Q4

April 24, 2024

ELEC4630 2024 Sem1 A2 Q4

Based on FastAI course lesson 00-is-it-a-bird-creating-a-model-from-your-own-data.

Ethan Waugh - 46416458

[ ]:
```
#NB: Kaggle requires phone verification to use the internet or a GPU. If you haven't done that yet, the cell below will fail
#    This code is only here to check that your internet is enabled. It doesn't do anything else.
#    Here's a help thread on getting your phone number verified: https://www.kaggle.com/product-feedback/135367

import socket,warnings
try:
    socket.setdefaulttimeout(1)
    socket.socket(socket.AF_INET, socket.SOCK_DGRAM).connect(('1.1.1.1', 53))
except socket.error as ex: raise Exception("STOP: No internet. Click '>|' in top right and set 'Internet' switch to on")
```

[ ]:
```
# It's a good idea to ensure you're running the latest version of any libraries you need.
# `!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 as root or incompatibilities
import os
iskaggle = os.environ.get('KAGGLE_KERNEL_RUN_TYPE', '')

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

0.1 Download Images

[ ]:
```
# Skip this cell if you already have duckduckgo_search installed
!pip install -Uqq duckduckgo_search
```

[ ]:
```
from duckduckgo_search import DDGS
from fastcore.all import *
```

```
ddgs = DDGS()
def search_images(term, max_images=200): return L(ddgs.images(term,␣
 ↪max_results=max_images)).itemgot('image')
```

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

[ ]:
```
'https://wallpaperaccess.com/full/850490.jpg'
```

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

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

[ ]:

Do the same for the other classes of CIFAR10 (automobile, bird, cat, deer, dog, frog, horse, ship, truck)

[ ]:
```
CIFAR10_classes = ['airplane', 'automobile', 'bird', 'cat', 'deer', 'dog',␣
 ↪'frog', 'horse', 'ship', 'truck']
```

[ ]:
```
for c in CIFAR10_classes:
    download_url(search_images(f'{c} photos', max_images=3)[2], f'{c}.jpg',␣
 ↪show_progress=False)
    Image.open(f'{c}.jpg').to_thumb(256,256)
    print(f'{c} class complete')
```

```
airplane class complete
automobile class complete
bird class complete
cat class complete
```

```
deer class complete
dog class complete
frog class complete
horse class complete
ship class complete
truck class complete
```

Can check the results which should be saved to the same directory as the codes directory. Results look good for this instance so load classes into different folders

[ ]:
```
searches = CIFAR10_classes
path = Path('a2_q4_classes')
```

[ ]:
```
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)
```

```
/home/vscode/.local/lib/python3.10/site-packages/PIL/Image.py:975: UserWarning:
Palette images with Transparency expressed in bytes should be converted to RGBA
images
  warnings.warn(
/home/vscode/.local/lib/python3.10/site-packages/PIL/Image.py:975: UserWarning:
Palette images with Transparency expressed in bytes should be converted to RGBA
images
  warnings.warn(
/home/vscode/.local/lib/python3.10/site-packages/PIL/Image.py:975: UserWarning:
Palette images with Transparency expressed in bytes should be converted to RGBA
images
  warnings.warn(
/home/vscode/.local/lib/python3.10/site-packages/PIL/Image.py:975: UserWarning:
Palette images with Transparency expressed in bytes should be converted to RGBA
images
  warnings.warn(
```

Check for dodgy downloaded photos and remove them

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

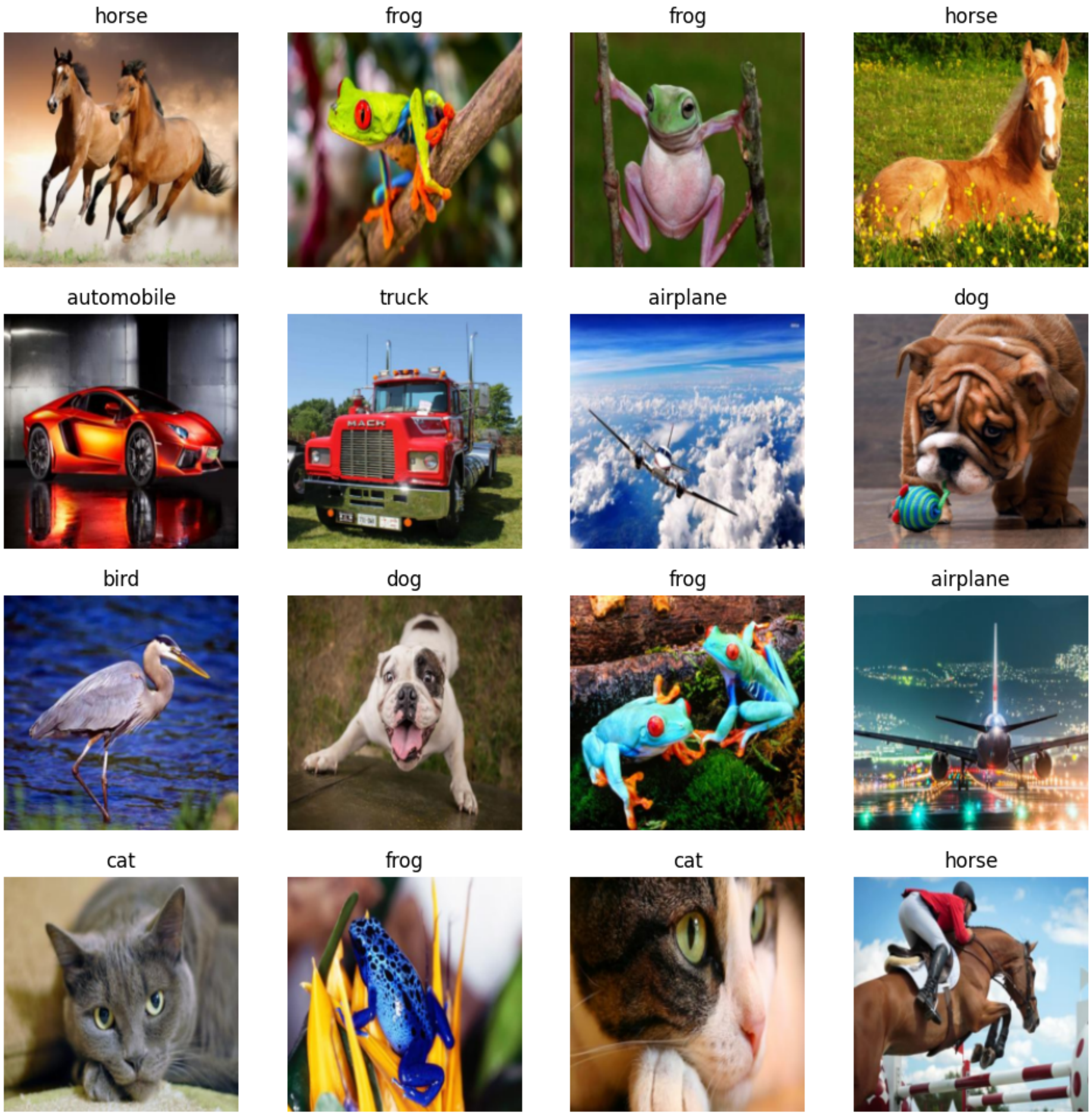
```
[ ]: 34
```

0.2 Train Model

Create a dataloader and then train a model. We'll examine a random batch just to check the labelling and images are accurate

```
[ ]: 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=16)
```



They all look fine so we can go ahead and train. Here I use FastAI's adaptation of torch.nn.CrossEntropyLoss, CrossEntropyLossFlat. This is essentially the same but 'flattens input and target'. Cross Entropy Loss functions by calculating the difference between the true probability distribution of target labels vs the distribution that the model predicted. The more confident an incorrect prediction by the model, the harsher it is penalised in backpropagation - e.g. if the true label is a dog and the model predicts 100% cat, the parameters will be heavily adjusted to optimise the loss function. This is great for multiclass classification as it functions perfectly with input to its formula being a probability distribution over several classes.

```
[ ]: learn = vision_learner(dls, resnet18, metrics=error_rate, loss_func=CrossEntropyLossFlat())
     learn.fine_tune(5)
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

I typically end up with error rates of 0.01227 in the end

0.3 Test on unseen pictures

Can use the section below for loading images and testing the model's predictions. Obviously the classification are limited to the CIFAR10 classes. I had some fun testing cartoon animals/pictures to see if model can recognise any of their features. All the predictions seem reasonable meaning we have a seemingly well trained model.

[ ]:

```
download_url(search_images('kermit', max_images=1)[0], 'test.jpg',␣
    ↪show_progress=False)
Image.open('test.jpg').to_thumb(256,256)
```

[ ]:

[ ]:

```
# Predict on unseen data and show probability distribution
category,_,probs = learn.predict(PILImage.create('test.jpg'))
print(f"This is a {category}.\n")
print(f"Probabilities:")
for i in range(len(CIFAR10_classes)):
    print(f"    {CIFAR10_classes[i]}: {probs[i]:.4f}")
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

This is a frog.

Probabilities:
   airplane: 0.0000
   automobile: 0.0009
   bird: 0.0007
   cat: 0.0002
   deer: 0.0006
   dog: 0.0001
   frog: 0.9974
   horse: 0.0000
   ship: 0.0001
   truck: 0.0000
```

0.4 Analyse results with t-SNE and Confusion Matrices

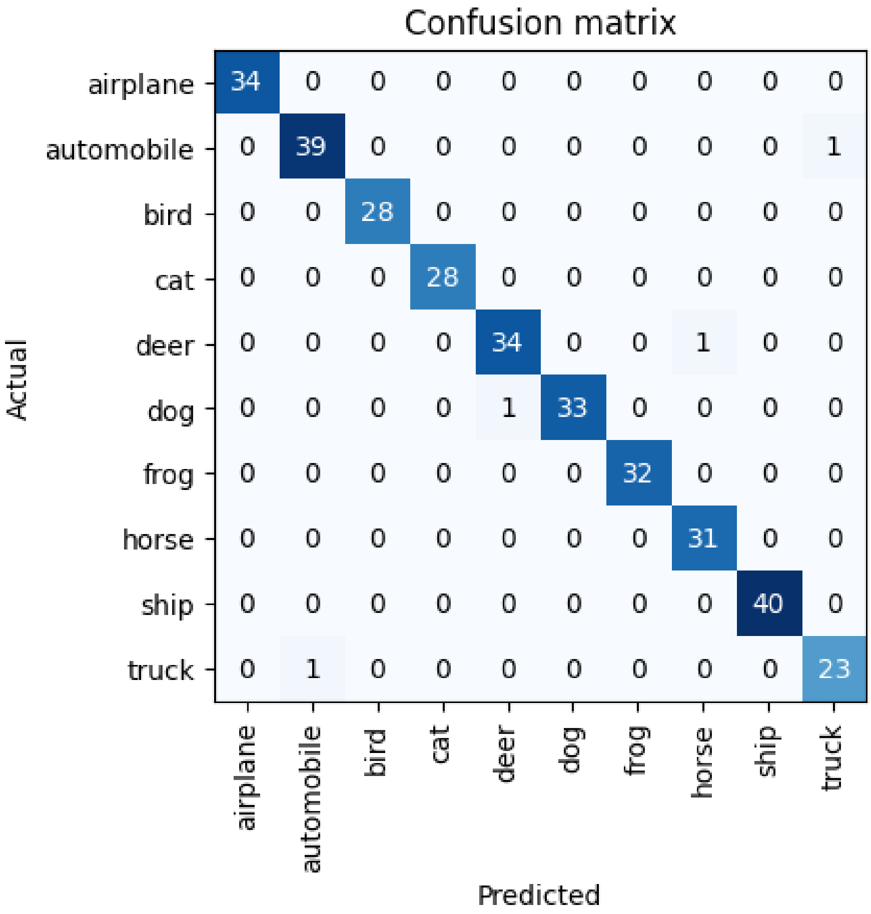
```
[ ]: # Make interpretation for confusion matrix and plot
     interp = ClassificationInterpretation.from_learner(learn)
     interp.plot_confusion_matrix()
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```



Confusion matrix

Actual \ Predicted	airplane	automobile	bird	cat	deer	dog	frog	horse	ship	truck
airplane	34	0	0	0	0	0	0	0	0	0
automobile	0	39	0	0	0	0	0	0	0	1
bird	0	0	28	0	0	0	0	0	0	0
cat	0	0	0	28	0	0	0	0	0	0
deer	0	0	0	0	34	0	0	1	0	0
dog	0	0	0	0	1	33	0	0	0	0
frog	0	0	0	0	0	0	32	0	0	0
horse	0	0	0	0	0	0	0	31	0	0
ship	0	0	0	0	0	0	0	0	40	0
truck	0	1	0	0	0	0	0	0	0	23

The confusion matrix shows that the majority of validation data is correctly classified, but there are some misclassifications - automobile and truck had misclassifications each way, a dog was misclassified as a deer and a deer misclassified as a horse. These make intuitive sense as these classes look similar in actuality. Overall, the model has performed how it is supposed to thus far.

t-SNE transforms the high dimensional feature space to two-dimensions suitable for plotting. It does this by calculating the similarities of points in the high dimensional space and comparing these similarities to the predicted points in the two-dimensional space - it is expected that similar points in high dimensions should be demonstrated as near neighbours in two dimensions. This is then repeated by moving the two-dimensional points around and comparing it with the calculated high dimension similarities until the representation is optimised. For this model and data, we'd expect clear distinguishing clusters for the separate groups with some potential overlapping between the groups with similar features like a automobile and truck. Let's see how effective it is.

[ ]:

```python
from sklearn.manifold import TSNE

# Get features and perform t-SNE
preds, targets = learn.get_preds()
features = preds.numpy()
```

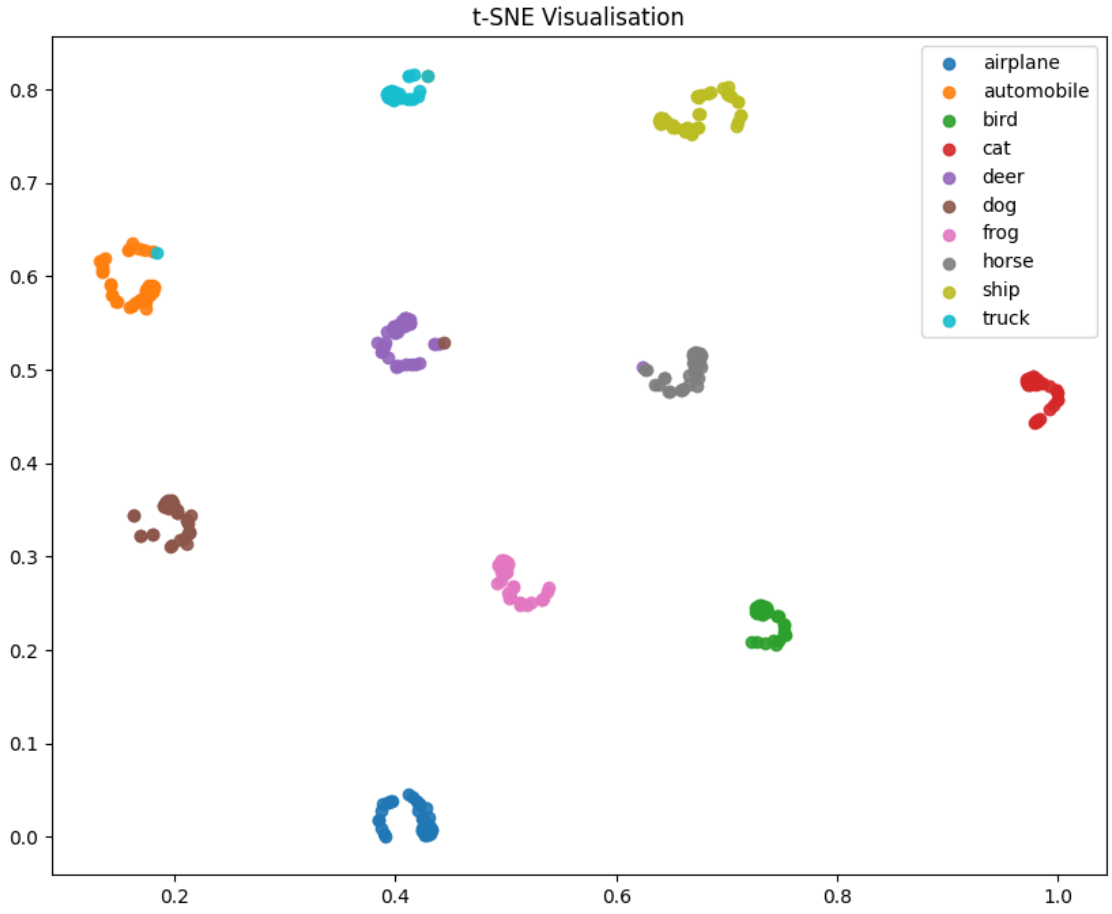
```python
tsne = TSNE(n_components=2, init='pca', perplexity=15, early_exaggeration=12).fit_transform(features)
tsne_min = np.min(tsne)
tsne_max = np.max(tsne)
tsne_norm = (tsne - tsne_min)/(tsne_max - tsne_min)

# Plot data on scatter
unique_labels = np.unique(targets)
plt.figure(figsize=(10,8))
for label in unique_labels:
    indices = np.where(targets == label)[0]
    plt.scatter(tsne_norm[indices, 0], tsne_norm[indices, 1], label=f'{CIFAR10_classes[label]}', alpha=0.9)

plt.legend()
plt.title('t-SNE Visualisation')
plt.show()
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```



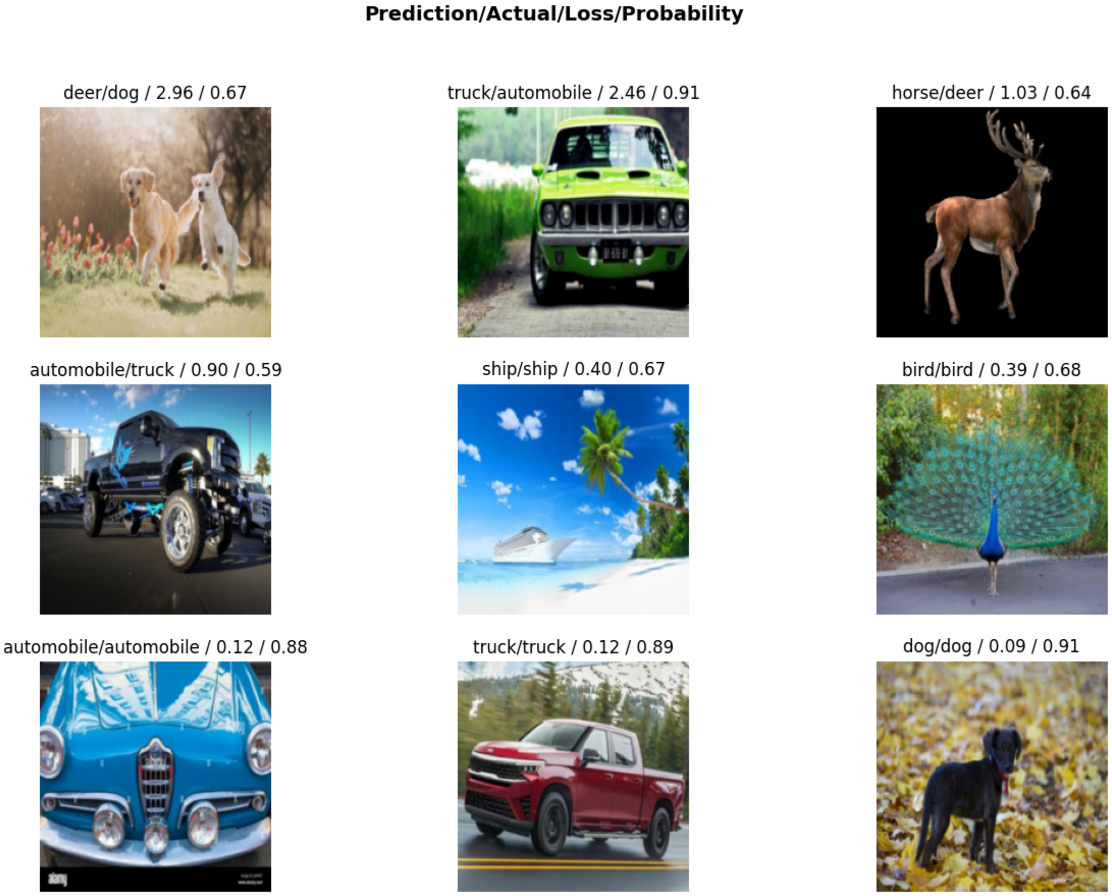
As seen above, the separate classes are very distinguishable in the t-SNE representation and the misclassifications are clearly seen too. The only potential concern is all of the misclassifications being so deeply nestled into the cluster of its misclassified class. There is a potential that these specific images were incorrectly labelled to begin with since the label is based on web search, so this is something to test and see.

[ ]:

```
interp.plot_top_losses(9, figsize=(15,10))
```

```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```


Prediction/Actual/Loss/Probability
deer/dog / 2.96 / 0.67
truck/automobile / 2.46 / 0.91
horse/deer / 1.03 / 0.64
automobile/truck / 0.90 / 0.59
ship/ship / 0.40 / 0.67
bird/bird / 0.39 / 0.68
automobile/automobile / 0.12 / 0.88
truck/truck / 0.12 / 0.89
dog/dog / 0.09 / 0.91

So the search labels worked as intended (although its a bit of a stretch calling one ute a truck and the other an automobile) and the loss function shows here which model predictions causes the most loss. These misclassifications were indeed model based rather than incorrect labels.