Homework 2

RoboCup@Home Object Classification

Martina Evangelisti 1796480

December 15, 2020

Contents

1 Introduction

The goal of the learning algorithm described in this report is to classify images regarding objects in a home environment.
The report will discuss two different solutions based on neural networks: the first one is obtained using a neural network designed from the scratch to solve this problem and trained on the given dataset; the second one is based on transfert learing starting from a pre-trained model.

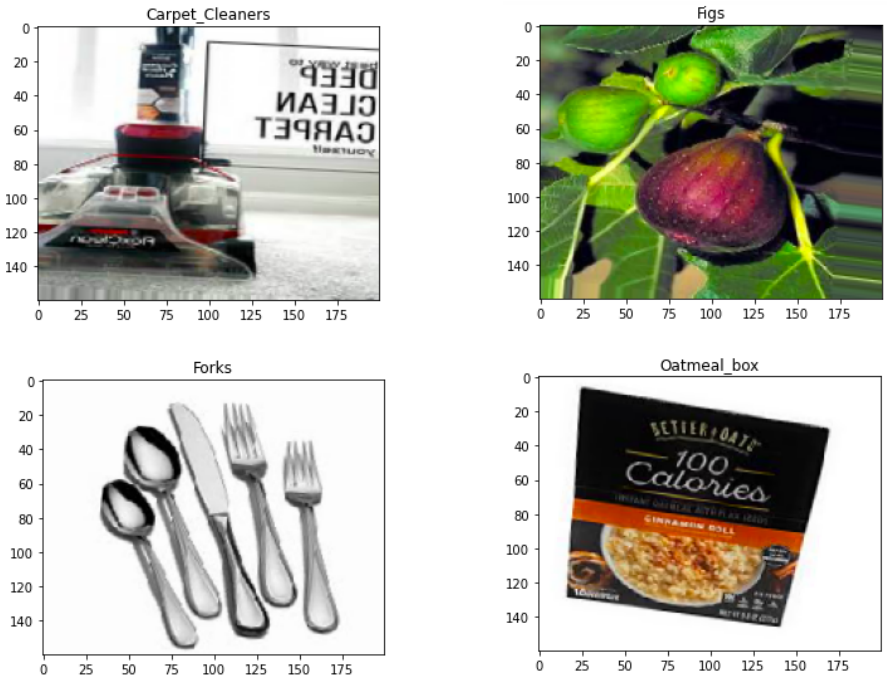
2 Dataset

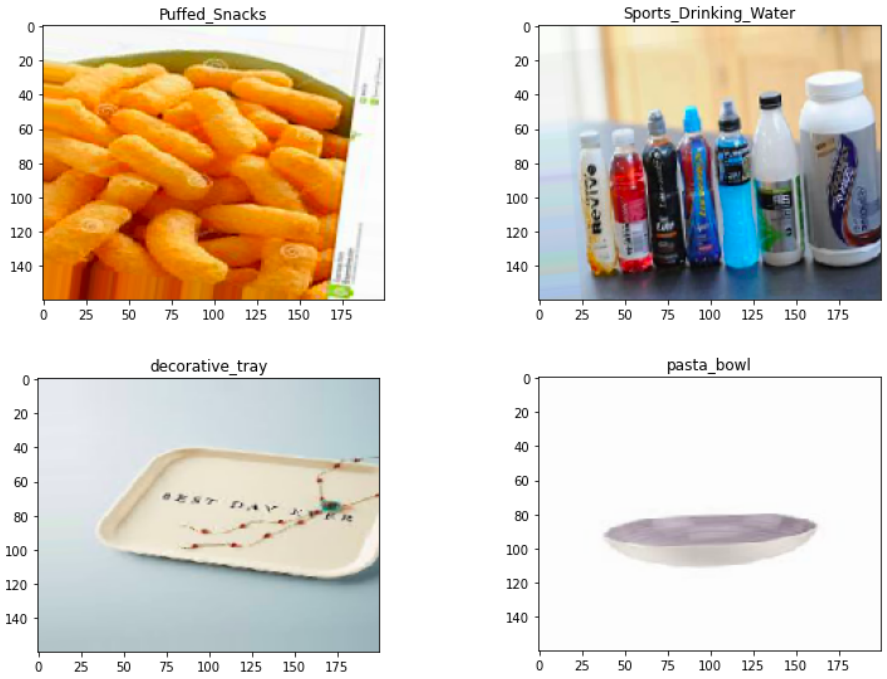
The Dataset is taken from the RoboCup@Home-Objects dataset that has been developed within the RoboCup@Home competition. The solution described in this report provides a solution for a 8-class classification of objects of the following categories:

- Carpet Cleaner
- Forks
- Puffed Snacks
- Decorative tray
- Figs
- Oatmeal box
- Sports Drinking Water
- Pasta bowl

.

Examples of images in the dataset:





2.1 Preprocessing

The original dataset contains 8949 images belonging to the previously described classes. The dataset is organized in 8 folders, one for each class, containing the images belonging to the class from which the directory takes the name.

```
.
├── Carpet_Cleaners
├── Figs
├── Forks
├── Oatmeal_box
├── Puffed_Snacks
├── Sports_Drinking_Water
├── decorative_tray
└── pasta_bowl

8 directories
```

Figure 1: Original Dataset Tree

I split the dataset into 3 different sets: training set, validation set and test set. The first two sets are used in the training phase: the training set to train the Neural Network and the validation set to validate it step by step during the training. The test set is used to evaluate the trained model. I let the 70% of the dataset be part of the training set and then 15% for validation and 15% for testing.

I used `splitfolders` to do this separation and the output obtained is organized as it follows:

```
.
├── test
│   ├── Carpet_Cleaners
│   ├── Figs
│   ├── Forks
│   ├── Oatmeal_box
│   ├── Puffed_Snacks
│   ├── Sports_Drinking_Water
│   ├── decorative_tray
│   └── pasta_bowl
├── train
│   ├── Carpet_Cleaners
│   ├── Figs
│   ├── Forks
│   ├── Oatmeal_box
│   ├── Puffed_Snacks
│   ├── Sports_Drinking_Water
│   ├── decorative_tray
│   └── pasta_bowl
└── val
    ├── Carpet_Cleaners
    ├── Figs
    ├── Forks
    ├── Oatmeal_box
    ├── Puffed_Snacks
    ├── Sports_Drinking_Water
    ├── decorative_tray
    └── pasta_bowl

27 directories
```

Figure 2: New Dataset Tree

The train set contains 6259 images, the validation set 1338 and the test set 1352. With `ImageDataGenerator` i re-scaled all the images and i preprocessed the Train set adding some **augmentation**: horizontal flip to randomly flip images, random zoom, 10 degree of rotation range for random rotations and height and width shift. Once instantiated the `ImageDataGenerator` both for the train and for the validation set, i loaded the images using `flow_from_directory` method.

3 CNN: Convolutional Neural Network

The first model i designed is a convolutional neural network, a class of neural networks characterized by the presence of at least one convolutional layer. A convolutional layer is composed by 3 phases: convolutions between input and kernel, non-linear activation function and pooling.

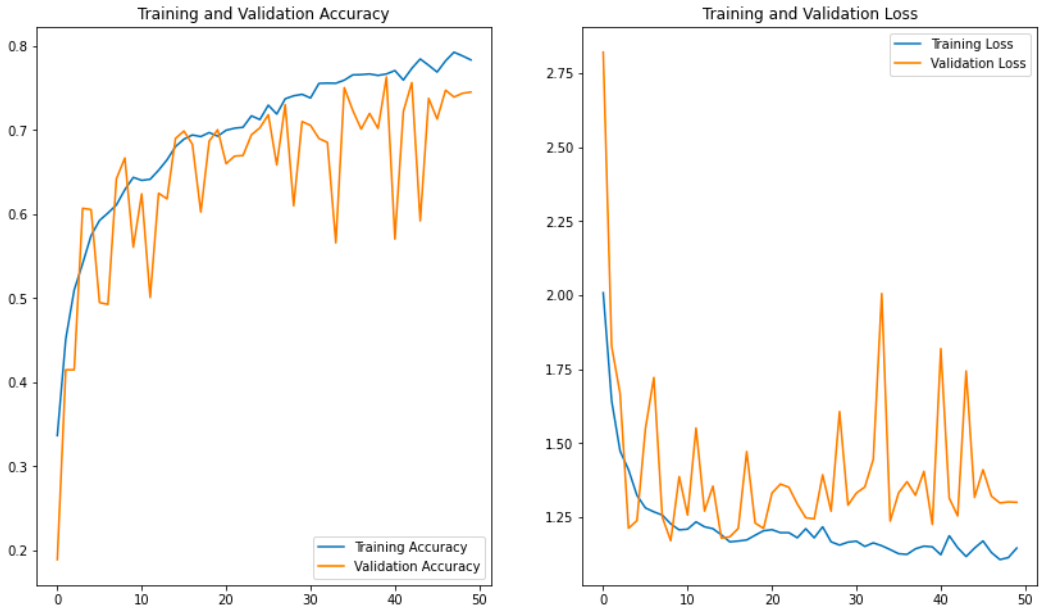
The model i created is sequential with 17 layers:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 160, 200, 32)      2432
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 80, 100, 32)       0
_________________________________________________________________
batch_normalization (BatchNo (None, 80, 100, 32)       128
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 80, 100, 64)       51264
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 80, 100, 64)       36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 26, 33, 64)        0
_________________________________________________________________
dropout (Dropout)            (None, 26, 33, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 22, 29, 96)        153696
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 11, 14, 96)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 11, 14, 96)        384
_________________________________________________________________
flatten (Flatten)            (None, 14784)             0
_________________________________________________________________
dense (Dense)                (None, 256)               3784960
_________________________________________________________________
activation (Activation)      (None, 256)               0
_________________________________________________________________
dropout_1 (Dropout)          (None, 256)               0
_________________________________________________________________
batch_normalization_2 (Batch (None, 256)               1024
_________________________________________________________________
dense_1 (Dense)              (None, 8)                 2056
_________________________________________________________________
activation_1 (Activation)    (None, 8)                 0
=================================================================
Total params: 4,032,872
Trainable params: 4,032,104
Non-trainable params: 768
_________________________________________________________________
```

I used 4 convolutional layers with different kernel sizes and shape, each of them with 'RELU' activation function. The third one has kernel size equal to (3,3), all the others to (5,5).The first one has 32 kernels, the second and third 64 and the forth 96. The fist `conv2D` is followed by `MaxPool2D` layer with pool size and strides equal to (2,2) and padding "valid". The second and the third convolutional layers are sequential and they're followed by `MaxPool2D` with stride and pool size (3,3). Last Max pooling layer is equal to the first one. The `Dropout` layers are used in order to reduce the overfitting, `Batch normalization` normalizes the input and increases the stability of the network. Moreover there are a `flatten` layer and two `dense` layers. The last one with 'softmax' as activation function and the dimension equal to the number of classes.

3.1 Training

I trained the model with model `fit` with a batch size equal to 32 and steps per epoch equal to the number of samples in the training set divided by the batch size, and validation steps equal to number of samples in the validation set divided by the batch size + 1.
I trained this model for 50 epochs.



We can observe that in the plot regarding the accuracy the performances of the model on the training set are always increasing while on the test set there is an oscillation that, in the end, stabilizes at the value of about 0.72. The same observations can be done for the loss in a symmetric way.

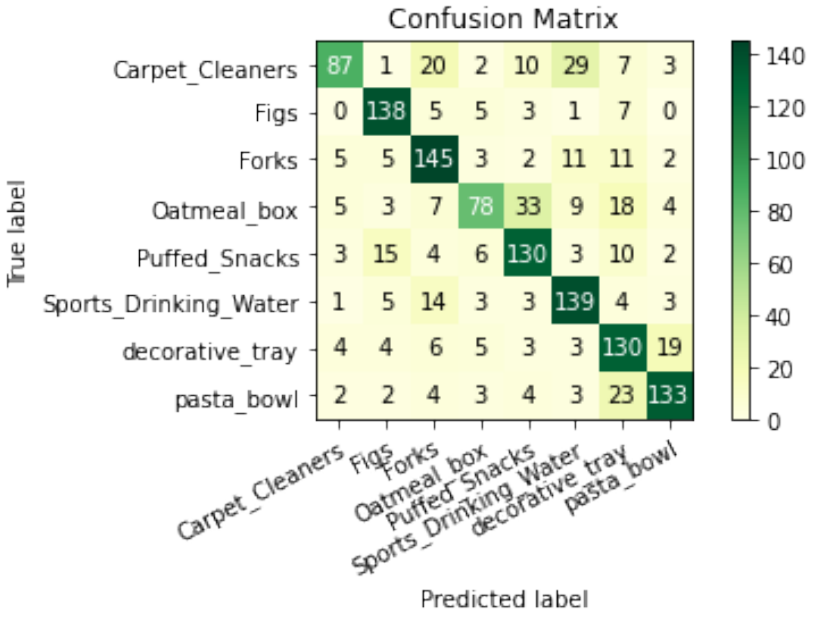
3.2 Prediction and Evaluation

In order to make prediction on new samples i loaded the test set using the `test generator` previously used for the validation set and `flow_from_directory` from test directory, obtaining an accuracy of 0.724.

```
Found 1352 images belonging to 8 classes.
43/43 [==============================] - 24s 547ms/step - loss: 1.3556 - accuracy: 0.7249
Test loss: 1.355571
Test accuracy: 0.724852
```

To see the distribution of the errors over the classes and the specific values of each prediction, here are the classification report and the confusion matrix.

	precision	recall	f1-score	support
Carpet_Cleaners	0.813	0.547	0.654	159
Figs	0.798	0.868	0.831	159
Forks	0.707	0.788	0.746	184
Oatmeal_box	0.743	0.497	0.595	157
Puffed_Snacks	0.691	0.751	0.720	173
Sports_Drinking_Water	0.702	0.808	0.751	172
decorative_tray	0.619	0.747	0.677	174
pasta_bowl	0.801	0.764	0.782	174
accuracy			0.725	1352
macro avg	0.734	0.721	0.720	1352
weighted avg	0.733	0.725	0.721	1352



4 Transfer Learning

Transfer learning concept is about exploiting the learning obtained from a different task to use it as the starting point for a model interested in learning a different task. In particular i implemented the Fine Tuning technique setting to trainable only the parameters of the last convolutional layer of VGG16.

4.1 VGG16

I loaded VGG16 pre-trained model setting the weights of this model trained with *imageNet*, which is a visual database made of over 15 millions labeled images of about 22000 classes, designed for use in visual object recognition software research.

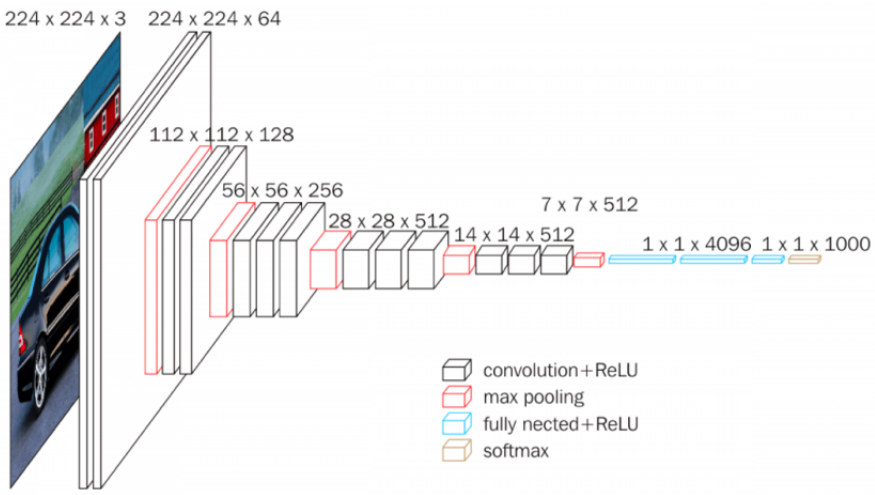


Figure 3: VGG16 architecture

VGG16 is a convolutional neural network with 19 layers, which achieves 92.7% top-5 test accuracy in ImageNet.

4.2 Build the Model

After loading VGG16 pre-trained model, i set all its layers to not be trainable except the last convolutional layer `block5_conv3`. Then i flattened the output of this layer and added 3 Dense Layers, the last one with the dimension of the number of classes that i have to classify and *softmax* as activation function. Here is the structure of the added layers:

```
_________________________________________________________________
flatten_3 (Flatten)          (None, 15360)             0
_________________________________________________________________
batch_normalization_12 (Batc (None, 15360)             61440
_________________________________________________________________
dropout_6 (Dropout)          (None, 15360)             0
_________________________________________________________________
dense_9 (Dense)              (None, 128)               1966208
_________________________________________________________________
batch_normalization_13 (Batc (None, 128)               512
_________________________________________________________________
dropout_7 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_10 (Dense)             (None, 92)                11868
_________________________________________________________________
batch_normalization_14 (Batc (None, 92)                368
_________________________________________________________________
batch_normalization_15 (Batc (None, 92)                368
_________________________________________________________________
dense_11 (Dense)             (None, 8)                 744
=================================================================
Total params: 16,756,196
Trainable params: 4,369,972
Non-trainable params: 12,386,224
_________________________________________________________________
```

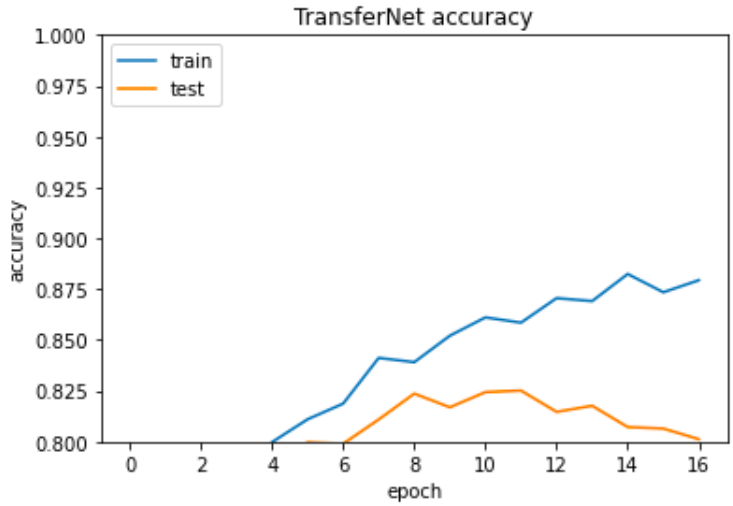
The model obtained has 4,369,972 trainable parameters belonging to last conv layer of VGG16 and to the layers added at the end. All the remaining layers are not trainable and keep the weights computed after the training with imageNet.

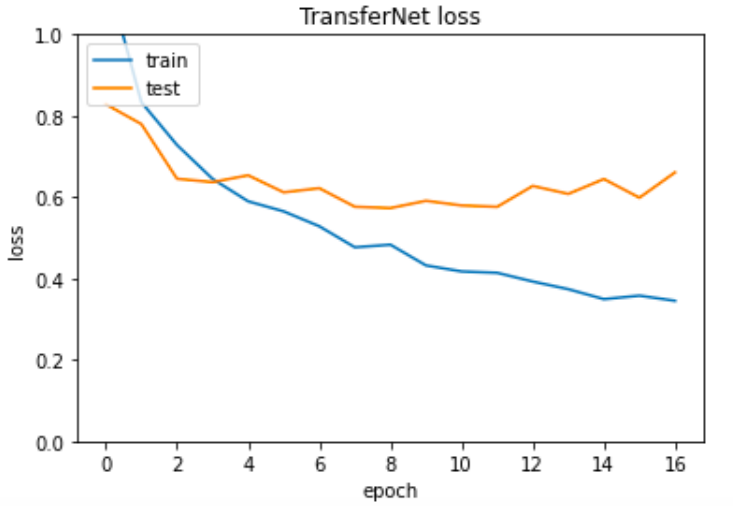
4.3 Training

The Dataset is loaded in the same way as done for the CNN network designed before. The batch size is set to 32 and the number of epochs to 25. In order to avoid overfitting i used the early stopping technique with patience set to 5. In fact, even if the epochs were 25 the training phase stopped after epoch 17. In fact, as seen in the figure, the accuracy was no more improving.

```
Epoch 15/25
195/195 [==============================] - 1252s 6s/step - loss: 0.3493 -
accuracy: 0.8824 - val_loss: 0.6446 - val_accuracy: 0.8072
Epoch 16/25
195/195 [==============================] - 1250s 6s/step - loss: 0.3578 -
accuracy: 0.8735 - val_loss: 0.5987 - val_accuracy: 0.8064
Epoch 17/25
195/195 [==============================] - 1255s 6s/step - loss: 0.3452 -
accuracy: 0.8794 - val_loss: 0.6610 - val_accuracy: 0.8012
```

I showed 2 graphics to evaluate the progress of the accuracy and the loss during the epochs both for the training set and for the validation set.





We can note that while the accuracy of the training was improving, the test set was almost stable around the value of 0.8, the same is for the loss that was decreasing in the train set while it was almost 0.65 for the validation set.

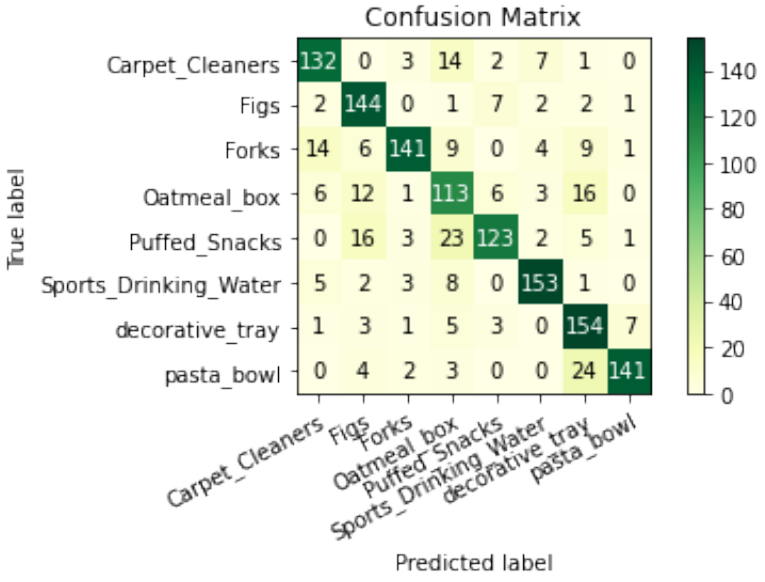
4.4 Prediction and Evaluation

I loaded the test set in order to evaluate the performances of the model on a new set of images, obtaining an accuracy of 0.814.

```
Found 1352 images belonging to 8 classes.
43/43 [==============================] - 247s 6s/step - loss: 0.6583 -
accuracy: 0.8143
Test loss: 0.658250
Test accuracy: 0.814349
```

Here are the classification report and the confusion matrix.

	precision	recall	f1-score	support
Carpet_Cleaners	0.825	0.830	0.828	159
Figs	0.770	0.906	0.832	159
Forks	0.916	0.766	0.834	184
Oatmeal_box	0.642	0.720	0.679	157
Puffed_Snacks	0.872	0.711	0.783	173
Sports_Drinking_Water	0.895	0.890	0.892	172
decorative_tray	0.726	0.885	0.798	174
pasta_bowl	0.934	0.810	0.868	174
accuracy			0.814	1352
macro avg	0.822	0.815	0.814	1352
weighted avg	0.826	0.814	0.816	1352



5 Conclusions

This report presented two different solution to solve an 8-class classification problem of images representing home objects.
I compared two neural networks: one designed and trained on the given dataset and another which uses tranfer learning exploiting the learn of a pre-trained model.
In the evaluation phases we can observe that with Tranfer Learning we reached higher values of accuracy and lower values of loss.
Transfer learning is a good solution if the images that we want to classify are "similar" to the ones used to train the chosen pre-trained model. But this isn't always possible so transfer learning does not always provide a solution better than the one obtained from a new neural network designed and trained on the dataset of the specific problem.
We can note that in both the neural networks we don't reach high results regarding accuracy, this is also because the dataset has some images that are not representative of the class to which they belong to.

References

[1] Tensorflow documentation
`https://www.tensorflow.org/api_docs/python/tf/all_symbols`

[2] Keras documentation
`https://keras.io/api/`

[3] VGG16 Net
`https://neurohive.io/en/popular-networks/vgg16/`

[4] SKLearn: Skikit learn documentation
`https://scikit-learn.org/stable/modules/classes.html`

[5] Split Folders
`https://pypi.org/project/split-folders/`