Homework 5

Note: If this is one of your two late homework submissions, please indicate below; also indicate whether it is your first or second late submission.

This homework assignment has **two parts**. In the first, you will practice building a recommender system; in the second, you'll practice training a neural network for image classification. I strongly advise using Lab 8 and Lab 9 for assistance as you work on this assignment. Make sure to **read the entire assignment**.

You also may need to use other functions. I encourage you to make use of available resources (including the Internet) to help you solve these problems. You can also work with your classmates. If you do work together, you must provide the names of those classmates below.

[Names of Collaborators (if any):]{.underline}

Recommender Systems

We'll work with the data in `data/movies.csv` and `data/movie-ratings.csv`. `movies` contains a list of $9,737$ movies and their basic description – title, year of release, and genres, separated by vertical bars (for example, `Comedy|Romance`). `movie-ratings` contains ratings of movies by 610 users, on a scale from 0 to 5.

The data come from this source at Kaggle: https://www.kaggle.com/datasets/gargmanas/movierecommenderdataset/

Exercise 1

Read both data files into Python. (You can also use R, if you prefer. If you do use R, I would recommend working with a smaller subset of the data.)

Movie title and year of release are in the same column. Create a new variable that represents year of release, as a four-digit number.

Exercise 2

Create a histogram of year of release. How would you describe the shape of the distribution? When were the most movies released?

Exercise 3

Create a bar chart of the top 10 highest-rated movies.

Exercise 4

Create a variable called `string` that contains the text of each movie's genres, title, and year of release. For example, the value of `string` for `movieID == 3` should be: `"Adventure Children Fantasy Jumanji (1995)"`.

Exercise 5

Using the `string` variable, create a tf-idf matrix with `TfidfVectorizer` and `tfv.fit`.

Exercise 6

Use the sigmoid kernel from `scikit-learn` to calculate pairwise similarities between all items in your tf-idf matrix.

Exercise 7

Define a function, `give_recommendation()`, that takes as input the title of a movie and returns the top 10 most similar movies.

Exercise 8

What movies does your recommender system suggest for a user who likes "Toy Story" (released in 1995)?

For 234 Students:

Exercise 9

Now we'll try making content-based recommendations. Turn the data into a CSR matrix using `scipy.sparse`.

Exercise 10

Fit a *k*-nearest neighbors model, using cosine similarity as the distance metric.

Exercise 11

Identify which movies your model deems most similar to "GoldenEye" (a James Bond movie, also released in 1995).

Image Classification

Now we'll work with the data in `data/Animals`. This dataset, intended for animal image classification, comes from Kaggle. It consists of $3,000$ JPEG RGB images, each of which are 256 x 256 pixels, that have been divided into three classes with $1,000$ images in each class. The classes are `cats`, `dogs`, and `snakes`.

Exercise 12

Randomly select 150 images of cats, 150 images of dogs, and 150 images of snakes. Set these aside in another directory labeled `test_images` to be your testing set. Using the same approach, randomly select another 150 images from each class, and set these aside in a `validation_images` directory to be your validation set.

Exercise 13

Display a random image from each of the three classes in your training set to verify that the data are set up correctly.

Exercise 14

Using `ImageDataGenerator` and `flow_from_directory`, rescale your training, testing, and validation sets. Load and preprocess your images in batches of size 10.

Exercise 15

Set up a convolutional neural net (CNN) with 7 layers using `Sequential()`. The layers should be as follows:

1. 2D convolutional input layer with a ReLU activation function;
2. Max pooling layer for 2D spatial data;
3. 2D convolutional layer with ReLU activation;
4. Max pooling layer for 2D spatial data;
5. Flattening layer;
6. Dense layer with 128 units and ReLU activation;
7. Dense output layer with softmax activation.

Exercise 16

Using Adam and categorical cross-entropy, fit the network you've created and let it run for 12 epochs.

Exercise 17

Create a plot of the accuracy and loss by the number of epochs.

For 234 Students:

Exercise 18

Look at your model's accuracy on your testing set. How did it do?

Exercise 19

Generate your model's prediction for a random image from the dataset.

Exercise 20

Create a confusion matrix using your testing set. Visualize the matrix as a heat map. Which classes was your model best at predicting? Which was it worst at predicting? How do you know?

In [1]:
```
# Exercise 1
# Read both data files into Python. (You can also use R, if you prefer. I
# Movie title and year of release are in the same column. Create a new va
```

In [27]:
```
import pandas as pd
import numpy as np
import re
import warnings
import matplotlib.pyplot as plt

warnings.filterwarnings('ignore')

movie = pd.read_csv("~/Desktop/homework-5/data/movies.csv")
rating = pd.read_csv("~/Desktop/homework-5/data/movie-ratings.csv")
```

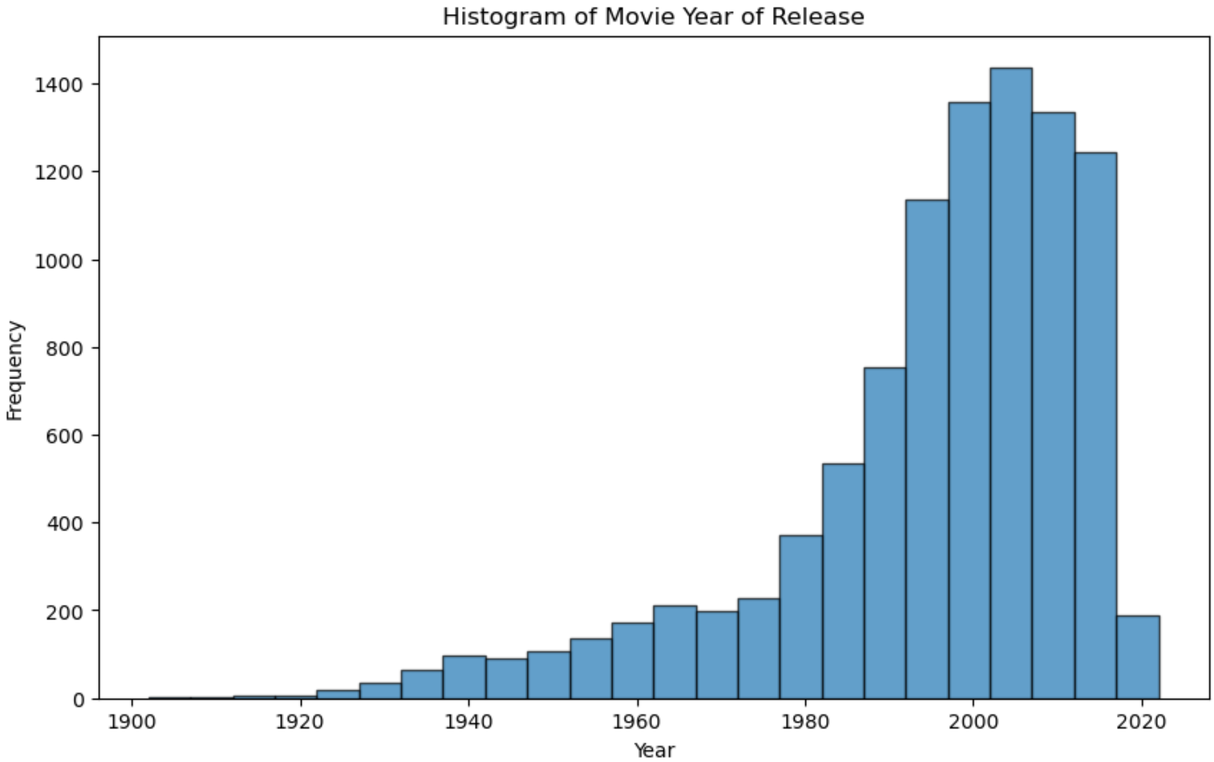
In [28]:
```
movie['year'] = movie['title'].str.extract(r'\((\d{4})\)')
movie = movie.dropna(subset=['year'])
movie.head()
```

Out[28]:

	movieId	title	genres	year
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995
1	2	Jumanji (1995)	Adventure|Children|Fantasy	1995
2	3	Grumpier Old Men (1995)	Comedy|Romance	1995
3	4	Waiting to Exhale (1995)	Comedy|Drama|Romance	1995
4	5	Father of the Bride Part II (1995)	Comedy	1995

In [84]:
```
# Exercise 2
# Create a histogram of year of release. How would you describe the shape
```

In [29]:
```
movie['year'] = pd.to_numeric(movie['year'], errors='coerce', downcast='i
plt.figure(figsize=(10,6))
plt.hist(movie['year'], bins=range(movie['year'].min(), movie['year'].max
plt.title("Histogram of Movie Year of Release")
plt.xlabel("Year")
plt.ylabel("Frequency")
plt.show()
```



In [86]:
```
# The shape of the distribution is skewed to the left, and most movies we
```

In [87]:
```
# Exercise 3
# Create a bar chart of the top 10 highest-rated movies.
```

In [31]:
```
fulldata = pd.merge(movie,rating,on="movieId")
fulldata.head(10)
```

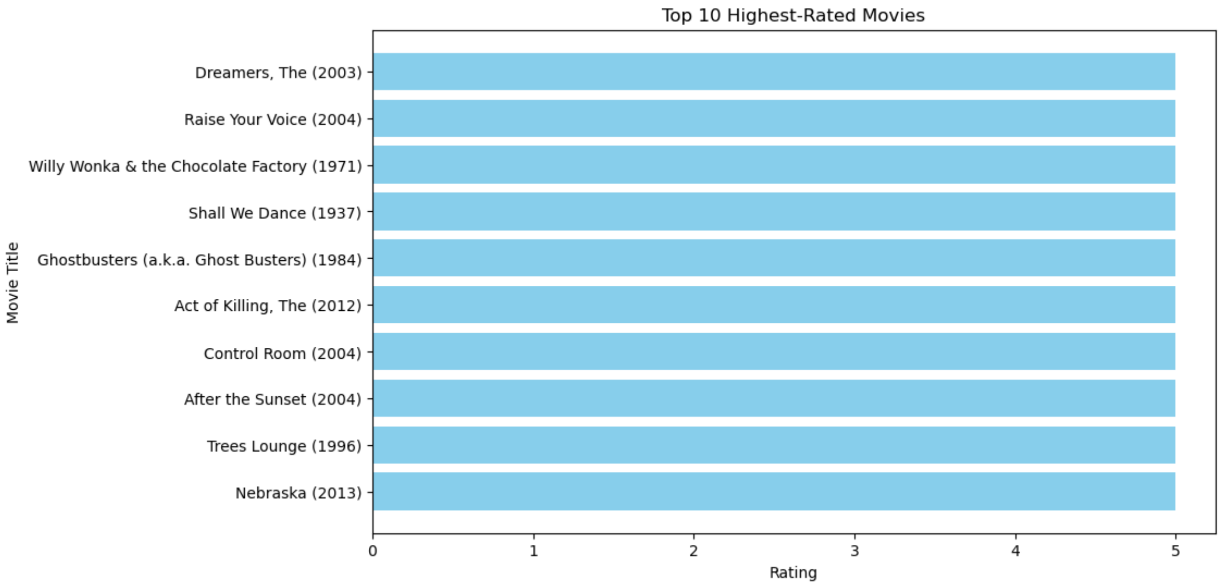
Out[31]:

	movieId	title	genres	year	userId	rating	tim
0	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	1	4.0	964
1	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	5	4.0	847
2	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	7	4.5	1106
3	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	15	2.5	151
4	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	17	4.5	1305
5	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	18	3.5	145
6	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	19	4.0	96
7	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	21	3.5	140
8	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	27	3.0	962
9	1	Toy Story (1995)	Adventure|Animation|Children|Comedy|Fantasy	1995	31	5.0	85

In [33]:

```
top_10_mov = (
    fulldata
    .drop_duplicates(subset='title')
    .sort_values(by='rating', ascending=False)
    .head(10)
)

plt.figure(figsize=(10, 6))
plt.barh(top_10_mov['title'], top_10_mov['rating'], color='skyblue')
plt.xlabel('Rating')
plt.ylabel('Movie Title')
plt.title('Top 10 Highest-Rated Movies')
plt.gca().invert_yaxis()  # Invert y-axis to have the highest rating at t
plt.show()
```



In [93]:
```
# Exercise 4
# Create a variable called string that contains the text of each movie's
```

In [34]:
```
fulldata['clean_genres'] = fulldata['genres'].str.replace('|', ' ')
fulldata['string'] = fulldata['clean_genres'] + " " + fulldata['title']

fulldata.loc[fulldata['movieId'] == 3, 'string'].values[0]
```

Out[34]:
```
'Comedy Romance Grumpier Old Men (1995)'
```

In [21]:
```
# Exercise 5
# Using the string variable, create a tf-idf matrix with TfidfVectorizer
```

In [38]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

rec_data = fulldata.copy() # create a copy of the orginal dataset
rec_data
rec_data.drop_duplicates(subset = "title", keep = "first", inplace = True
rec_data.reset_index(drop = True, inplace = True)


tfv = TfidfVectorizer(min_df=3, max_features=None, strip_accents="unicode
                      token_pattern=r"\w{1,}", ngram_range=(1, 3), stop_w

tfv_matrix = tfv.fit_transform(rec_data['string'])
tfv_matrix
```

Out[38]:
```
<9706x3435 sparse matrix of type '<class 'numpy.float64'>'
	with 71966 stored elements in Compressed Sparse Row format>
```

In [62]:
```
# Exercise 6
# Use the sigmoid kernel from scikit-learn to calculate pairwise similari
```

In [39]:
```
from sklearn.metrics.pairwise import sigmoid_kernel

sig = sigmoid_kernel(tfv_matrix, tfv_matrix) # Computing sigmoid kernel
rec_indices = pd.Series(rec_data.index, index = rec_data["title"]).drop_d
```

In [64]:
```
# Exercise 7
# Define a function, give_recommendation(), that takes as input the title
```

In [41]:
```
def give_recommendation(title, sig = sig):

    idx = rec_indices[title]

    sig_score = list(enumerate(sig[idx]))  # Getting pairwsie similarity
    sig_score = sorted(sig_score, key=lambda x: x[1], reverse=True)
    sig_score = sig_score[1:11]
    movie_indices = [i[0] for i in sig_score]

    # Top 10 most similar anime
    rec_dic = {"No" : range(1,11),
               "Movie Name": fulldata["title"].iloc[movie_indices].values
               "Rating": fulldata["rating"].iloc[movie_indices].values,}
    dataframe = pd.DataFrame(data = rec_dic)
    dataframe.set_index("No", inplace = True)

    print(f"Recommendations for {title} viewers :\n")

    return dataframe.style.set_properties(**{"background-color": "white",
```

In [66]:
```
# Exercise 8
# What movies does your recommender system suggest for a user who likes "
```

In [42]:
```
give_recommendation("Toy Story (1995)")
```

```
Recommendations for Toy Story (1995) viewers :
```

Out[42]:

No	Movie Name	Rating
1	Pocahontas (1995)	3.000000
2	Mary Shelley's Frankenstein (Frankenstein) (1994)	3.000000
3	Babe (1995)	1.500000
4	While You Were Sleeping (1995)	3.000000
5	Dolores Claiborne (1995)	3.000000
6	Star Wars: Episode IV - A New Hope (1977)	4.000000
7	Braveheart (1995)	4.000000
8	French Kiss (1995)	3.000000
9	Quiz Show (1994)	4.000000
10	Clerks (1994)	3.000000

In [ ]:

```
import zipfile
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import tensorflow as tf
from tensorflow import keras
from keras.layers import *
from keras.models import *
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import os, shutil
import warnings
import random
warnings.filterwarnings('ignore')
```

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

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content
```

```
zip_path = '/content/drive/My Drive/Colab Notebooks/data.zip'

extract_path = '/content/drive/My Drive/Colab Notebooks/data'

shutil.unpack_archive(zip_path, extract_path)
```

Exercise 12

Randomly select 150 images of cats, 150 images of dogs, and 150 images of snakes. Set these aside in another directory labeled test_images to be your testing set. Using the same approach, randomly select another 150 images from each class, and set these aside in a validation_images directory to be your validation set.

```
base_dir = '/content/drive/My Drive/Colab Notebooks/data/data/Animals'

test_dir = '/content/drive/My Drive/Colab Notebooks/data/test_images'
validation_dir = '/content/drive/My Drive/Colab Notebooks/data/validation_images'
train_dir = '/content/drive/My Drive/Colab Notebooks/data/train_images'

classes = ['cats', 'dogs', 'snakes']

for directory in [test_dir, validation_dir, train_dir]:
    os.makedirs(directory, exist_ok=True)
    for class_name in classes:
        os.makedirs(os.path.join(directory, class_name), exist_ok=True)


def select_and_move_images(source_dir, target_dir, classes, num_images):
    for class_name in classes:
        target_class_dir = os.path.join(target_dir, class_name)

        # Check if the target class directory already contains images
        if os.path.exists(target_class_dir) and any(img.endswith('.jpg') for img in os.listdir(target_c
            continue

    for class_name in classes:
        target_class_dir = os.path.join(target_dir, class_name)

        if not os.path.exists(target_class_dir) or not any(img.endswith('.jpg') for img in os.listdir(t
            source_class_dir = os.path.join(source_dir, class_name)
            os.makedirs(target_class_dir, exist_ok=True)

            images = [img for img in os.listdir(source_class_dir) if img.endswith('.jpg')]
            selected_images = random.sample(images, num_images)
```

```
            for image in selected_images:
                shutil.move(os.path.join(source_class_dir, image), os.path.join(target_class_dir, image


num_images = 150
# test
select_and_move_images(base_dir, test_dir, classes, num_images)
# validation
select_and_move_images(base_dir, validation_dir, classes, num_images)


def move_remaining_to_train(source_dir, train_dir, classes):
    for class_name in classes:
        source_class_dir = os.path.join(source_dir, class_name)
        train_class_dir = os.path.join(train_dir, class_name)

        images = [img for img in os.listdir(source_class_dir) if img.endswith('.jpg')]

        for image in images:
            shutil.move(os.path.join(source_class_dir, image), os.path.join(train_class_dir, image))

move_remaining_to_train(base_dir, train_dir, classes)


def count_images(directory, classes):
    for class_name in classes:
        class_dir = os.path.join(directory, class_name)
        print(f"{class_name}: {len(os.listdir(class_dir))} images")

print("Test set:")
count_images(test_dir, classes)

print("Validation set:")
count_images(validation_dir, classes)

print("Train set:")
count_images(train_dir, classes)
```

```
Test set:
cats: 150 images
dogs: 150 images
snakes: 150 images
Validation set:
cats: 150 images
dogs: 150 images
snakes: 150 images
Train set:
cats: 700 images
dogs: 700 images
snakes: 700 images
```

Exercise 13

Display a random image from each of the three classes in your training set to verify that the data are set up correctly.

```
classes = ['cats', 'dogs', 'snakes']

def plot_images(train_dir, classes):
    plt.figure(figsize=(12, 12))

    for i, class_name in enumerate(classes):
        class_dir = os.path.join(train_dir, class_name)

        images = [img for img in os.listdir(class_dir) if img.endswith('.jpg')]

        random_image = random.choice(images)
```

```
        img_path = os.path.join(class_dir, random_image)
        img = image.load_img(img_path, target_size=(256, 256))
        img_array = image.img_to_array(img)

        plt.subplot(1, len(classes), i+1)
        plt.imshow(img_array.astype('uint8'))
        plt.title(class_name)
        plt.axis('off')  # Turn off axis

    plt.show()

plot_images(train_dir, classes)
```

Exercise 14

Using ImageDataGenerator and flow_from_directory, rescale your training, testing, and validation sets. Load and preprocess your images in batches of size 10 .

```
# Training set
train_gen = ImageDataGenerator(rescale = 1.0/255.0) # scale the data
train_image_generator = train_gen.flow_from_directory(train_dir,
                                                      target_size=(150, 150),
                                                      batch_size=10,
                                                      class_mode='categorical')

# Validation set
val_gen = ImageDataGenerator(rescale = 1.0/255.0) # scale the data
val_image_generator = train_gen.flow_from_directory(validation_dir,
                                                    target_size=(150, 150),
                                                    batch_size=10,
                                                    class_mode='categorical')

# Testing set
test_gen = ImageDataGenerator(rescale = 1.0/255.0) # scale the data
test_image_generator = train_gen.flow_from_directory(test_dir,
                                                     target_size=(150, 150),
                                                     batch_size=10,
                                                     class_mode='categorical')
```

```
Found 2100 images belonging to 3 classes.
Found 450 images belonging to 3 classes.
Found 450 images belonging to 3 classes.
```

Exercise 15

Set up a convolutional neural net (CNN) with 7 layers using Sequential(). The layers should be as follows:

1. 2D convolutional input layer with a ReLU activation function;
2. Max pooling layer for 2D spatial data;

3. 2D convolutional layer with ReLU activation;
4. Max pooling layer for 2D spatial data;
5. Flattening layer;
6. Dense layer with 128 units and ReLU activation;
7. Dense output layer with softmax activation.

```
model = Sequential()

model.add(Conv2D(filters=32, kernel_size=3, strides=1,
                 padding='same', activation='relu',
                 input_shape=[150, 150, 3]))
model.add(MaxPooling2D(2, ))
model.add(Conv2D(filters=64, kernel_size=3, strides=1,
                 padding='same', activation='relu'))
model.add(MaxPooling2D(2))

model.add(Flatten())

model.add(Dense(128, activation='relu'))
model.add(Dropout(0.25))
model.add(Dense(128, activation='relu'))
model.add(Dense(3, activation='softmax'))

# print the model summary
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 150, 150, 32)	896
max_pooling2d (MaxPooling2D)	(None, 75, 75, 32)	0
conv2d_1 (Conv2D)	(None, 75, 75, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 37, 37, 64)	0
flatten (Flatten)	(None, 87616)	0
dense (Dense)	(None, 128)	11,214,976
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 128)	16,512
dense_2 (Dense)	(None, 3)	387

Total params: 11,251,267 (42.92 MB)
Trainable params: 11,251,267 (42.92 MB)
Non-trainable params: 0 (0.00 B)

Exercise 16

Using Adam and categorical cross-entropy, fit the network you've created and let it run for 12 epochs.

```
early_stopping = keras.callbacks.EarlyStopping(patience=12)

model.compile(optimizer='Adam', loss='categorical_crossentropy', metrics=['accuracy'])
hist = model.fit(train_image_generator,
                 epochs=12,
                 verbose=1,
                 validation_data=val_image_generator,
                 steps_per_epoch=15000//32, validation_steps=3000//32,
                 callbacks=early_stopping)
```

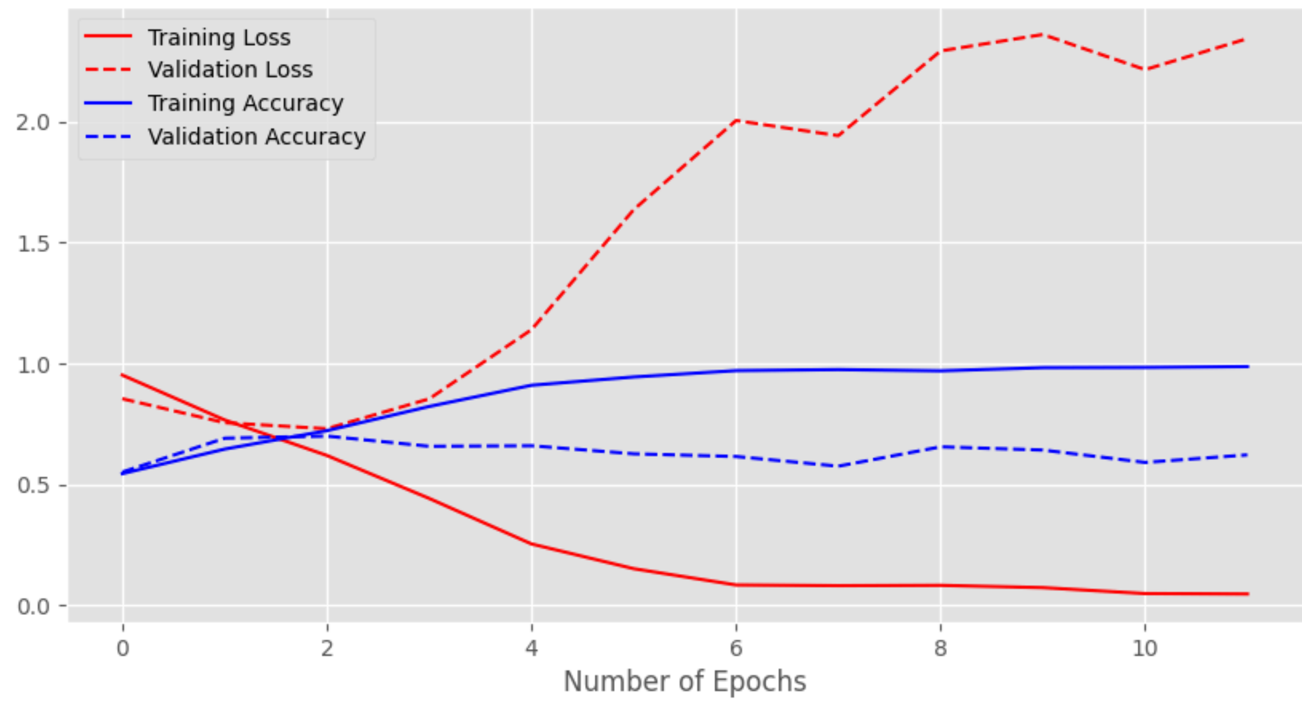
```
Epoch 1/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 128s 263ms/step - accuracy: 0.5221 - loss: 1.0260 - val_accuracy: 0.55
```

```
Epoch 2/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 139s 259ms/step - accuracy: 0.6295 - loss: 0.7784 - val_accuracy: 0.69
Epoch 3/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 136s 247ms/step - accuracy: 0.7256 - loss: 0.6167 - val_accuracy: 0.70
Epoch 4/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 126s 267ms/step - accuracy: 0.8267 - loss: 0.4361 - val_accuracy: 0.65
Epoch 5/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 111s 235ms/step - accuracy: 0.9079 - loss: 0.2549 - val_accuracy: 0.66
Epoch 6/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 116s 246ms/step - accuracy: 0.9448 - loss: 0.1535 - val_accuracy: 0.62
Epoch 7/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 138s 237ms/step - accuracy: 0.9721 - loss: 0.0804 - val_accuracy: 0.61
Epoch 8/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 116s 246ms/step - accuracy: 0.9780 - loss: 0.0708 - val_accuracy: 0.57
Epoch 9/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 151s 264ms/step - accuracy: 0.9640 - loss: 0.0938 - val_accuracy: 0.65
Epoch 10/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 134s 248ms/step - accuracy: 0.9823 - loss: 0.0742 - val_accuracy: 0.64
Epoch 11/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 137s 238ms/step - accuracy: 0.9880 - loss: 0.0379 - val_accuracy: 0.59
Epoch 12/12
468/468 ━━━━━━━━━━━━━━━━━━━━ 146s 245ms/step - accuracy: 0.9883 - loss: 0.0436 - val_accuracy: 0.62
```

Exercise 17

Create a plot of the accuracy and loss by the number of epochs

```
# plot the error and accuracy
h = hist.history
plt.style.use('ggplot')
plt.figure(figsize=(10, 5))
plt.plot(h['loss'], c='red', label='Training Loss')
plt.plot(h['val_loss'], c='red', linestyle='--', label='Validation Loss')
plt.plot(h['accuracy'], c='blue', label='Training Accuracy')
plt.plot(h['val_accuracy'], c='blue', linestyle='--', label='Validation Accuracy')
plt.xlabel("Number of Epochs")
plt.legend(loc='best')
plt.show()
```



开始借助 AI 编写或生成代码。