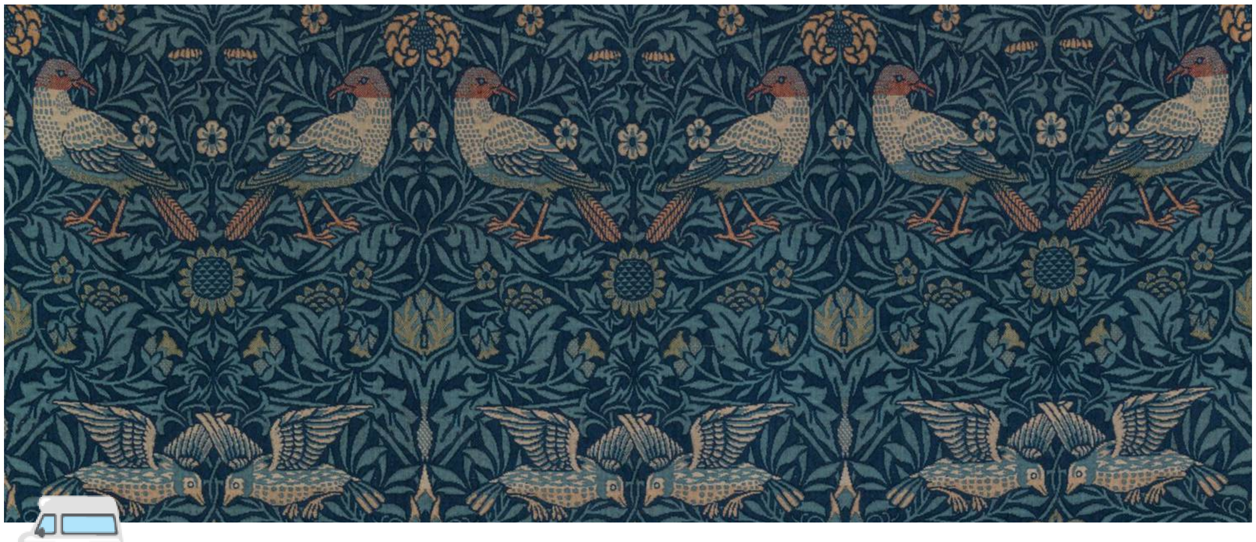

CMP 409 AI

CHAPTER 1:

Project setup

open command prompt and install tensorflow, numpy, matplotlib

```
pip install tensorflow
pip install numpy
pip install matplotlib
```

Add the imports required

```
import numpy as np
import random
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dense,
```

The dataset we will be using to train our model has over a 1000 images for cats and dogs and we will be using them to train our model and test them also to make it fully capable of being able to tell a cat and a dog from just an image

NOTE: The dataset is over 1.5GB in size so it might take a long time to load the dataset and to download it

```
X_train = np.loadtxt('input.csv', delimiter=',')
Y_train = np.loadtxt('labels.csv', delimiter=',')

X_test = np.loadtxt('input_test.csv', delimiter=',')
Y_test = np.loadtxt('labels_test.csv', delimiter=',')
```

Print shape of dataset after it loads

```
print("1. Shape of X_train: ", X_train.shape)
print("2. Shape of Y_train: ", Y_train.shape)
print("3. Shape of X_test: ", X_test.shape)
print("4. Shape of Y_test: ", Y_test.shape)

# RESULT
# 1. Shape of X_train: (2000, 30000)
# 2. Shape of Y_train: (2000,)
# 3. Shape of X_test: (400, 30000)
# 4. Shape of Y_test: (400,)
```

On line 1. the 2000 represents the total number of images in our training set. On line 3. the 400 represents the total number of images in our test set
Every single image is of the size 100 X 100 pixels (100 width, 100 height) with 3 for RGB channels, that's why we have 30000 on line 1. and line 3. (100 * 100 * 3 = 3000)

Next we want to reshape this images and remove the (..., empty) in line 2. and line 4.

```
X_train = X_train.reshape(len(X_train), 100, 100, 3)
Y_train = Y_train.reshape(len(Y_train), 1)
```

```
X_test = X_test.reshape(len(X_test), 100, 100, 3)
Y_test = Y_test.reshape(len(Y_test), 1)
```

Print shape of dataset again

```
print("1. Shape of X_train: ", X_train.shape)
print("2. Shape of Y_train: ", Y_train.shape)
print("3. Shape of X_test: ", X_test.shape)
print("4. Shape of Y_test: ", Y_test.shape)

# RESULT
# 1. Shape of X_train: (2000, 100, 100, 3)
# 2. Shape of Y_train: (2000, 1)
# 3. Shape of X_test: (400, 100, 100, 3)
# 4. Shape of Y_test: (400, 1)
```

Let's print a single image data to see what it consists of

```
X_train[1,:]

# Result
# array([[[number.,number., number.] ... and many more ]])
```

from the result we see that it consists of numbers that range from 0 to 255 which is the typical rgb color range

So with this knowledge, to properly train our model we need to rescale this values to be between 0 to 1. To do that we divide all the values by 255

```
X_train = X_train/255.0
X_test = X_test/255.0
```

Let's print a single image data again to see our numbers are now between 0-1

```
X_train[1,:]

# Result
# array([[[number.,number., number.] ... and many more ]])
```

Next we want to display a random image from our train dataset using matplotlib (we already imported this up above)

```
index = random.randint(0, len(X_train))
plt.imshow(X_train[idx, :])
plt.show()

# Result
# displays random image from dataset
```

CHAPTER 2: Model (Building our model)
model = Sequential()

Now how the `Sequential()` is going to work is that the layers are stacked up one after another in the sequence. First the convolutional layer then the Max Pooling layer and then the convolutional layer and the Max Pooling layer and then other layers. This is how a Convolutional Neural Network is made

```
model = Sequential([
   Conv2D(32, (3,3), activation='relu', input_shape=(100, 100, 
   MaxPoolin2D((2,2)),
   Conv2D(32, (3,3), activation='relu'),
   MaxPoolin2D((2,2)),

   Flatten(),
   Dense(64, activation='relu')
   Dense(1, activation='sigmoid')
])

# Explanation

#  Conv2D Params
#  1st: Number of filters we want to use
#  2nd: Size of the filter we want to use (we used 3,3 sized fi
#  3rd: Activation function we used is 'relu'
#  4th: The shape of the input our model should be expecting. O

#  MaxPoolin2D
#  1st: The filter size

#  Flatten flattens the layer

#  Dense (which refers to fully connected layers)
#  1st: Number of neurons we want to keep in our Dense layers
#  2nd: The activation function we want to use

#  The last Dense, is the last fully connected layer which is o
#  The output layer must have the same amount of neuron as our
#  And for our output classification we will be using Binary wh
#  neuron.
#  The activation function we used is 'sigmoid' which is a Bina

#  Congratulations, this is how a real model looks like in kera
#  successfully implemented one
```

Next up
We want to add our loss function and the back propagation

```
model.compile(loss='binary_crossentropy', optimizer='adam', metr

# model.compile
# compiles the model we just created by adding the loss, the bac

# loss
# We use 'binary_crossentropy' because we are working with a Bin
# There are other loss functions out there to use, it all depend
# application you are making
```

Now up on to this point, keras using tensorflow has already created the computational graphs needed in the background. Now we want to train the model.

```
model.fit(X_train, Y_train, epochs = 5, batch_size = 64)

# Epoch 1/5
# 32/32 [===========================] - ...ms/step - loss: 0.35
# X 5 (times - since epoch was set to 5, it trains the model on
```

Notice on each epoch the `loss` decreases while the `accuracy` increases. Note that you can run the `model.fit(...)` multiple times to improve the models accuracy. And you should until the accuracy on an epoch reaches 0.9

Next up:
Lets evaluate our model based on our test data to see how accurate our model is

```
model.evaluate(X_test, Y_test)
# result
# 13/13 [===========================] - ...ms/step - loss: 0.97
```

Now, our accuracy at 0.6 is too low, our expectations should be above 0.9 and this can be improved on by increasing the amount of test dataset we use in training our model.

The training data we currently have is **2000**, if we had more we could improve our test accuracy.

CHAPTER 3: Making Predictions

We want to now select some random image and print whether the image contains a dog or a cat

```
# selects a random image from our test dataset and displays it
idx2 = random.randint(0, len(Y_test))
plt.imshow(X_test[idx2, :])
plt.show()

# prints "Our model says it is a ?"
y_pred = model.predict(X_test[idx2, :]).reshape(1, 100, 100, 3)
pred = "cat" if y_pred > 0.5 else "dog"
print("Our model says it is a :", pred)

# Result
# [           ]
# |           |
# |   image   |
# |           |
# [           ]
# Our model says it is a : dog
```

Feel free to run the above code multiple time to see that our model guesses pretty well