Sapienza University of Rome

Machine Learning

Homework 2: Venice Boat Classification

Ahmed Tarik Mohamed Elhelow

1823229

Introduction

The goal of this project is to train a classifier in order to classify different categories of boats navigating in the City of Venice (Italy). The dataset that is being used here is the MarDCT from the Sapienza University of Rome. The training dataset contains 4.774 images and the testing dataset contains 1.969 from 24 different categories.

For the sake of simplification, only three classes have been considered. They are the three most common classes between the training dataset and the testing dataset.

Two different model have been used. The first one is a pre-trained version of VGG-16, then it have been re-trained partially with a technique called fine-tuning on the MarDCT dataset. Colored images have been used for training this model. The resulting network is able to identify boats of the 3 different classes with an accuracy of 55% on the training set and an accuracy of 52% on the testing dataset. The project has been developed in Python. Keras framework for deep learning have been used for this part of the project since it contains a straight forward method for the pre-trained version of VGG-16.

The second model is a a Convolutional Neural Network which has been built using tensorflow framework. Gray images have been used for training this model. The resulting network is able to identify boats of the 3 different classes with an accuracy of 97% on the training set and an accuracy of 91.4% on the testing dataset, which is much better than the first model.

Data Preprocessing

Due to the differences between the classes in the Training and Testing sets and due to the fact that the training phase will run on a limited memory, this paper chose to find the 3 common classes that have the highest amount of images in both the Training and Testing datasets. This filtering of the data yielded a total of 2311 images for Training and 816 images for Testing, and the 3 classes with the highest amount of images were found to be:

1. Lanciafino 10m Bianca
2. Mototopo
3. Vaporetto ACTV

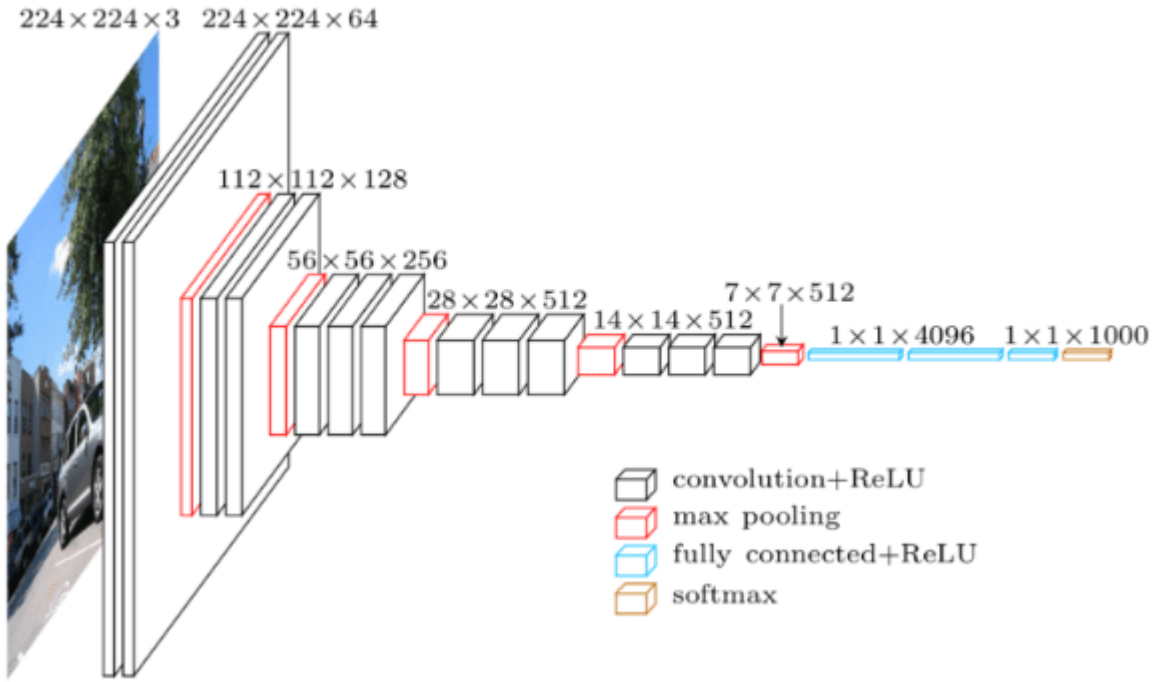
Moreover, images in the dataset were resized to 224 by 224 and kept the color scale for VGG-16 model, however they were resized to 120 by 120 and converted to gray scale for the Convolutional Neural Network model.

VGG-16 and Fine-Tuning (The First Model)

As said before modern frameworks provide pretrained models for a bunch of different networks. One of this is VGG-16, a 16-layer convolutional neural network developed at Oxford University during the 2014 ILSVRC (ImageNet) competition.

This network is composed of a sequence of convolutional layers with ReLU activation function followed by max pooling layers, as shown in the following figure, followed in the end by three fully connected layer and a

softmax layer which is the output layer containing the probabilites of the classes.



The fine-tuning technique consists in loading a network pre-trained on a very large dataset (like ImageNet), replacing the softmax layer with a softmax layer of the correct size and train the network on another dataset, in this case MarDCT, while keeping constant all the weights except the ones of the last layer.

The training has been performed on Google Compute Engine which provides high performances GPUs. Here an NVIDIA Tesla K80 has been used to train the network for 5 epochs on the training set using stochastic gradient descent with Nesterov momentum and weight decay obtaining an accuracy of 55%. On the test set the accuracy obtained is 52%. The learning rate has been set to 0.001, the momentum to 0.9 and the weight decay to 10^−6 .

To have a comparison on the importance of using the GPU, on the NVIDIA Tesla K80 the training has been completed is less than 10 minutes, while on a high end notebook on a single CPU the training time requires more than 5 hours for a network of this size on MarDCT.

The CNN Model Implementation (The Second Model)

A Convolutional Neural Network was implemented using the open-source libraries TF Learn and TensorFlow. TF Learn was chosen as it is a minimal abstraction layer on top of TensorFlow, so it offers a simple interface for building a CNN. TF Learn does not abstract TensorFlow radically or immensely, which makes building a network the matter of a few lines and keeps all of TensorFlow's functionality readily available for use.

The Network consisted of the following layers:

- 1 Input Layer with size 224x224x1
- 6 Convolutional Layers and Max Pooling Layers
- 1 Fully Connected Layer with 1024 units
- 1 Output Layer with 3 units corresponding to the 3 classes

The activation function used in the 6 Convolutional Layers is ReLU as it does not saturate and guarantees that learning will commence. For the output layer however, since it is a multi-class classification problem, the Softmax activation function was used.

In the first few experiments, the learning rate was set to 0.01. This yielded fast learning but horrible accuracy of around 30%. Consequently, the learning rate was adjusted to 0.001 to increase the chances of gradient descent finding a local minimum.

Development

Below the development of the project is detailed, commenting each of the functions and steps given to achieve the desired goals for the project.

Import necessary libraries for the project

In the following block is listed all the necessary libraries for the project, the seed for the random numbers and the path for the folder containing.

```
# Difine libraries
import cv2
import numpy as np
import os
import matplotlib.pyplot as plt
import sklearn
from sklearn import preprocessing, metrics
from sklearn.metrics import confusion_matrix
import random
from random import shuffle
from heapq import nlargest

# Importing and mount google drive
from google.colab import drive
drive.mount('/content/gdrive',force_remount=False)

# Importing TF Learn
import tensorflow as tf
import tflearn
from tflearn.layers.conv import conv_2d, max_pool_2d
from tflearn.layers.core import input_data, dropout, fully_connected
from tflearn.layers.estimator import regression

# Importing keras
import keras
```

Output:

```
Drive already mounted at /content/gdrive; to attempt to forcibly
remount, call drive.mount("/content/gdrive", force_remount=True).
Using TensorFlow backend.
```

Define the Paths to the relevant Data Sets and Files

We have to define exactly the paths for the training and testing datasets and related files

```
# Set seed for random numbers
random.seed(1)

# Define directories and required files
DIR = "/content/gdrive/My Drive/"
TRAIN_DIR = "/content/gdrive/My Drive/sc5/"
DB_info = "/content/gdrive/My Drive/sc5/DBinfo.txt"
TEST_DIR = "/content/gdrive/My Drive/sc5-test/"
GROUND_TRUTH = "/content/gdrive/My Drive/sc5-test/ground_truth.txt"

# Define the image size and learning rate for the CNN
IMG_WIDTH_COLOR = 224
IMG_HEIGHT_COLOR = 224
IMG_WIDTH_GRAY = 120
IMG_HEIGHT_GRAY = 120
LEARN_RATE = 0.001
```

Explore the MarDCT Database

```
# Explore the Boats Training Data Set to check for the available classes
training_classes = [f.name for f in os.scandir(TRAIN_DIR) if f.is_dir()]

# Explore the Boats Testing Data Set to check for the available classes
test_data_dict = dict()
freq_of_each_class = dict()
classes = list()
testing_classes = list()

# Opening the Ground Truth File and Reading its contents
with open(GROUND_TRUTH, 'r') as f:
    f_content = f.read().splitlines()
for line in f_content:
    filename, label = line.split(";")
    label = label.replace(" ", "")  # Format the class label by removing spaces
    label = label.replace(":", "")  # Format the class label by removing colons
    classes.append(label)
    if label not in testing_classes:
        testing_classes.append(label)

# Count the frequency of occurrence of each class/label in the Testing Data Set
for cls in classes:
    freq_of_each_class[cls] = freq_of_each_class.get(cls, 0) + 1

print("freq_of_each_class: \n", freq_of_each_class)
```

Output:

```
freq_of_each_class:
{'SnapshotAcqua': 420, 'Motobarca': 59, 'VaporettoACTV': 325,
'Mototopo': 274, 'Patanella': 74, 'SnapshotBarcaMultipla': 171,
'MotoscafoACTV': 1, 'Lanciafino10mBianca': 217, 'SnapshotBarcaParziale':
116, 'Lanciafino10mMarrone': 125, 'Mototopocorto': 10,
'Lanciamaggioredi10mBianca': 6, 'Topa': 29, 'Barchino': 51,
'Raccoltarifiuti': 19, 'Ambulanza': 22, 'Alilaguna': 19, 'Polizia': 15,
```

```
'Lanciafino10m': 7, 'Motopontonerettangolare': 3, 'Gondola': 3,
'Sandoloaremi': 3}
```

Find the Common Classes between the MarDCT Testing and Training Data Sets with the most images

The 3 classes with the highest amount of images were found to be:

1. Lanciafino 10m Bianca
2. Mototopo
3. Vaporetto ACTV

```
# Find the Common Classes
common_classes = [element for element in training_classes if element in \
                  testing_classes]

freq_of_common_class = dict()
for cls in classes:
    if cls in common_classes:
        freq_of_common_class[cls] = freq_of_common_class.get(cls, 0) + 1

# Pick 3 Classes with the most number of images from the common Classes
NUM_CLASSES = 3
counts = nlargest(NUM_CLASSES, freq_of_common_class.values())
classes_to_be_considered = [key for key, value in \
                            freq_of_common_class.items() if value in counts]

# Transform labels from a list of strings to a list of Numbers
numerical_form_classes = np.asarray([classes_to_be_considered.index(t) \
                                     for t in classes_to_be_considered])
classes_onehot = preprocessing.OneHotEncoder(sparse=False). \
                 fit_transform(numerical_form_classes.reshape(-1, 1))

# Dictionary with Class name and Corresponding label
corres_label = dict(zip(classes_to_be_considered, classes_onehot))
print("Dictionary with Class name and Corresponding label: \n", corres_label)
```

Output:

```
Dictionary with Class name and Corresponding label:
{'VaporettoACTV': array([1, 0, 0]), 'Mototopo': array([0, 1, 0]),
'Lanciafino10mBianca': array([0, 0, 1])}
```

Build the Training Set that we will use to train and test the CNN

In order to create the dataset, first it reads image by image from the folders, resize (because the images are really big for a good classification) and append them to the training vector. Images were resized to 224 by 224 and kept the color scale for VGG-16 model, however they were resized to 120 by 120 and converted to gray scale for the Convolutional Neural Network model.

```
X_color = []
```

```
X_gray = []
Y = []

for folder in training_classes:
    if folder in classes_to_be_considered:
        path = TRAIN_DIR + folder
        files = os.listdir(path)
        for img in files:
#             print(":", end='')
            label = corres_label[folder]
            img_path = os.path.join(path, img)
            img = cv2.resize(cv2.imread(img_path, cv2.IMREAD_UNCHANGED), \
                             (IMG_HEIGHT_COLOR, IMG_WIDTH_COLOR))
            img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
            img_gray = cv2.resize(img_gray,(IMG_HEIGHT_GRAY, IMG_WIDTH_GRAY))
            X_color.append(img)
            X_gray.append(img_gray)
            Y.append(label)
#         print()

training_data = list(zip(X_color, X_gray, Y))
shuffle(training_data)
X_color, X_gray, Y = zip(*training_data)
```

Build the Testing Set that we will feed the CNN

The process to create the test dataset is basically the same as the training dataset, but here it is a little bit different because the files are all together, not divided by folders and the labels are in a txt file with names in different formats from the folders names used above. Also, testing images were resized to 224 by 224 and kept the color scale for VGG-16 model, and they were resized to 120 by 120 and converted to gray scale for the Convolutional Neural Network model.

```
# Build the Testing Data Set
considered_test_filenames = list()
considered_test_labels = list()

with open(GROUND_TRUTH, 'r') as f:
    f_content = f.read().splitlines()
for line in f_content:
    filename, label = line.split(";")
    label = label.replace(" ", "")  # Format the class label by removing spaces
    label = label.replace(":", "")  # Format the class label by removing colons
    if label in classes_to_be_considered:
        considered_test_filenames.append(filename)
        considered_test_labels.append(corres_label[label])

testing_corres_label = dict(zip(considered_test_filenames, \
                                considered_test_labels))

# Building the model's testing data set
x_test_color = []
x_test_gray = []
y_test = []
for test_img in os.listdir(TEST_DIR):
    if test_img == "Thumbs.db":
        continue
    if test_img in testing_corres_label.keys():
#         print(":", end='')
        test_data_label = testing_corres_label[test_img]
        test_img_path = os.path.join(TEST_DIR, test_img)
        test_img = cv2.resize(cv2.imread(test_img_path, cv2.IMREAD_UNCHANGED) \
                              ,(IMG_HEIGHT_COLOR, IMG_WIDTH_COLOR))
        test_img_gray = cv2.cvtColor(test_img, cv2.COLOR_BGR2GRAY)
        test_img_gray = cv2.resize(test_img_gray,(IMG_HEIGHT_GRAY, \
```

```
                                                    IMG_WIDTH_GRAY))
        x_test_color.append(test_img)
        x_test_gray.append(test_img_gray)
        y_test.append(test_data_label)
# print()

testing_data = list(zip(x_test_color, x_test_gray, y_test))
shuffle(testing_data)
x_test_color, x_test_gray, y_test = zip(*testing_data)
```

```
# Prepare the Testing and Training Data that will be used to train the Network
X_color = np.array(X_color)
X_gray = np.array(X_gray).reshape(-1, IMG_WIDTH_GRAY, IMG_HEIGHT_GRAY, 1)
Y = np.array(Y)

x_test_color = np.array(x_test_color)
x_test_gray = np.array(x_test_gray).reshape(-1,IMG_WIDTH_GRAY, \
                                            IMG_HEIGHT_GRAY, 1)
y_test = np.array(y_test)
```

Obtaining the FIRST MODEL (VGG-16) and Reconfigure it

Keras framework for deep learning have been used for this part since it contains a straight forward method for the pre-trained version of VGG-16.

```
# Load pretrained VGG-16 model:
vgg16_tmp = keras.applications.vgg16.VGG16()

# Build a new model similar to the previous one with the right number
# of nodes on the softmax layer and make all the other layer not trainable:
vgg16_model = keras.models.Sequential()

for layer in vgg16_tmp.layers:
    vgg16_model.add(layer)

vgg16_model.layers.pop()

for layer in vgg16_model.layers:
    layer.trainable = False

vgg16_model.add(keras.layers.Dense(NUM_CLASSES, activation="softmax"))
```

Output:

```
Downloading data from https://github.com/fchollet/deep-learning-
models/releases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels.h
5

553467904/553467096 [==============================] - 7s 0us/step
```

Train the VGG-16 model

Also Keras framework have been used for training the VGG-16 model. The network has been trained for 5 epochs on the training set using stochastic gradient descent with Nesterov momentum and weight decay

obtaining an accuracy of 55%. On the test set the accuracy obtained is 52%. The learning rate has been set to 0.001, the momentum to 0.9 and the weight decay to 10^−6 .

```
# Train the network with SGD with momentum:
vgg16_model.compile(keras.optimizers.SGD(lr=1e-3, decay=1e-6,
                                         momentum=0.9, nesterov=True),
                    loss="categorical_crossentropy",
                    metrics=["accuracy"])

vgg16_model.fit(X_color, Y, batch_size=32, epochs=5)
```

Output:

```
Epoch 1/5
2311/2311 [==============================] - 1279s 553ms/step - loss:
1.0977 - acc: 0.3522
Epoch 2/5
2311/2311 [==============================] - 1263s 547ms/step - loss:
1.0784 - acc: 0.5206
Epoch 3/5
2311/2311 [==============================] - 1262s 546ms/step - loss:
1.0637 - acc: 0.5370
Epoch 4/5
2311/2311 [==============================] - 1260s 545ms/step - loss:
1.0518 - acc: 0.5439
Epoch 5/5
2311/2311 [==============================] - 1263s 546ms/step - loss:
1.0419 - acc: 0.5504
```

Test the model

For the training set it is possible to obtain a higher accuracy, however, it is necessary to test in images never seen before to get the real accuracy of the model.

```
y_pred = vgg16_model.predict(x_test_color)
y_pred = np.argmax(y_pred, axis=1)
y_test_orig = np.argmax(y_test, axis=1)

print("Accuracy: " + str(sklearn.metrics.accuracy_score(y_test_orig, y_pred)))
print("Confusion matrix: \n", confusion_matrix(y_test_orig, y_pred))
```

Output:

```
Accuracy: 0.5208333333333334

Confusion matrix:
 [[323   2   0]
  [172 102   0]
  [156  61   0]]
```

It is clear that 52% is not a very great result. Now we will try the second model.

Build the second model

It is Convolutional Neural Network using the open-source libraries TF Learn and TensorFlow that has:

- 1 Input Layer with size 224x224x1
- 6 Convolutional Layers and Max Pooling Layers
- 1 Fully Connected Layer with 1024 units
- 1 Output Layer with 3 units corresponding to the 3 classes

The learning rate was adjusted to 0.001 to increase the chances of gradient descent finding a local minimum.

```
tf.reset_default_graph()

# Model 2
# Build the model and train it
convnet = input_data(shape=[None, IMG_WIDTH_GRAY, IMG_HEIGHT_GRAY, 1], \
                     name='input')

convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = conv_2d(convnet, 32, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

convnet = conv_2d(convnet, 64, 2, activation='relu')
convnet = max_pool_2d(convnet, 2)

# You have 2 fully connected layers here
# One is the fully connected and the other is the o/p layer
convnet = fully_connected(convnet, 1024, activation='relu')
convnet = dropout(convnet, 0.8)

convnet = fully_connected(convnet, 3, activation='softmax') # Outplut layer
convnet = regression(convnet, optimizer='adam', learning_rate=LEARN_RATE,
                     loss='categorical_crossentropy', name='targets')

# Experiment changing the learning rate and
model = tflearn.DNN(convnet, tensorboard_dir='log_1')
```

Train the second model

After defining the architecture, it is necessary to really train the classifier. This classifier is using the Adam Optimizer, 5 epochs and a batch size of 1 image.

Below is printed the time, loss and accuracy of each epoch.

```
model.fit({'input': X_gray}, {'targets': Y}, n_epoch=5, \
          validation_set=({'input': x_test_gray}, {'targets': y_test}), \
          snapshot_step=50, show_metric=True, \
          run_id="boatClassifier-6conv-0.001LR")

print("Model 2 Trained!")
```

Output:

```
Training Step: 36  | total loss: 0.54534 | time: 46.793s
| Adam | epoch: 001 | loss: 0.54534 - acc: 0.7374 -- iter: 2304/2311

Training Step: 73  | total loss: 0.26775 | time: 49.171s
| Adam | epoch: 002 | loss: 0.26775 - acc: 0.8969 -- iter: 2304/2311

Training Step: 110  | total loss: 0.24429 | time: 47.943s
| Adam | epoch: 003 | loss: 0.24429 - acc: 0.9156 -- iter: 2304/2311

Training Step: 147  | total loss: 0.10301 | time: 44.581s
| Adam | epoch: 004 | loss: 0.10301 - acc: 0.9534 -- iter: 2304/2311

Training Step: 184  | total loss: 0.07564 | time: 48.279s
| Adam | epoch: 005 | loss: 0.07564 - acc: 0.9718 -- iter: 2304/2311

Model 2 Trained!
```

Test the second model

For the training set it is possible to obtain a higher accuracy, however, it is necessary to test in images never seen before to get the real accuracy of the model.

```
predicted = list()
truth = list()

for _, the_img, the_img_label in testing_data:
    the_image = the_img.reshape(IMG_WIDTH_GRAY, IMG_HEIGHT_GRAY, 1)
    model_out = model.predict([the_image])[0]

    predicted.append(np.argmax(model_out))
    truth.append(np.argmax(the_img_label))

print("The Second Model: ")
print("Accuracy: " + str(sklearn.metrics.accuracy_score(truth, predicted)))
print("Confusion matrix: \n", confusion_matrix(truth, predicted))
```

Output:

```
The Second Model:


Accuracy: 0.9142156862745098
```

```
Confusion matrix:
[[321   4   0]
 [  1 271   2]
 [  0  63 154]]
```

Conclusion

It was noticed during the training and evaluation phases, the accuracy levels reached by the CNN network are too high than the VGG-16 network. The two confusion matrices supported this observation. The confusion matrix shows that in fact, the network performs as good as implied by the accuracy that is displayed during the training phase. The code has been tested and debugged with multiple variations multiple times.

To deploy it in the real world, a further improvement is suggested, implementing image augmentation to augment the data with transformed versions of the images in the dataset in order to better the performance of the CNN is highly recommended as it will allow the network to classify the boats even if viewed from a different poses.

For the VGG-16, the use of Keras made it possible to have a working model in a few steps without worrying too much about the implementation of the network itself. The availability of pretrained models are probably the fastest way to make a neural network behave somehow correctly on a particular task like the boat classification, but it is not the better way. It’s, in fact, very hard to have a good quality model without enough computational power, and even with a good machine the training time could take a few days and could not achieve a good accuracy because of the size of the training set, this is why all the pretrained models are trained on a large-size dataset like ImageNet. Moreover, since the first layers of the network are responsible for the identification of hirarchical features which are common to all the objects, retraining the network to learn exactly the same things does not take to improvements of the model.