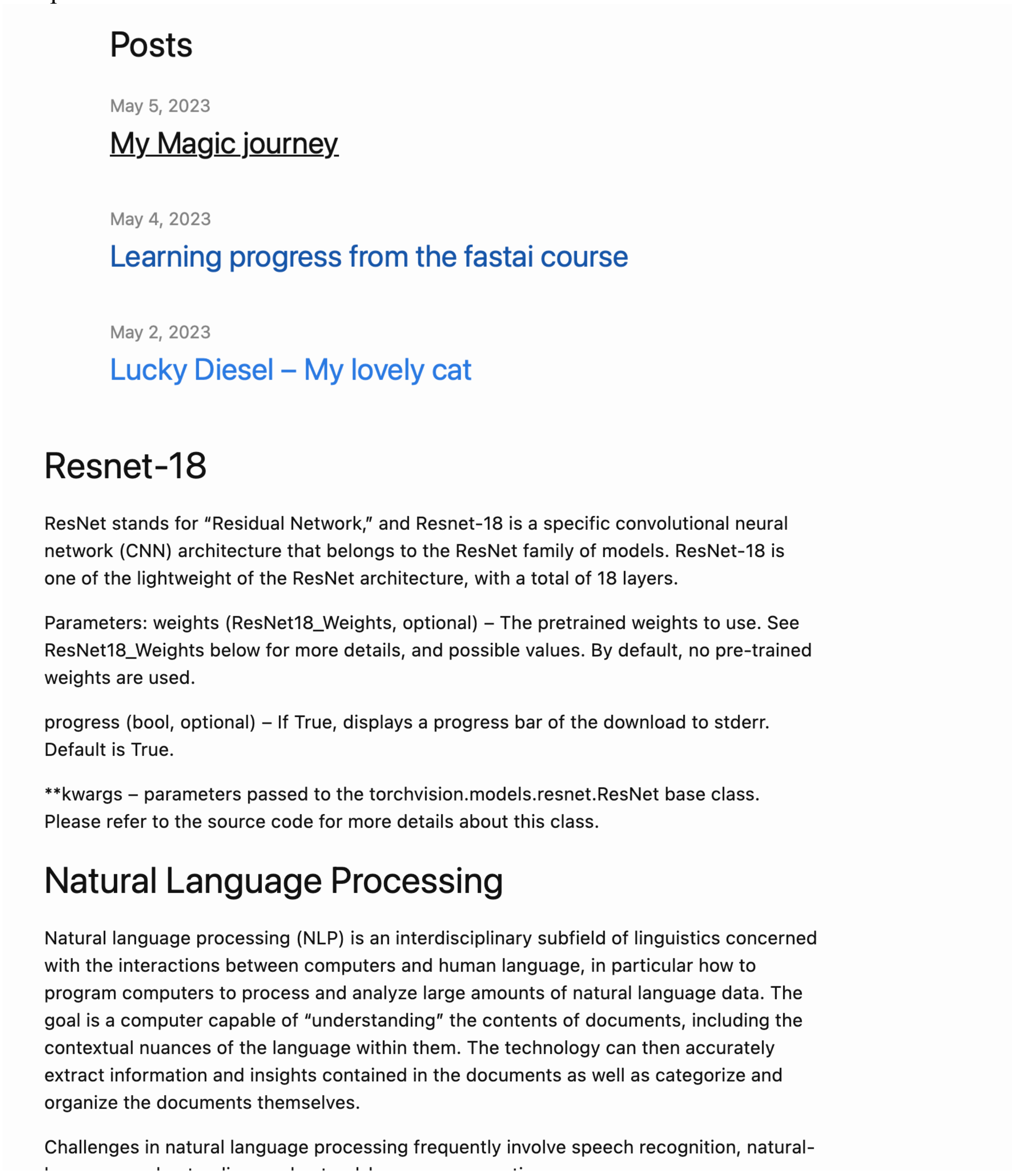
4630 A3

Kuang Sheng

Q1.

666sk.github.io

Sample Posts:



Posts

May 5, 2023

My Magic journey

May 4, 2023

Learning progress from the fastai course

May 2, 2023

Lucky Diesel – My lovely cat

Resnet-18

ResNet stands for "Residual Network," and Resnet-18 is a specific convolutional neural network (CNN) architecture that belongs to the ResNet family of models. ResNet-18 is one of the lightweight of the ResNet architecture, with a total of 18 layers.

Parameters: weights (ResNet18_Weights, optional) – The pretrained weights to use. See ResNet18_Weights below for more details, and possible values. By default, no pre-trained weights are used.

progress (bool, optional) – If True, displays a progress bar of the download to stderr. Default is True.

**kwargs – parameters passed to the torchvision.models.resnet.ResNet base class. Please refer to the source code for more details about this class.

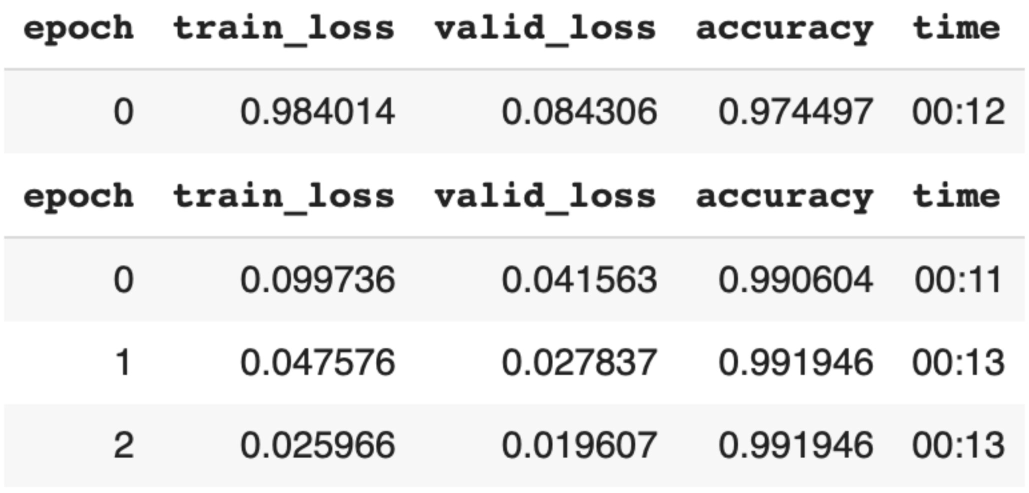
Natural Language Processing

Natural language processing (NLP) is an interdisciplinary subfield of linguistics concerned with the interactions between computers and human language, in particular how to program computers to process and analyze large amounts of natural language data. The goal is a computer capable of "understanding" the contents of documents, including the contextual nuances of the language within them. The technology can then accurately extract information and insights contained in the documents as well as categorize and organize the documents themselves.

Challenges in natural language processing frequently involve speech recognition, natural-

Q2.
See attached PDF below.

Performance of Resnet18:



epoch	train_loss	valid_loss	accuracy	time
0	0.984014	0.084306	0.974497	00:12

epoch	train_loss	valid_loss	accuracy	time
0	0.099736	0.041563	0.990604	00:11
1	0.047576	0.027837	0.991946	00:13
2	0.025966	0.019607	0.991946	00:13

Q3.
Introduction

With the development of AI techniques, colourful images can be generated artfully which is difficult for human eyes to distinguish. As a result, Kaggle held a competition to reduce concerns of authenticity. To effectively distinguish the real images and AI-generated fake images, deep learning can be used to learn the features from the images through appropriate training process.

Literature Review

Sabitha et al. [1] introduced a novel approach called Enhanced Model for Fake Image Detection (EMFID) that aims to classify images as real or AI-generated. The approach implemented Histogram-based Feature Extraction, Discrete Wavelet Transform-based Feature Extraction, and Image Classification with Convolutional Neural Networks (CNN). The proposed model demonstrated fair accuracy and model efficiency when tested on a dataset containing real and fake images. The study also proposed the detection of image alterations using splicing methods.

Jiachen et al. [2] highlighted the urgency of detecting fake images, particularly in the context of Deepfake technology based on Generative Adversarial Networks (GANs). They proposed leveraging texture differences between real and fake images, which is specially effective in face detection. The approach implemented image saliency and guided filters to amplify these texture differences, and a ResNet18 classification network was employed for real and fake detection. The experimental result demonstrated good accuracy.

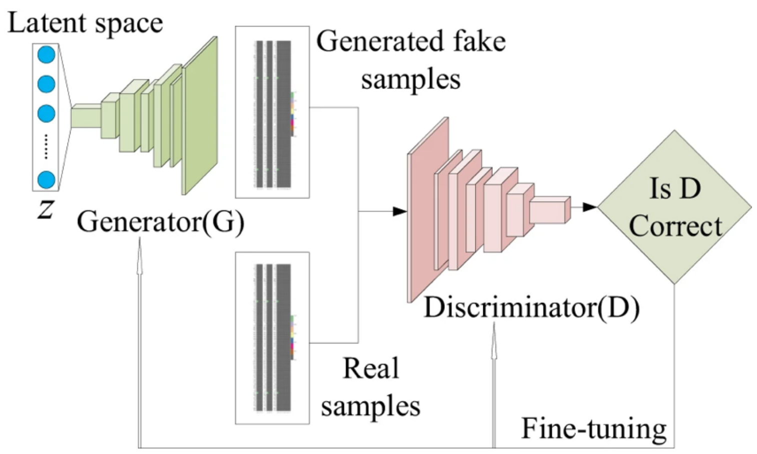


Figure 1: Architecture of GAN [3]

Moreover, Khalaid et al. [4] provided an overview of the strategies used for detecting fake colorized images which mainly focused on machine learning and deep learning, aims to help researchers understand the benefits and drawbacks of existing technologies.

Methodology

Please see attached PDF below.

Works Cited

[1] R. S. e. al., "Enhanced model for fake image detection (EMFID) using convolutional neural networks with histogram and wavelet based feature extractions," *ELSEVIER,* 2021.

[2] J. Y. e. al., "Detecting fake images by identifying potential texture difference," *ELSEVIER,* 2020.

[3] Y. D. e. al., *Generative adversarial networks (GAN) based efficient sampling of chemical composition space for inverse design of inorganic materials,* 2020.

[4] K. e. al., "Fake Colorized Image Detection Approaches: A Review," *World Scientific,* 2023.

[5] FastAI, "00-is-it-a-bird-creating-a-model-from-your-own-data.ipynb"

[6] OpenAI, Personal Communication

Q2

May 25, 2023

0.1 Introduction

This question aims to classify 10 different classes animals. First DuckDuckGo search engine was used to scrape images. The result was analyzed using both confusion matrix and t-SNE techniques. The methodology was based on the fastai course example on birds

0.2 Methodology

0.3 Step 1: Check the environment

[2]:
```
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 right and set 'Internet' switch to on")

import os
iskaggle = os.environ.get('KAGGLE_KERNEL_RUN_TYPE', '')

if iskaggle:
    !pip install -Uqq fastai

!pip install -Uqq duckduckgo_search
```

0.4 Step 2: Download images

[ ]:
```
!pip install fastbook
from fastbook import search_images_ddg

from duckduckgo_search import ddg_images
from fastcore.all import *
from fastai.losses import CrossEntropyLossFlat
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE

# Search images using DuckDuckGo, with 200 maximum number of images to retrieve.
```

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

Then search images for ten different animals for classification

```
[ ]: from fastdownload import download_url
     from fastai.vision.all import *

     searches =␣
     ↪'rabbit','bird','kangaroo','cat','dog','tiger','otter','rhinoceros','penguin','flamingo'
     path = Path('ANIMALS')
     from time import sleep

     for o in searches:
         dest = (path/o)
         dest.mkdir(exist_ok=True, parents=True)
         download_images(dest, urls=search_images_ddg(f'{o} photo'))
         sleep(10)  # Pause between searches to avoid over-loading server
         resize_images(path/o, max_size=200, dest=path/o)
```

```
[20]: # Data pre-processing
      # Verify the effectiveness of downloaded images and remove any invalid images
      failed = verify_images(get_image_files(path))
      failed.map(Path.unlink)
      len(failed)
```

```
[20]: 65
```

0.5 Step 3: Model Training

Load data using DataBlock which contains training dataset and vaidation dataset split randomly with 20% of the data allocated for validation.

```
[21]: 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=6)  #Display maximum 6 images below
```

Now training the model using vision_learner

A loss funcion is used to quantify the error between the true outputs and model's predictions Cross entropy loss is implemented as a multiclass loss function. Entropy refers to the level of uncertainty inherent in the variabls possible outcome. Cross entropy is used when adjusting model weights during training, the aim is to minimize the loss by assigning one of several possible classes to each output (Kiprono 2020).

Since we need to classify ten animals into animal categories, the cross entropy lossfunction is well-suited as it is capable to describe a predited animal compare to the actual searched animal. In addition, the cross entropy loss function can classify each input to one class only, which is useful in single-label classification because each image is supposed to belong to a single animal class only.

[22]:
```
learn = vision_learner(dls, resnet18, metrics=accuracy,␣
 ↪loss_func=CrossEntropyLossFlat()) #baseloss
learn.fine_tune(3)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208:
UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
removed in the future, please use 'weights' instead.
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223:
UserWarning: Arguments other than a weight enum or `None` for 'weights' are
deprecated since 0.13 and may be removed in the future. The current behavior is
```

```
equivalent to passing `weights=ResNet18_Weights.IMAGENET1K_V1`. You can also use
`weights=ResNet18_Weights.DEFAULT` to get the most up-to-date weights.
  warnings.warn(msg)
```

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

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

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

In the context of the fine_tune(3), the number 3 represents the number of epochs or complete passes through the training dataset during the fine-tuning process. Each epoch involves forward and backward passes through the network, updating the model's parameters based on the calculated gradients.

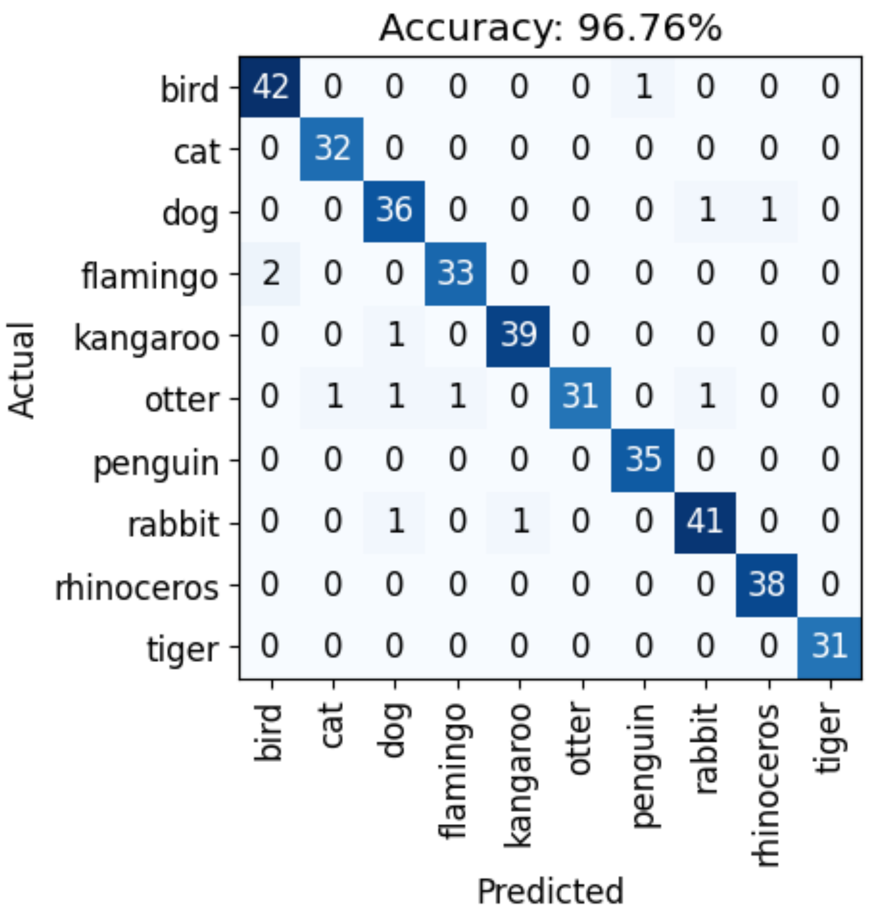
0.6 Step 4: Model evaluation

Heatmap was used as a graphical way to visualize the accuracy of the model in the scheme of warm-to-cool colour. The heatmap was generated based on a confusion matrix, which not only displays the samples that were incorrectly predicted by the proposed model but also shows how they were misclassified (C. Albon, 2018). The x-axis represents for the actual animal class, the y-axis represents for the predicted animal class, and each cell represents one possible combination of predicted and true classes.

```
[9]: import numpy as np
     interp = ClassificationInterpretation.from_learner(learn)
     cm = interp.confusion_matrix()
     accuracy = np.diag(cm).sum() / cm.sum()  # calculate th predicted accuracy
     interp.plot_confusion_matrix()
     plt.title('Accuracy: {:.2f}%'.format(accuracy * 100))
```

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

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

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

```
[9]: Text(0.5, 1.0, 'Accuracy: 96.76%')
```

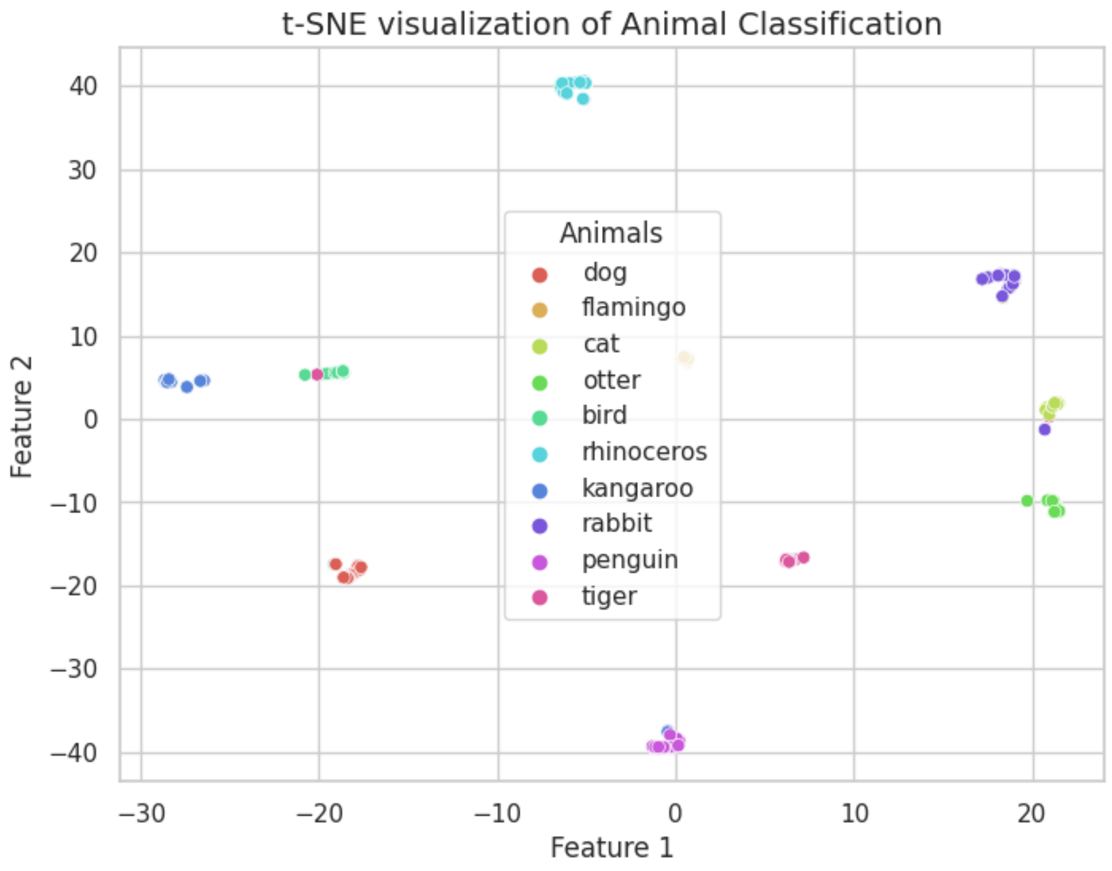


t-distributed stochastic neighbor embedding (t-SNE) is a statistical method for visualizing high-dimensional data by giving each datapoint a location in a two-dimensional map (WikiPedia).

[10]:
```
# Refer to GPT with self-modification
from sklearn.manifold import TSNE
import numpy as np
import seaborn as sns
import pandas as pd
import matplotlib.pyplot as plt

preds, targs = learn.get_preds(dl=dls.valid)
X_embedded = TSNE(n_components=2, learning_rate='auto', perplexity=30,␣
 ↪random_state=42).fit_transform(preds)

animal_labels =␣
 ↪['rabbit','bird','kangaroo','cat','dog','tiger','otter','rhinoceros','penguin','flamingo']

df = pd.DataFrame()
```

```
df["Animal"] = [animal_labels[i] for i in targs]
df["Component 1"] = X_embedded[:, 0]
df["Component 2"] = X_embedded[:, 1]

sns.set(style="whitegrid")
plt.figure(figsize=(8, 6))
sns.scatterplot(x="Component 1", y="Component 2", hue="Animal", palette="hls",␣
↪data=df)
plt.title("t-SNE visualization of Animal Classification", fontsize=14)
plt.xlabel("Feature 1", fontsize=12)
plt.ylabel("Feature 2", fontsize=12)
plt.legend(loc="best", title="Animals")
plt.show()
```

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

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



[12]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
[17]: !jupyter nbconvert --to html '/content/drive/MyDrive/Colab Notebooks/Q2.ipynb'
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/Colab
Notebooks/Q2.ipynb to html
[NbConvertApp] Writing 1287507 bytes to /content/drive/MyDrive/Colab
Notebooks/Q2.html
```

```
[ ]: import os
     os.listdir()
     from google.colab import files
     files.download('Q2.pdf')
```

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

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

### Q3: Detecting Real or Fake images

Step 1: Path Locating & Dataset preparing

```
from google.colab import drive
drive.mount('/content/drive')
```

```
    Mounted at /content/drive
```

```
# Reload the latest Q2 training set data
!unzip '/content/drive/MyDrive/Colab Notebooks/data.zip'
```

```
!pip install fastbook
from fastcore.all import *
from fastai.losses import CrossEntropyLossFlat
import matplotlib.pyplot as plt
from sklearn.manifold import TSNE
from fastdownload import download_url
from fastai.vision.all import *
```

Step 2: Store data into datablock for further processing

The dataset foler has following path:

- dataset
  - Train
    - REAL
    - FAKE
  - Test
    - REAL
    - FAKE

```
path = Path('dataset')
path_test = Path('dataset/test')

dls = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    get_y=parent_label,
    #splitter=RandomSplitter(valid_pct=0.2, seed=42),
    splitter=GrandparentSplitter(train_name='train', valid_name='test'),
    item_tfms=Resize(192, method='squish')
).dataloaders(path)

dls_test = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    get_y=parent_label,
    #splitter=RandomSplitter(valid_pct=0.2, seed=42),
    #splitter=GrandparentSplitter(train_name='train', valid_name='test'),
    item_tfms=Resize(192, method='squish')
).dataloaders(path_test)

dls.show_batch(max_n=6)
```

Step 3: training the model

Resnet50 as Resnet family CNN with 50 layers was tested and evaluated. Cross Entropy Loss has been applied becuase of its effectiveness on binary classification problem. Different models have been tested by changing the parameters, however the accuracy of detecting real or fake images on the test dataset are all around 57% provided that the training accuracy of the model has an accuracy of 98%.

```
learn = vision_learner(dls, resnet50, metrics=accuracy, loss_func=CrossEntropyLossFlat()) #baseloss
learn.fine_tune(3)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.176141	0.129906	0.949850	03:10

epoch	train_loss	valid_loss	accuracy	time
0	0.105042	0.093087	0.964100	04:10
1	0.051007	0.049538	0.982200	04:12
2	0.014240	0.039576	0.986900	04:11

```
# Calculate the accuracy of the trained model on the test dataset
# Get the predictions and targets
preds, targs = learn.get_preds(dl=dls_test)

# Calculate accuracy
acc = accuracy(preds, targs)
print('Accuracy: {:.2f}%'.format(acc.item()*100))
```

```
Accuracy: 57.34%
```

```
dls2 = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    get_y=parent_label,
    splitter=GrandparentSplitter(train_name='train', valid_name='test'),
    item_tfms=Resize(192, method='squish')

).dataloaders(path)

dls_test2 = DataBlock(
    blocks=(ImageBlock, CategoryBlock),
    get_items=get_image_files,
    get_y=parent_label,

    item_tfms=Resize(192, method='squish')

).dataloaders(path_test)
```

In comparsion, Resnet18 has 18 layers and it is more lightweight than Resnet50. There is a trade off between the accuracy and computational resources, depending on the complexity of the model.

```
learn = vision_learner(dls2, resnet18, metrics=accuracy, loss_func=CrossEntropyLossFlat()) #baseloss
learn.fine_tune(3)
```

```
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:208: UserWarning: The parameter 'pretrained' is deprecated since 0.13 and may be
  warnings.warn(
/usr/local/lib/python3.10/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arguments other than a weight enum or `None` for 'weights' are
  warnings.warn(msg)
```

epoch	train_loss	valid_loss	accuracy	time
0	0.212142	0.184448	0.927450	02:03

epoch	train_loss	valid_loss	accuracy	time
0	0.106496	0.078776	0.970950	02:06
1	0.052501	0.055879	0.978800	02:06
2	0.020360	0.050535	0.983800	02:18

```
# Calculate the accuracy of the trained model on the test dataset
# Get the predictions and targets
preds2, targs2 = learn.get_preds(dl=dls_test)

# Calculate accuracy
acc2 = accuracy(preds2, targs2)
print('Accuracy: {:.2f}%'.format(acc2.item()*100))
```

```
Accuracy: 57.14%
```

```
!sudo apt-get install texlive-xetex texlive-fonts-recommended texlive-plain-generic
```

```
!jupyter nbconvert --to pdf /content/drive/MyDrive/test2.ipynb
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/test2.ipynb to pdf
[NbConvertApp] Support files will be in test2_files/
[NbConvertApp] Making directory ./test2_files
[NbConvertApp] Writing 33477 bytes to notebook.tex
[NbConvertApp] Building PDF
[NbConvertApp] Running xelatex 3 times: ['xelatex', 'notebook.tex', '-quiet']
[NbConvertApp] Running bibtex 1 time: ['bibtex', 'notebook']
[NbConvertApp] WARNING | bibtex had problems, most likely because there were no citations
[NbConvertApp] PDF successfully created
[NbConvertApp] Writing 234487 bytes to /content/drive/MyDrive/test2.pdf
```