Before you turn this problem in, make sure everything runs as expected. First, **restart the kernel** (in the menubar, select Kernel→Restart) and then **run all cells** (in the menubar, select Cell→Run All).

Make sure you fill in any place that says `YOUR CODE HERE` or "YOUR ANSWER HERE", as well as your name and collaborators below:

In [1]:
```
NAME = ""
COLLABORATORS = ""
```

Problem description

To a large degree, financial data has traditionally been numeric in format.

But in recent years, non-numeric formats like image, text and audio have been introduced.

Private companies have satellites orbiting the Earth taking photos and offering them to customers. A financial analyst might be able to extract information from these photos that could aid in the prediction of the future price of a stock

- Approximate number of customers visiting each store: count number of cars in parking lot
- Approximate activity in a factory by counting number of supplier trucks arriving and number of delivery trucks leaving
- Approximate demand for a commodity at each location: count cargo ships traveling between ports

In this assignment, we will attempt to recognize ships in satellite photos. This would be a first step toward counting.

As in any other domain: specific knowledge of the problem area will make you a better analyst.

For this assignment, we will ignore domain-specific information and just try to use a labeled training set (photo plus a binary indicator for whether a ship is present/absent in the photo), assuming that the labels are perfect.

Goal:

In this notebook, you will need to create a model in `TensorFlow/Keras` to classify satellite photos.

- The features are images: 3 dimensional collection of pixels
    - 2 spatial dimensions
    - 1 dimension with 3 features for different parts of the color spectrum: Red, Green, Blue
- The labels are either 1 (ship is present) or 0 (ship is not present)

There are two notebook files in this assignment:

- The one you are viewing now: First and only notebook you need to work on.
    - Train your models here
    - There are cells that will save your models to a file
- **`Model_test.ipynb`**:
    - PLEASE IGNORE

You will create several Keras `Sequential` models, of increasing complexity

- A model that implements only a Classification Head (no transformations other than perhaps rearranging the image)
- A model that adds a Dense layer before the head
- (Later assignment) A model that adds Convolutional layers before the Head

Learning objectives

- Learn how to construct Neural Networks using Keras Sequential model
- Appreciate how layer choices impact number of weights

Imports modules

In [2]:
```
## Standard imports
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import sklearn

import os
import math

%matplotlib inline


## Import tensorflow and check the version
import tensorflow as tf
from tensorflow.keras.utils import plot_model

print("Running TensorFlow version ",tf.__version__)

# Parse tensorflow version
import re

version_match = re.match("([0-9]+)\.([0-9]+)", tf.__version__)
tf_major, tf_minor = int(version_match.group(1)) , int(version_match.group(2))
print("Version {v:d}, minor {m:d}".format(v=tf_major, m=tf_minor) )
```

```
Running TensorFlow version  2.4.1
Version 2, minor 4
```

API for students

We have defined some utility routines in a file `helper.py`. There is a class named `Helper` in it.

This will simplify problem solving

More importantly: it adds structure to your submission so that it may be easily graded

`helper = helper.Helper()`

- getData: Get a collection of labeled images, used as follows

  `data, labels = helper.getData()`

- showData: Visualize labelled images, used as follows

  `helper.showData(data, labels)`

- plot training results: Visualize training accuracy, loss and validation accuracy, loss

  `helper.plotTrain(history, modelName)`, *where history is the result of model training*

- save model: save a model in `./models` directory

  `helper.saveModel(model, modelName)`

- save history: save a model history in `./models` directory

In [3]:

```
# Load the helper module
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# Reload all modules imported with %aimport
%reload_ext autoreload
%autoreload 1

# Import nn_helper module
import helper
%aimport helper
```

You may pass an optional `data_dir` argument to the

constructor **`helper.Helper`**

- if you have your data directory located somewhere other than the default location
- no need to provide an argument otherwise

In [4]:
```
helper = helper.Helper()
```

Get the data

The first step in our Recipe is Get the Data.

We have provided a utility method `getData` to simplify this for you

In [5]:
```
# Get the data
data, labels = helper.getData()
n_samples, width, height, channel = data.shape

print("Data shape: ", data.shape)
print("Labels shape: ", labels.shape)
print("Label values: ", np.unique(labels))
```

```
Data shape:  (4000, 80, 80, 3)
Labels shape:  (4000,)
Label values:  [0 1]
```

We will shuffle the examples before doing anything else.

This is usually a good idea

- Many datasets are naturally arranged in a *non-random* order, e.g., examples with the sample label grouped together
- You want to make sure that, when you split the examples into training and test examples, each split has a similar distribution of examples

In [6]:
```
# Shuffle the data first
data, labels = sklearn.utils.shuffle(data, labels, random_state=42)
```

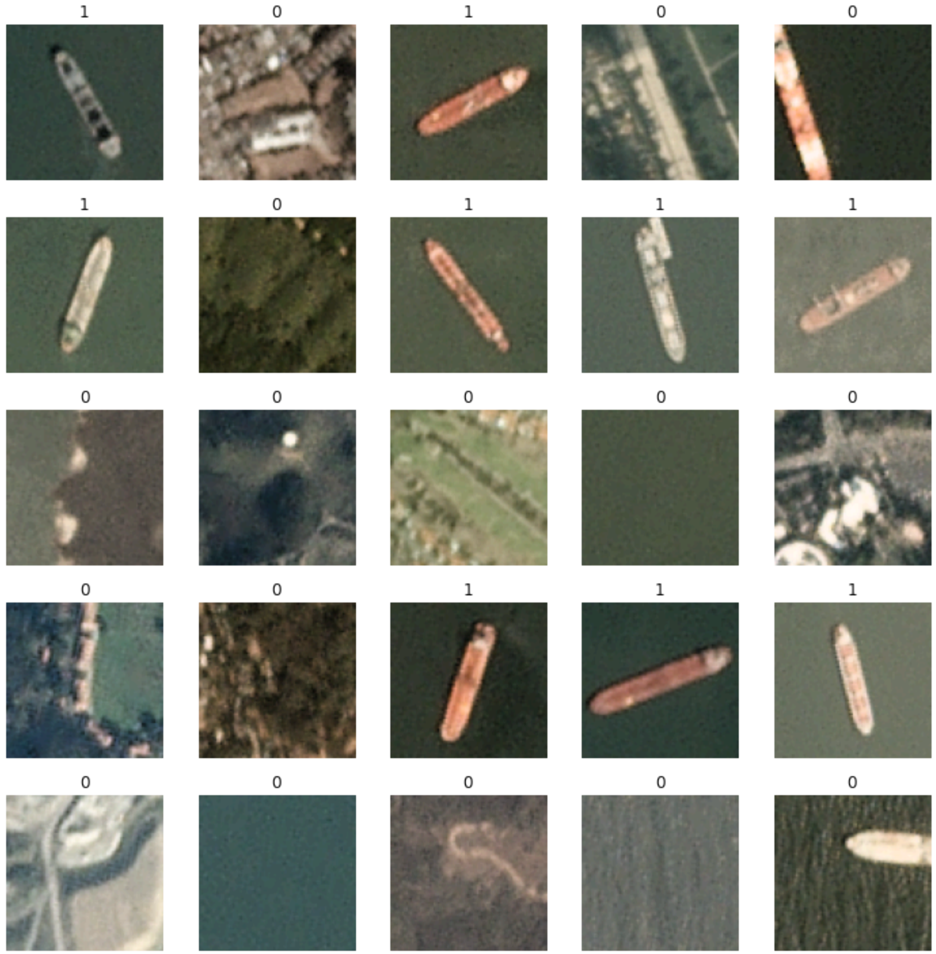
Have a look at the data

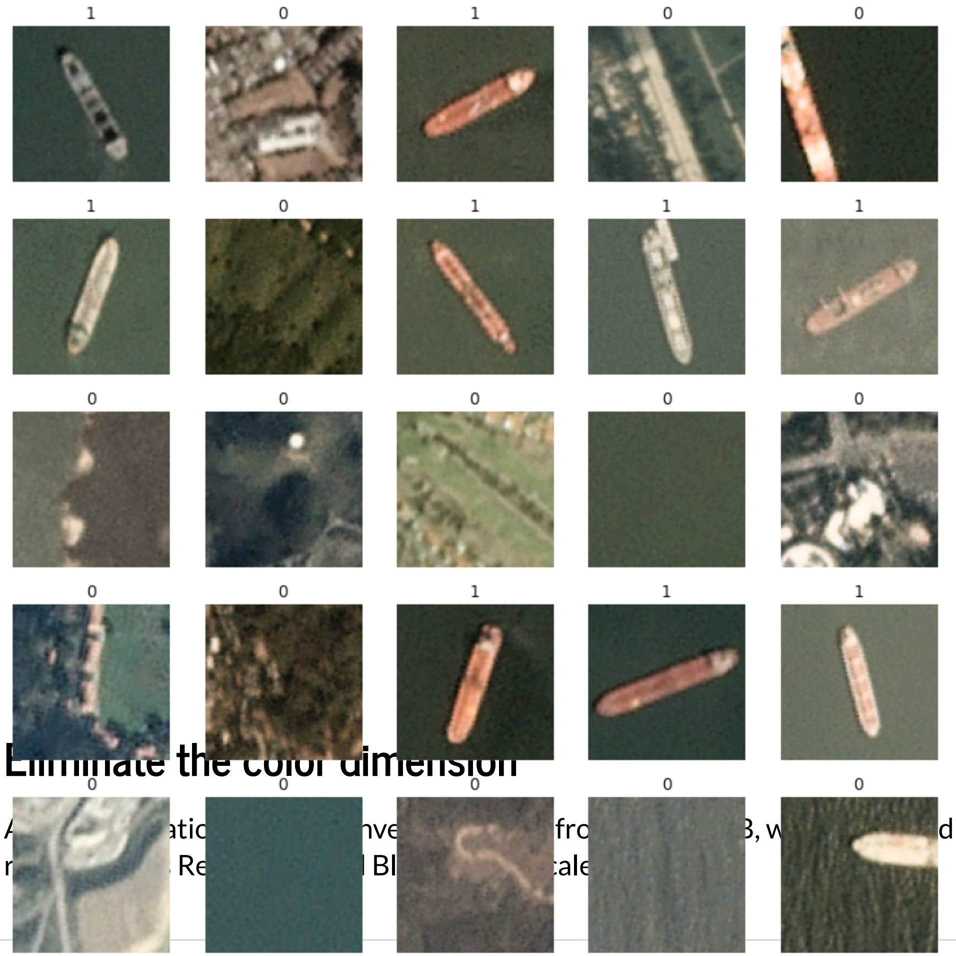
We will not go through all steps in the Recipe, nor in depth.

But here's a peek

In [7]:
```
# Visualize the data samples
helper.showData(data[:25], labels[:25])
```

Out[7]:





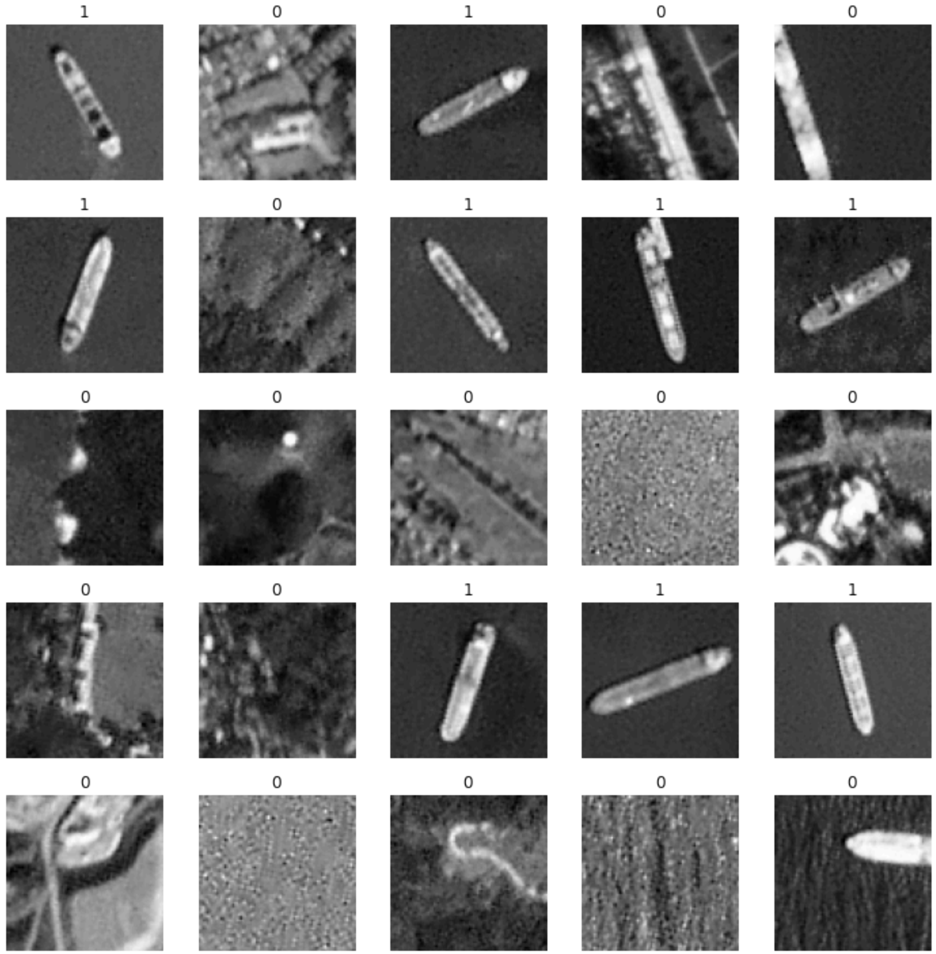
Eliminate the color dimension

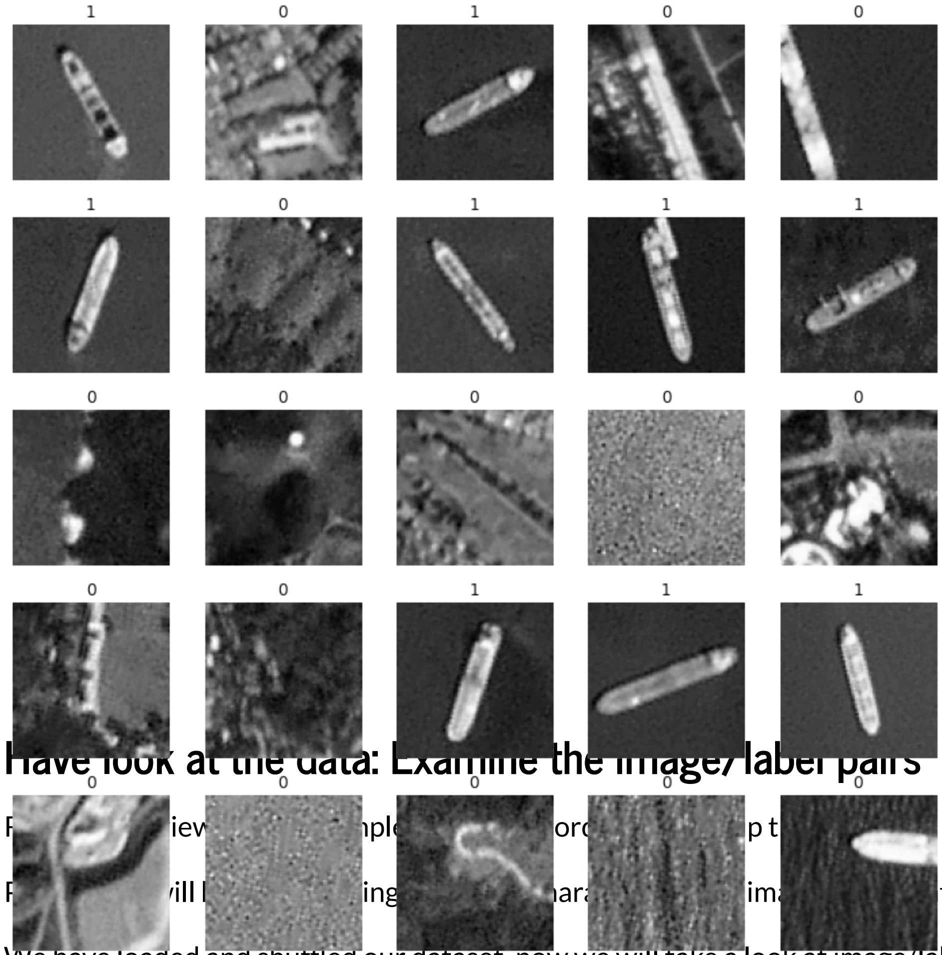
In [8]:
```
print("Original shape of data: ", data.shape)

w = (.299, .587, .114)
data_bw = np.sum(data *w, axis=3)

print("New shape of data: ", data_bw.shape)

data_orig = data.copy()
```

```
Original shape of data:  (4000, 80, 80, 3)
New shape of data:  (4000, 80, 80)
```

In [9]:
```
# Visualize the data samples
helper.showData(data_bw[:25], labels[:25], cmap="gray")
```

Out[9]:





Have look at the data: Examine the image/label pairs

F iew nple ord p t

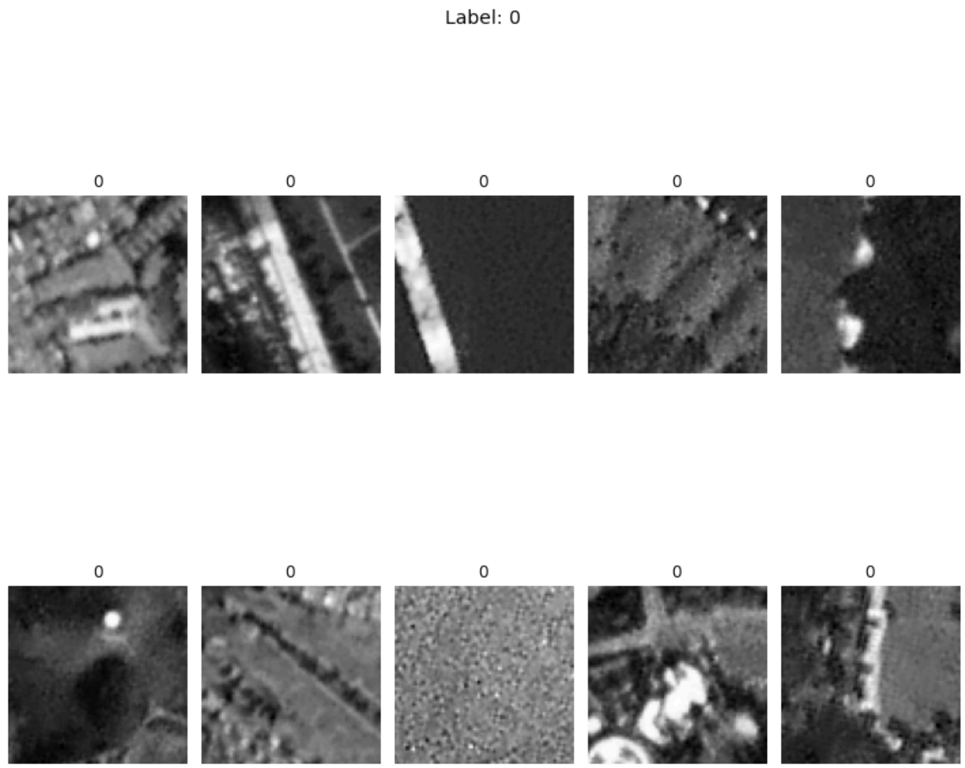
F vill l ing hara ima tain ships.

We have loaded and shuffled our dataset, now we will take a look at image/label pairs.

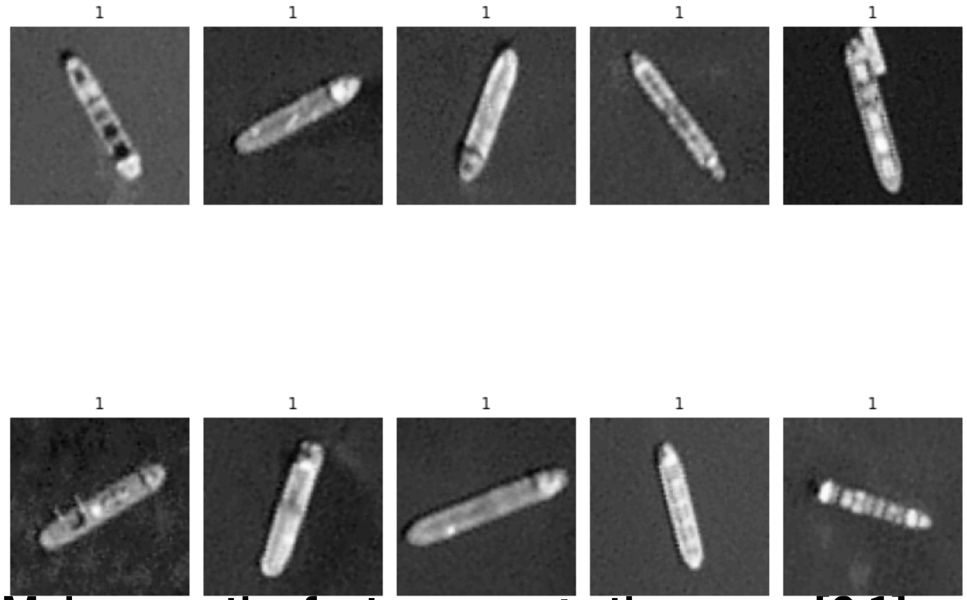
Feel free to explore the data using your own ideas and techniques.

In [10]:

```python
# Inspect some data (images)
num_each_label = 10

for lab in np.unique(labels):
    # Fetch images with different labels
    X_lab, y_lab = data_bw[ labels == lab ], labels[ labels == lab]
    # Display images
    fig = helper.showData( X_lab[:num_each_label], [ str(label) for label in y_lab[:num_each_label] ], cmap="gray")
    _ = fig.suptitle("Label: "+  str(lab), fontsize=14)
    print("\n\n")
```



Label: 1



Make sure the features are in the range [0,1]

Warm up exercise: When we want to train image data, the first thing we usually need to do is scaling.

Since the feature values in our image data are between 0 and 255, to make them between 0 and 1, we need to divide them by 255.

We also need to consider how to represent our target values

- If there are more than 2 possible target values, One Hot Encoding may be appropriate
    - **Hint**: Lookup `tf.keras.utils.to_categorical`
- If there are only 2 possible targets with values 0 and 1 we can use these targets without further encoding

Question

- Set variable X to be our gray-scale examples (`data_bw`), but with values in the range [0,1]
- Set variable y to be the representation of our target values

In [11]:
```
# Scale the data
# Assign values for X, y
#  X: the array of features
#  y: the array of labels
# The length of X and y should be identical and equal to the length of data.
from tensorflow.keras.utils import to_categorical
X, y = np.array([]), np.array([])

# YOUR CODE HERE
raise NotImplementedError()
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
Input In [11], in <cell line: 10>()
      7 X, y = np.array([]), np.array([])
      9 # YOUR CODE HERE
---> 10 raise NotImplementedError()

NotImplementedError:
```

In [ ]:
```
# Check if your solution is right

assert X.shape == (4000, 80, 80)
assert y.shape == (4000,)
```

Split data into training data and testing data

To train and evaluate a model, we need to split the original dataset into a training subset (in-sample) and a test subset (out of sample).

We will do this for you in the cell below.

DO NOT shuffle the data until after we have performed the split into train/test sets

- We want everyone to have the **identical** test set for grading
- Do not change this cell

In [ ]:

```
# Split data into train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.10, rando
m_state=42)

# Save X_train, X_test, y_train, y_test for final testing
if not os.path.exists('./data'):
    os.mkdir('./data')
np.savez_compressed('./data/train_test_data.npz', X_train=X_train, X_test=X_tes
t, y_train=y_train, y_test=y_test)
```

Create a model using only Classification, no data transformation (other than reshaping)

Question: You need to build a 1-layer (head layer only) network model with `tensorflow.keras` . Please name the head layer "dense_head".

Set variable `model0` to be a Keras `Sequential` model object that implements your model.

Hints:

1. Since the dataset is 2-dimensional, you may want to use `Flatten()` in

In [ ]:
```
# Get the number of unique labels
num_cases = np.unique(labels).shape[0]
if num_cases > 2:
    activation = "softmax"
    loss = 'categorical_crossentropy'
else:
    activation = "sigmoid"
    num_cases = 1
    loss = 'binary_crossentropy'

# Set model0 equal to a Keras Sequential model
model0 = None

# YOUR CODE HERE
raise NotImplementedError()

model0.summary()
```

In [ ]:
```
# We can plot our model here using plot_model()
plot_model(model0)
```

Train model

Question:

Now that you have built your first model, you will compile and train it. The requirements are as follows:

- Split the **training** examples `X_train, y_train` again !
    - 80% will be used for training the model
    - 20% will be used as validation (out of sample) examples

In [ ]:
```
model_name0 = "Head only"

# YOUR CODE HERE
raise NotImplementedError()
```

How many weights in the model ?

Question:

Calculate the number of parameters in your model.

Set variable `num_parameters0` to be equal to the number of parameters in your model.

Hint: The model object may have a method to help you ! Remember that Jupyter can help you find the methods that an object implements.

In [ ]:
```
# Set num_parameters0 equal to the number of weights in the model
num_parameters0 = None

# YOUR CODE HERE
raise NotImplementedError()

print("Parameters number in model0: ", num_parameters0)
```

Evaluate the model

Question:

We have trained our model. We now need to evaluate the model using the test dataset created in an earlier cell.

Please store the model score in a variable named `score0` .

Hint: The model object has a method `evaluate`. Use that to compute the score.

In [ ]:
```
score0 = []

# YOUR CODE HERE
raise NotImplementedError()

print("{n:s}: Test loss: {l:3.2f} / Test accuracy: {a:3.2f}".format(n=model_name
0, l=score0[0], a=score0[1]))
```

Save the trained model0 and history0 for submission

Your fitted model can be saved for later use

- In general: so you can resume training at a later time
- In particular: to allow us to grade it !

Execute the following cell to save your model, which you will submit to us for grading.

In [ ]:
```
helper.saveModel(model0, model_name0)
helper.saveHistory(history0, model_name0)
```

Question:

Make sure that the saved model can be successfully restored.

- Set variable `model_loss` to the value of the loss parameter you used in the `compile` statement for your model
- Set variable `model_metrics` to the value of the metrics parameter you used in the `compile` statement for your model

In [ ]:
```
## Restore the model (make sure that it works)

model_loss=None
model_metrics=None

# YOUR CODE HERE
raise NotImplementedError()

model_loaded = helper.loadModel(model_name0, loss=model_loss, metrics=model_metr
ics)
score_loaded = model_loaded.evaluate(X_test, y_test, verbose=0)

assert score_loaded[0] == score0[0] and score_loaded[1] == score0[1]
```

Create a new model with an additional Dense layer

Question:

We will add more layers to the original model0.

- You need to add **AT LEAST ONE** Dense layer followed by an activation function (for example, ReLU)
    - You can add more layers if you like
- The number of units in your very **FIRST** Dense layer should be equal to the value of variable `num_features_1`, as set below.
    - Please name this Dense layer "dense_1" and the head layer "dense_head".

Hints:

- Don't forget to flatten your input data!
- A Dropout layer maybe helpful to prevent overfitting and accelerate your training process.
    - If you want to use a Dropout layer, you can use `Dropout()`, which is in `tensorflow.keras.layers`.

Hopefully your new model performs **better** than your first.

In [ ]:
```
# Set model1 equal to a Keras Sequential model
model1 = None
num_features_1 = 32

# YOUR CODE HERE
raise NotImplementedError()

model1.summary()
```

In [ ]:
```
# Plot your model
plot_model(model1)
```

Train your new model

Question:

Now that you have built your new model1, you will compile and train model1. The requirements are as follows:

- Split the **training** examples `X_train, y_train` again!
    - 80% will be used for training the model
    - 20% will be used as validation (out of sample) examples
    - Use `train_test_split()` from `sklearn` to perform this split
        - Set the `random_state` parameter of `train_test_split()` to be 42
- Loss function and Metric as per first model's instructions.
- Use exactly 15 epochs for training
- Save your training results in a variable named `history1`
- Plot your training results using the`plotTrain` method described in the Student API above.

In [ ]:
```
# Train the model using the API
model_name1 = "Dense + Head"

# YOUR CODE HERE
raise NotImplementedError()
```