Turkish Lira image classifier

Luca Papparotto n°960853

24/09/2020

"I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study."

1 Introduction

The project I decided to carry out is the Turkish banknote image classifier, as neural networks and machine learning were the topics that interested me most in the course. I immediately implemented a fairly complex neural network in order to understand what was the best result I could get, then going to simplify the architecture and making small adjustments to try to achieve and improve the results. The project led to the implementation of a convolutional neural network capable of classifying Turkish Lira banknotes with an accuracy of $\approx 99,6\%$.

2 The experiment

2.1 Data organization

The Turkish Lira Banknote dataset provided by Kaggle is composed by 6000 color images divided into folders of 1000 images for each banknote category. For each category (10, 100, 20, 200, 5, 50) there are 925 images belonging to the training set, the other 25 for the validation set: each image has a resolution of 1280 x 720 in the .png format.

With this type of structure it is possible to use the image_dataset_from_directory() method provided by the keras library, which will automatically associate to each image a label corresponding to the name of the folder in which the image is contained.

```
def get_training_dataset(height, width, batch_size, resize):
    train_ds = tf.keras.preprocessing.image_dataset_from_directory(
        to_path_train,
        labels="inferred",
        label_mode="categorical",
        image_size = (height, width),
        color_mode="rgb",
        batch_size=batch_size,
        shuffle=True,
        seed=None,
        validation_split=None,
        subset=None,
        follow_links=False,
    )
    if(resize):
      train_ds = train_ds.map
        (lambda x, y: (data_augmentation(x, training=True), y),
                    num_parallel_calls=AUTO)


    train_ds = train_ds.repeat().prefetch(AUTO)
    return train_ds
```

For a better organization of the data it was also necessary to divide the images belonging to the training dataset from those of the validation dataset, made possible by moving the images into their respective folders using the files provided by Kaggle as a guide:

```
!mkdir dataset/txtFiles
!mv dataset/downloaded/train.txt dataset/txtFiles
!mv dataset/downloaded/validation.txt dataset/txtFiles

!mkdir -p dataset/train_set/{\10,100,20,200,5,50}
!mkdir -p dataset/valid_set/{\10,100,20,200,5,50}

import os


train_file = "dataset/txtFiles/train.txt"

from_path_train = "dataset/downloaded/"
to_path_train = "dataset/train_set/"


with open(train_file, "r") as file1:
    for line in file1:
      os.rename(from_path_train + line[:-1] , to_path_train + line[:-1])
```

```
validation_file = "dataset/txtFiles/validation.txt"

from_path_valid = "dataset/downloaded/"
to_path_valid = "dataset/valid_set/"

with open(validation_file, "r") as file1:
    for line in file1:
      os.rename(from_path_valid + line[:-1] , to_path_valid + line[:-1])
```

2.2 Dataset an preprocessing

As previously mentioned, the dataset consists of 6000 images: 5550 belonging to the training dataset, 450 to the validation dataset with a shape of 1280 x 720 x 3.

It can be noted that the number of images for each category provided by the dataset are few but of considerable size, this made it necessary to carry out preliminary operations in order to proceed with the experiment and obtain good results.

Using so few images it would appear that our model will not be able to recognize in an acceptable way (75% accuracy) the banknotes of the validation set (or those provided by the user) despite having an accuracy of about 99.8% on the training set. Fewer images also implies that the model overfits very easily causing incorrect classifications, therefore, it was necessary to use data augmentation techniques in order to bypass all these problems.

We have decided to add a first preprocessing layer in the neural network architecture using the experimental preprocessing layer methods given by the keras library, which allows us to use GPU acceleration ensuring better scalability for larger datasets. The layer inside the model gives us the possibility of not carry out the pre-process separately and independently, especially if you only want to distribute the trained model. On the other hand, if the dataset is composed of a large number of images (sufficient to obtain a good result), this layer will affect the performance as it will do the data augmentation anyway.

```
data_augmentation = tf.keras.Sequential(
  [
    tf.keras.layers.experimental.preprocessing.RandomFlip("horizontal",
                                                input_shape=(img_height,
                                                             img_width,
                                                             3)),
    tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),
    tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

As we can see, this layer applies transformations (zoom, rotations, flips) to the images and creates new ones enlarging the dataset. There is also a second problem related to the dataset: the size of the images.

All the images contained in the dataset have a shape of 1280 x 720 x 3 which will generate very large tensors that will lead us to run out of RAM very quickly: it is necessary to perform dimensionality reduction. The images are not so big that they need sophisticated algorithms to reduce their size, so we have simply resized it. This could be a problem if we want to consider very big images (10M pixels) and in this case a clever algoritm for dimensionality reduction such PCA will perform better. A good resolution we found was 80 * 45.

Another minor preprocess needed is to rescale the image by $\frac{1}{255}$ because our images are RGB and for the learning process a value between 0 and 1 is needed instead one than ranges from 1 to 255.

2.3 The Neural Network

Since we are working on an image dataset, we have chosen to use a convolutional neural network which is perfect for these purposes. The first neural network implemented was the following:

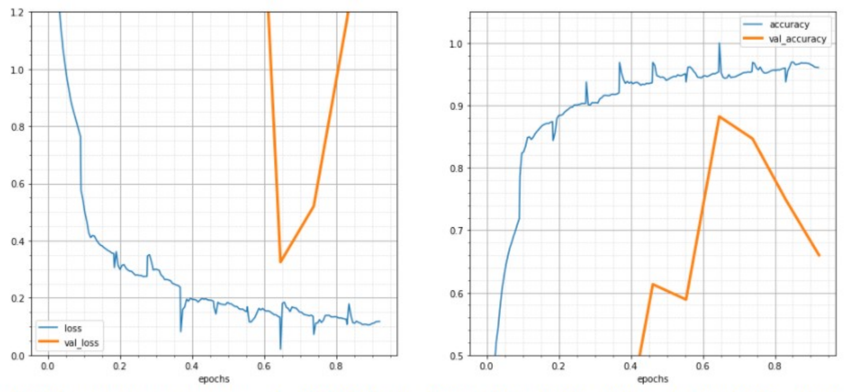
```
model = tf.keras.Sequential([
data_augmentation ,
tf.keras.layers.experimental.preprocessing
  .Rescaling(1./255, input_shape=(img_height, img_width, 3)),

tf.keras.layers.Conv2D(kernel_size=3, filters=16,
                        padding='same', use_bias=False),
tf.keras.layers.BatchNormalization(center=True, scale=False),
tf.keras.layers.Activation('relu'),


tf.keras.layers.Conv2D(kernel_size=3, filters=24,
                   padding='same', use_bias=False),
tf.keras.layers.BatchNormalization(center=True, scale=False),
tf.keras.layers.Activation('relu'),


tf.keras.layers.Conv2D(kernel_size=3, filters=32,
                        padding='same', use_bias=False),
tf.keras.layers.BatchNormalization(center=True, scale=False),
tf.keras.layers.Activation('relu'),


tf.keras.layers.Flatten(),

tf.keras.layers.Dense(200, use_bias=False),
tf.keras.layers.BatchNormalization(center=True, scale=False),
tf.keras.layers.Activation('relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(6, activation='softmax')
])

model.compile(optimizer=tf.keras.optimizers.Adam(lr=0.01),
              loss='categorical_crossentropy',
              metrics=['accuracy'])
```

As previously specified we have first of all the data augmentation layer followed by the rescale layer, then we have 3 convolutional layers with batch normalization and 'relu' as activation function. These convolutional layers always have the same kernel size, but each time the number of filters is increased in order to be able to extract increasingly complex features from the images. Furthermore, the 'Adam' optimization function was used with a static learning rate of 0.01 and the categorical crossentropy as loss function since it is a multiclass classification problem.

With this neural network we start by doing some experiments with the following parameters:

- Batch size = 32
- Epochs = 10
- Image resolution = 64 * 36

And we obtained:



```
173/173 [==============================] - 108s 627ms/step - loss: 0.1166 - accuracy: 0.9612 - val_loss: 2.0133 - val_accuracy: 0.6600
```

Not good result which can be caused by two factors: the static learning rate and too much image compression. So we added the dynamic learning rate and resized the images to 80 * 45:

```
316s 1s/step - loss: 0.0241 - accuracy: 0.9927 - val_loss: 0.0167 - val_accuracy: 0.9956
```

After that we have tried to improve the model by tweaking the architectures so we implemented a different network:

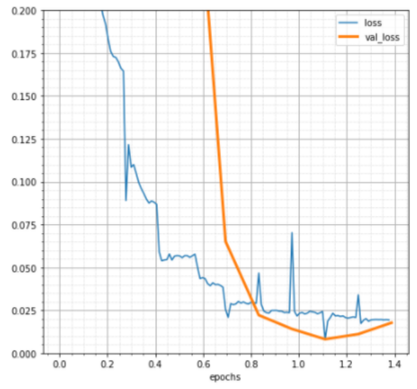
```
model = tf.keras.Sequential([
  data_augmentation,
 tf.keras.layers.experimental.preprocessing
        .Rescaling(1./255, input_shape=(img_height, img_width, 3)),

  tf.keras.layers.Conv2D(kernel_size=3, filters=16,
        padding='same', use_bias=False),
  tf.keras.layers.BatchNormalization(center=True, scale=False),
  tf.keras.layers.Activation('relu'),
  tf.keras.layers.MaxPooling2D(),


  tf.keras.layers.Conv2D(kernel_size=6, filters=24,
        padding='same', use_bias=False),
  tf.keras.layers.BatchNormalization(center=True, scale=False),
  tf.keras.layers.Activation('relu'),
  tf.keras.layers.MaxPooling2D(),

  tf.keras.layers.Flatten(),
  tf.keras.layers.Dense(200, use_bias=False),
  tf.keras.layers.BatchNormalization(center=True, scale=False),
  tf.keras.layers.Activation('relu'),

  tf.keras.layers.Dense(60, use_bias=False),
  tf.keras.layers.BatchNormalization(center=True, scale=False),
  tf.keras.layers.Activation('relu'),

  tf.keras.layers.Dropout(0.3),
  tf.keras.layers.Dense(6, activation='softmax')
])

model1.compile(optimizer=tf.keras.optimizers.Adam(lr=0.01),
               loss='categorical_crossentropy',
               metrics=['accuracy'])
```

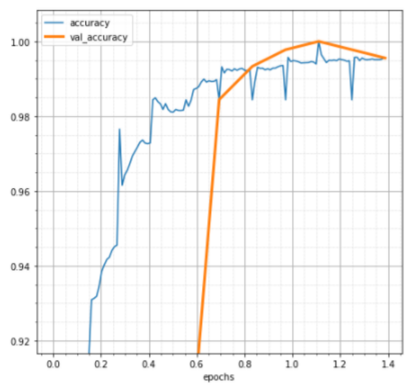
This new model has some differences compared to the previous one, which can be summarized as follows:

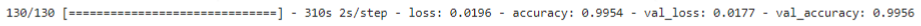
- it only has two covolutional layers which this time have a different kernel size;
- has a second dense layer of size 60;
- both covolutional layers use pooling functions that allow us to compress the result of the previous layer by decreasing the number of parameters to be learned;

with this new network we have decreased the number of total parameters from 23,052,822 to 1,083,522.

This new network improves in all aspects the previous one obtaining these results:





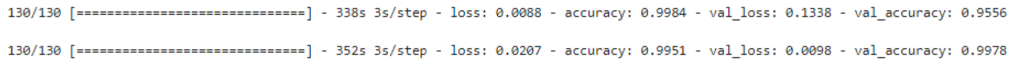


```
130/130 [==============================] - 310s 2s/step - loss: 0.0196 - accuracy: 0.9954 - val_loss: 0.0177 - val_accuracy: 0.9956
```

This model was the best we were able to get.

2.4 Fine tweaking and other experimets

In the TurkishLira.ipynb file we can see all the experiment done trying to study our model for example how it performs with different dropout parameter (0,4) or a new one between the the two dense layer(0.1):

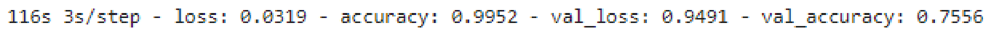


```
130/130 [==============================] - 338s 3s/step - loss: 0.0088 - accuracy: 0.9984 - val_loss: 0.1338 - val_accuracy: 0.9556

130/130 [==============================] - 352s 3s/step - loss: 0.0207 - accuracy: 0.9951 - val_loss: 0.0098 - val_accuracy: 0.9978
```

same or worst result. After that we focused on testing all those assumptions and optimizations made from the beginning to see if they were really useful analyzing their impact on performance. In particular, we focused on the testing of:

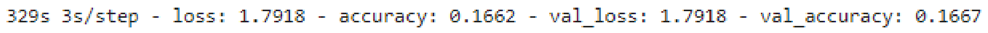
- data augmentation;
- batch normalization;
- dropout

as expected, the impact on performance is really crucial: without data augmentation our model is unable to recognize the images belonging to the validation dataset in an acceptable way;



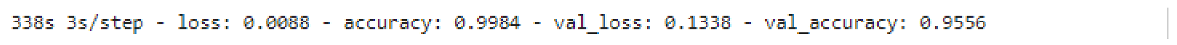
```
116s 3s/step - loss: 0.0319 - accuracy: 0.9952 - val_loss: 0.9491 - val_accuracy: 0.7556
```

without batch normalization, the neural network cannot learn effectively;



```
329s 3s/step - loss: 1.7918 - accuracy: 0.1662 - val_loss: 1.7918 - val_accuracy: 0.1667
```

and without the dropout the performance is considerably reduced due to the heavy overfitting of the model.



```
338s 3s/step - loss: 0.0088 - accuracy: 0.9984 - val_loss: 0.1338 - val_accuracy: 0.9556
```

3 Conclusion and consideration

We managed to get a pretty decent model capable of about 99.6% accuracy by limiting the number of epochs to 10. It was also tried to increase this number to 20 (not present in the code) and it was noted that the precision suffered a flattening of the curve and sometimes an increase of the same due to overfitting. In case you have a larger dataset (more images) this model may not be optimal due to the data augmentation present inside it, but necessary in the case of this work as seen from the experiments. It should also be noted that all the assumptions and optimizations present from the beginning were correct and crucial for the correct functioning of the model, in line with expectations and with the theory studied in class. All these experiments are present in the TurkishLira.ipynb file while a cleaner version, with only the best model, is located inside the Turkish_clean_code.ipnyb file where there is also a method to insert user images.