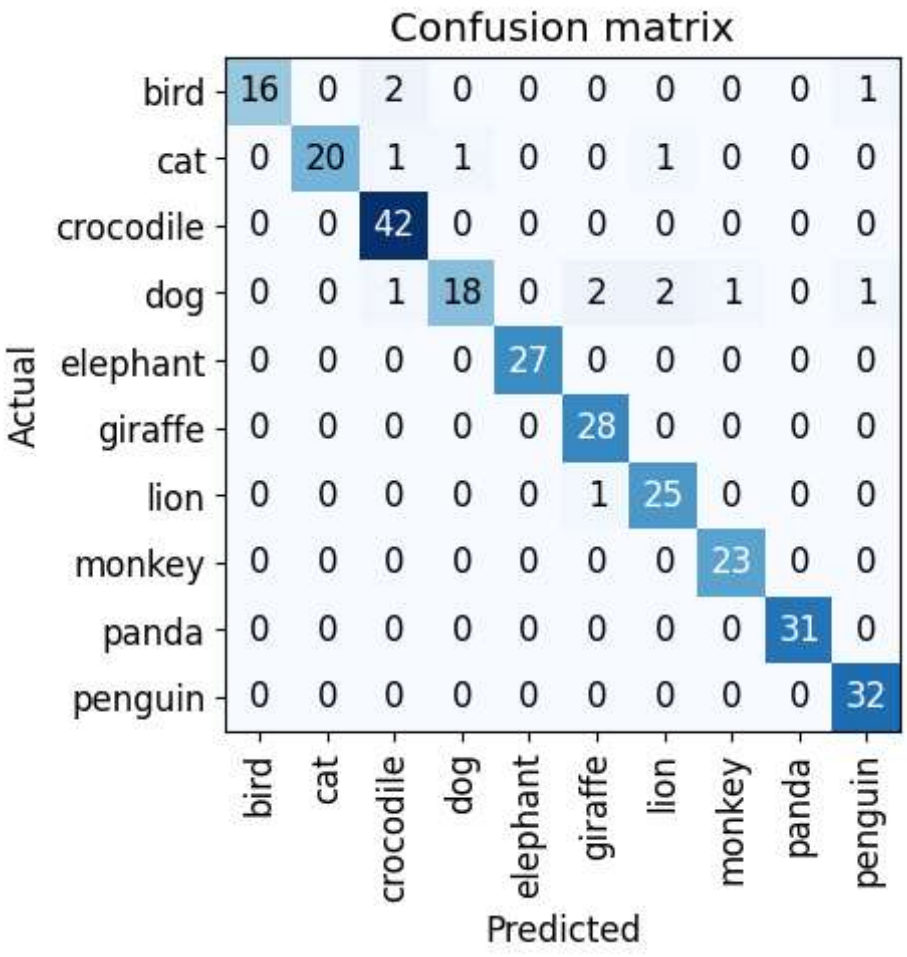
Now we're ready to train our model. The fastest widely used computer vision model is `resnet18`. Y

`fastai` comes with a helpful `fine_tune()` method which automatically uses best practices for fine tuning a pre-trained model, so we'll use that. In our case since we are using 10 sets of images the best loss function which would be automatically used is CrossEntropyLossFlat, for it's suitablitity of measuring dissimilarity between predicted probabilities and true labels. We can optimise the predictions for multiclass tasks and learn the correct class probability for each image.

In [ ]:
```
# Create a vision learner using the provided data loaders, model architecture (r
# and metric (error_rate)
learn = vision_learner(dls, resnet18, metrics=error_rate)

# Fine-tune the model for 3 epochs
learn.fine_tune(3)
```

Now we make the Confusion Matrix, to evaluate model's performance for the training dataset. This allows us to get a summary of the model's predictions compared to the true labels.

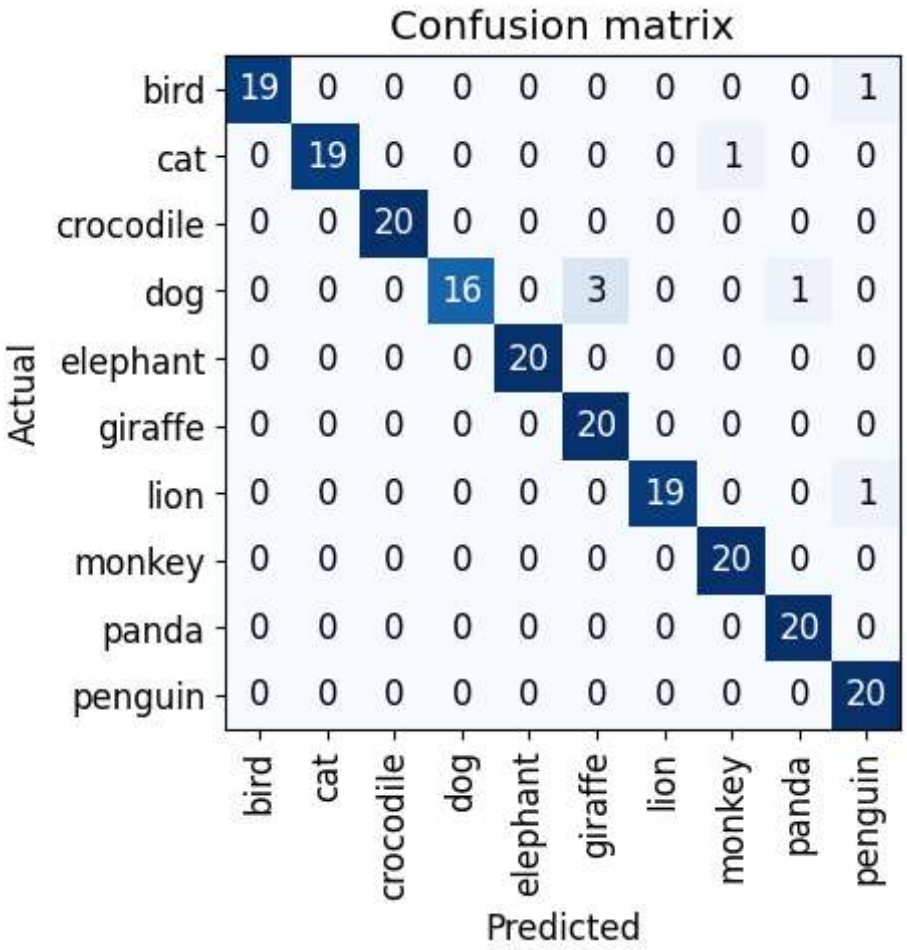
In [ ]:
```
# Create a ClassificationInterpretation object from the trained learner
interp = ClassificationInterpretation.from_learner(learn)

# Plot the confusion matrix for the training set
interp.plot_confusion_matrix()
```



Step 3 - Apply our model on the test dataset

We now apply the trained model to the test dataset, getting the predictions and targets, then assign them to 'ClassificationInterpretation'. Then we plot the confusion matrix for the model's performance on the test set. We can see that our model did better on the test dataset after being trained on the training dataset.

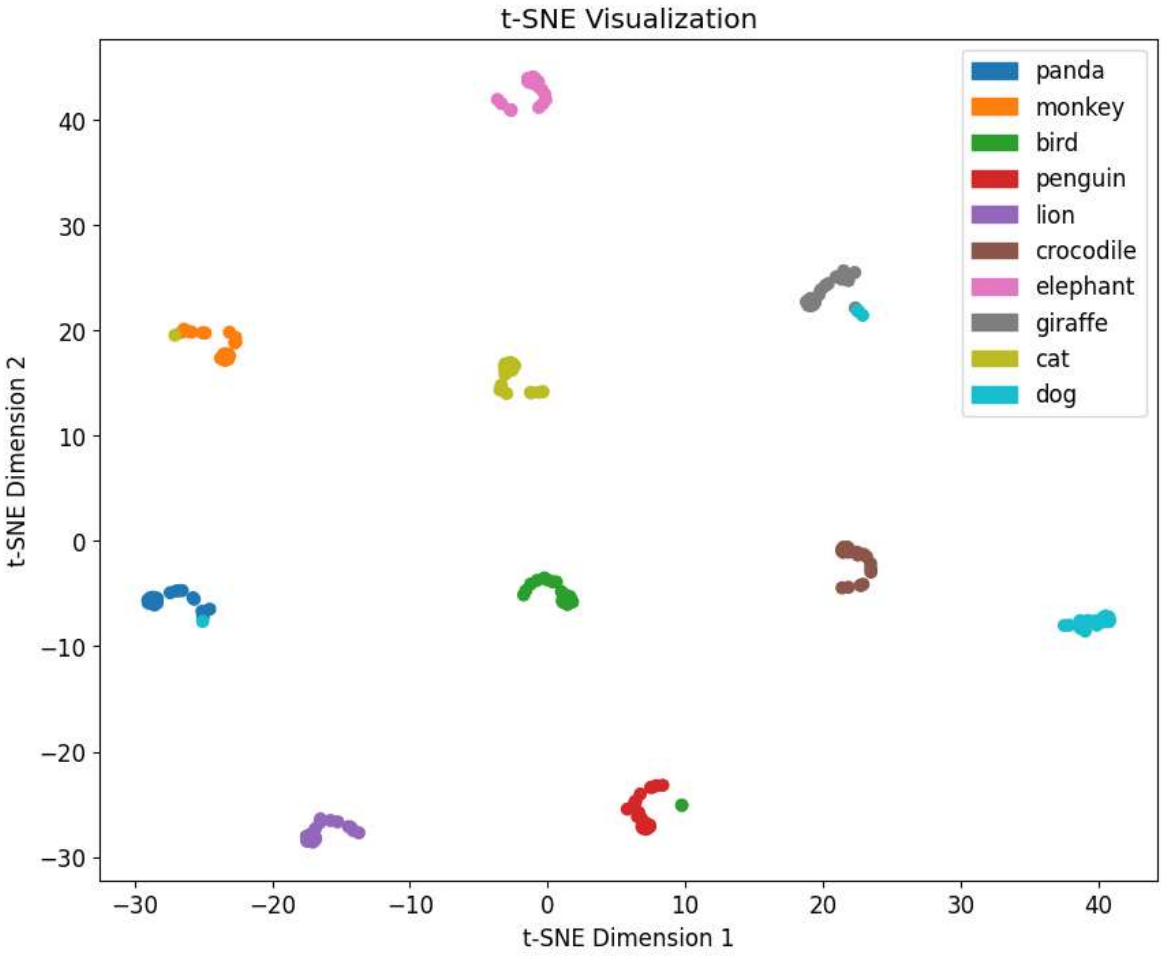
In [ ]:
```
#I got help from ChatGPT 3.5 to generate the following code:
# Define the path to the test directory
test_path = path.parent/'test'

# Create a test dataloader using the trained learner and the image files from th
test_dl = learn.dls.test_dl(get_image_files(test_path), with_labels=True)

# Create a ClassificationInterpretation object from the trained learner and the
interp = ClassificationInterpretation.from_learner(learn, dl=test_dl)

# Get the predictions and targets from the trained learner using the test datalo
preds, targets = learn.get_preds(dl=test_dl)

# Get the predicted labels by selecting the class with the highest probability
pred_labels = preds.argmax(dim=1)

# Assign the predicted labels and targets to the ClassificationInterpretation ob
interp.pred_class = pred_labels
interp.y_true = targets

# Plot the confusion matrix for the test set
interp.plot_confusion_matrix()
```



Step 4: t-SNE

We want to visualise how well our model seperates and groups different images into their classes. We do this through making a t-SNE plot. The following code gets the t-SNE embeddings for the predicted probabilities, assings colours to each class, and then plots the t-SNE. In the plot below, we can see our model was fairly effective at seperating and grouping the images into their correct animal classes.

In [ ]:
```
#I got help from ChatGPT 3.5 to generate the following code:
import numpy as np
from sklearn.manifold import TSNE
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches

# Create test DataLoader
test_dl = learn.dls.test_dl(get_image_files(test_path), with_labels=True)

# Get predictions
preds, targets = learn.get_preds(dl=test_dl)
pred_labels = preds.argmax(dim=1)

# Use the predictions for the TSNE
embedding = TSNE(perplexity=10).fit_transform(preds)

# Get the names of the labels corresponding to the targets
labels = [dls.vocab[t] for t in targets]

# Create a dictionary mapping unique labels to distinct colors
label_colour_dict = {label: plt.cm.tab10(i/len(set(labels))) for i, label in enu

# Map each label to a color
label_colours = [label_colour_dict[label] for label in labels]

plt.figure(figsize=(10, 8))
# Color the scatter plot points by their labels
plt.scatter(embedding[:, 0], embedding[:, 1], c=label_colours)
plt.title("t-SNE Visualization")
plt.xlabel("t-SNE Dimension 1")
plt.ylabel("t-SNE Dimension 2")

# Add a legend for the labels
handles = [mpatches.Patch(color=colour, label=label) for label, colour in label_
plt.legend(handles=handles, loc='upper right')

plt.show()
```



Now that we applied our trained model to the test dataset, optionally we can look at the images with the largest losses.

In [ ]:
```
interp.plot_top_losses(5, nrows=1, figsize=(17,4))
```

We can then run the cleaner and remove or reclassify the images that had large losses. This further trains our model, but requires manual input.

In [ ]:
```
#cleaning
cleaner = ImageClassifierCleaner(learn)
cleaner
```