MACHINE LEARNING

NEURONAL NETWORKS: Traffic Signs Recognition

SEBASTIÁN FERNÁNDEZ GARCÍA Y NAHIMA ORTEGA RODRÍGUEZ

INTRODUCTION

In this report, we are going to summarize the main aspects of the work we have done. It is a neural network that is capable of recognizing nine different types of traffic signs of the caution type, or warning of danger.

Our main motivation is that a large majority of today's cars are able to recognize speed limit signs and then show them to the driver. This has become a feature of even those vehicles that are not necessarily high-end. However, they are not yet capable of detecting traffic signs that warn the driver of some kind of danger, despite their great importance.

We have selected a total of 9 different categories:

1. P-50 sign. Other dangers.
2. Signal P-14b. Dangerous curves to the left.
3. P-20 signal. Pedestrians
4. P-19 signal. Slippery pavement.
5. P-18 signal. Plays.
6. P-3 signal. Traffic lights.
7. P-15a sign. Projection.
8. P-1 signal. Intersection with priority.
9. P-24 sign. Animal passage in freedom

The images have been extracted to a large extent from the GTSRB (German Traffic Sign Recognition Benchmark) dataset, although we have also obtained images from other datasets or have been completely personal elaboration (Own photos, Maps etc.)

HYPERPARAMETERS CONFIGURATION

Next, we are going to make a comparative table with the hyperparameters that we have tested and their respective results, since we will be left with the best of them.

	Layers	Batch Size	Epoch	Optimizer	Validation Accuracy

Version 1	`Conv2D, 32, 3x3, relu`
	`MaxPooling2D 2x2`
	`Conv2D, 64, 3x3, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Conv2D, 128, 3x3, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Dense, 128, relu`
	`Dropout 0.5`
	`Dense, 9, softmax`
		32	200	Adam	0.9874 (train and test data different from the rest)

Version 2	`Conv2D, 32, 3x3, relu`
	`MaxPooling2D 2x2`
	`Conv2D, 64, 3x3, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Conv2D, 128, 3x3, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Dense, 128, relu`
	`Dropout 0.5`
	`Dense, 9, softmax`
		32	200	Adam	0.9667 (new data)

Version 3	`Conv2D, 32, 3x3, relu`
	`MaxPooling2D 2x2`
	`Conv2D, 64, 3x3, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Conv2D, 128, 3x3, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Dense, 128, relu`
	`Dropout 0.5`
	`Dense, 9, softmax`
		32	200	SGD	0.8667

Version 3.2	"	"	"	Adagrad	0.8861
Version 3.3	"	"	"	Nadam	0.9778
Version 3.4	"	"	"	RMSprop	0.9694

Version 4	`Conv2D, 32, 5x5, relu`
	`MaxPooling2D 2x2`
	`Conv2D, 64, 5x5, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Conv2D, 128, 5x5, relu`
	`MaxPooling2D 2x2`
	`Dropout 0.25`
	`Dense, 128, relu`
	`Dropout 0.5`
	`Dense, 9, softmax`
		32	200	Nadam	0.9694

Version 4.3	`Conv2D, 32, 5x5, relu` `MaxPooling2D 2x2`	32	200	Adam	0.9806

	Conv2D, 64, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Conv2D, 128, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 128, relu Dropout 0.5 Dense, 9, softmax				
Version 5	Conv2D, 32, 5x5, relu MaxPooling2D 2x2 Conv2D, 64, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 128, relu Dropout 0.5 Dense, 9, softmax	32	200	Adam	0.9222
Version 5.2	Conv2D, 32, 3x3, relu MaxPooling2D 2x2 Conv2D, 64, 3x3, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 128, relu Dropout 0.5 Dense, 9, softmax	32	200	Adam	0.9389
Version 6	Conv2D, 16, 3x3, relu MaxPooling2D 2x2 Conv2D, 32, 3x3, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 64, relu Dropout 0.5 Dense, 9, softmax	32	200	Adam	0.9278
Version 7	Conv2D, 16, 3x3, relu MaxPooling2D 2x2 Conv2D, 32, 3x3, relu MaxPooling2D 2x2 Dropout 0.25 Conv2D, 64, 3x3, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 64, relu Dropout 0.5 Dense, 9, softmax	32	200	Adam	0.9806
Version 7.2	Conv2D, 16, 5x5, relu MaxPooling2D 2x2 Conv2D, 32, 5x5, relu MaxPooling2D 2x2 Dropout 0.25 Conv2D, 64, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 64, relu Dropout 0.5 Dense, 9, softmax	32	200	Adam	0.9750
Version 8	Conv2D, 16, 5x5, relu MaxPooling2D 2x2	32	200	Adam	0.9833

	Conv2D, 32, 5x5, relu MaxPooling2D 2x2 Dropout 0.25 Conv2D, 64, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Conv2D, 128, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 128, relu Dropout 0.5 Dense, 9, softmax				
Version 8.2	Conv2D, 16, 5x5, relu MaxPooling2D 2x2 Conv2D, 32, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Conv2D, 64, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Conv2D, 128, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 128, relu Dropout 0.5 Dense, 9, softmax	32	200	Nadam	0.9750
Version 10	Conv2D, 32, 5x5, relu MaxPooling2D 2x2 Conv2D, 64, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Conv2D, 128, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 128, relu Dropout 0.5 Dense, 9, softmax	64	100	Adam	0.9778
Version 10.2	"	100	"	"	0.9750
Version 10.3	"	32	"	"	0.9806
Version 11	Conv2D, 32, 5x5, relu MaxPooling2D 2x2 Conv2D, 64, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Conv2D, 128, 5x5, relu MaxPooling2D 2x2 Dropout 0.5 Dense, 9, softmax	32	100	Adam	0.9667

Our best values are obtained for version 7, 8 and 10.3. However, in version 8 we have used one more convolutional layer, that is, we have made the network a little more complex, but we have not obtained much better results. Therefore, we will stick with version 7.

Some of the conclusions that we have obtained after testing with the different parameters is that, for our case, the Adam and Nadam optimizers have worked much better for us. Also, to avoid overfitting we have included a dropout rate of 0.5. This has led to a slight increase in the success rate as well.
We have had better results for batch sizes of 32 images and worse results for larger batch sizes. In addition, by reducing the number of layers we obtain a success rate of around 90%, while adding a third layer, this value is above 95%. However, adding a fourth layer does not increase this value considerably. Therefore, we believe that the number of layers that best suits our objective is three. Regarding the size of the kernel of each layer, the size of 5x5 had given us, at first, a slight improvement. Propagating it to all layers, the difference has been a bit more significant. Despite this, there are certain configurations that we have tested with the 3x3 size and it has turned out to be more effective than 5x5, as is the case with the seventh version.

As can be seen, we have obtained very high success values in general. We believe this has been due to the fact that images are always presented in similar conditions. Hence, as of version 2, we began to use a validation data set prepared by us manually, in which we included images a little more distorted, with obstacles, from different perspectives, etc., to ensure that this set it was not very simple.

CODE

Next we are going to explain the most relevant aspects of the code, which has been developed in Google Colab.

In the first part, we must connect to Google Drive to be able to load our training and validation datasets, which had been previously uploaded to this platform. Initially, all categories generally have 200 images. Of these, 40 have been dedicated to the validation set and the remaining 160, for training.

Mounting our Google Drive unit to be able to access our files.

```
[1]  from google.colab import drive
     drive.mount('/content/drive')
```

We must also import Tensorflow, since we are going to work with this library to program our neural network. In this case, we check in this chunk that we have a GPU in our Colab notebook, which will be very useful for speeding up the calculations when training our model.

```
[2]  import tensorflow as tf
     tf.test.gpu_device_name()

     '/device:GPU:0'
```

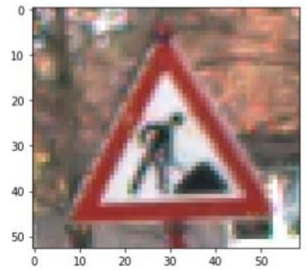
We visualize one of the images of our Training Set. For this we use a series of libraries that we have imported (matplotlib, numpy and PIL).

```
[ ]  !ls "/content/drive/My Drive/Colab Notebooks/datasets/"

     from matplotlib.pyplot import imshow
     import numpy as np
     from PIL import Image

     %matplotlib inline
     pil_im = Image.open('/content/drive/My Drive/Colab Notebooks/datasets/Traffic-Signs/5/00025_00040_00018.png', 'r')
     imshow(np.asarray(pil_im))
```

```
Traffic-Signs  Traffic-Signs-2  Traffic-Signs-Validation
<matplotlib.image.AxesImage at 0x7f7d4003eb90>
```



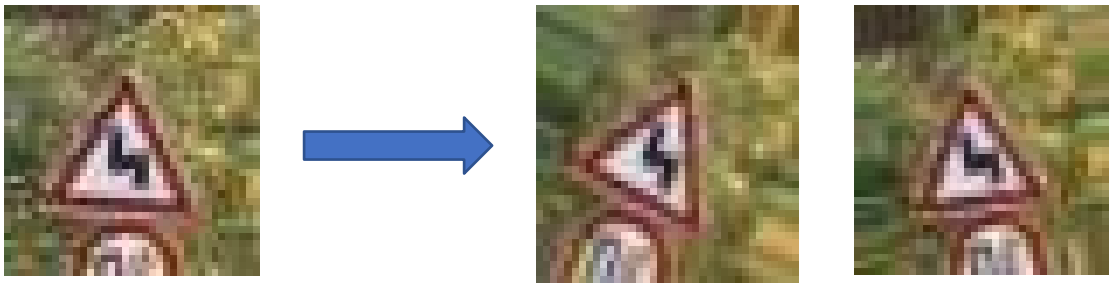
We have also performed data augmentation on the training dataset.
The main "problem" we have encountered using this traffic sign dataset is that, in most cases, we encounter the signs from the right side, from similar angles and distances. Furthermore, these signals (compared to other datasets

related to objects that may differ from one image to another, such as animals, people, etc.), have the particularity that they all have the same conditions (same size, color, representation), since they are regulated and standardized by the authorities competent authorities of each country. Therefore, in order to have the maximum possible variety of images, we created two different images from each image in the training set using data augmentation.

First, we create an ImageDataGenerator, which will allow us to create new images according to the parameters that we pass to it. We indicate that we do not want it to turn our images horizontally or vertically, since this would not make sense in any real case. This is because the signals are not usually symmetrical nor do we usually find them down. Our main interest is to rotate and zoom them. Therefore, we set a rotation range of 25º and a zoom range of 20%. Also, we give a cutoff range of 0.20.

Once the parameters have been established, we read all the images in our Training Set, and we generate the new images with the flow () function. We store these images in our Google Drive for monitoring purposes, to later check if the images that are being generated meet the characteristics we are looking for.

Indeed, noticeable changes are perceived that help us to create a slightly more varied dataset, as is the following.



Now, we will indicate what our training and validation sets will be. Before, we establish the size that our images will have. Since we do not want a large resolution, we choose a size of 150x150. In addition, we indicate a batch size of 32. This means that 32 samples will be chosen to calculate the error gradient before the model weights are updated. That is, the training images will be passed to the network in batches of 32 images.

We bring our training set from our Google Drive directory, and we indicate the parameters of image size and batch size. In the label_mode we indicate categorical because the labels are going to be represented as a categorical vector, since we are going to have more than two classes. Next, we carry out the same procedure with the validation set.

Finally, we indicate the maximum number of elements to be stored in the buffer during the prefetch procedure.

```
[18] # DATA SOURCE ------------------------------------------------

     image_size = (150, 150)
     batch_size = 32

     train_ds = tf.keras.preprocessing.image_dataset_from_directory(
         "/content/drive/My Drive/Colab Notebooks/datasets/Traffic-Signs",
         image_size=image_size,
         batch_size=batch_size,
         label_mode='categorical'
     )
     val_ds = tf.keras.preprocessing.image_dataset_from_directory(
         "/content/drive/My Drive/Colab Notebooks/datasets/Traffic-Signs-Validation",
         image_size=image_size,
         batch_size=batch_size,
         label_mode='categorical'
     )

     train_ds = train_ds.prefetch(buffer_size=32)
     val_ds = val_ds.prefetch(buffer_size=32)
```

Now we enter the part of creating the model of our neural network.

First, with Sequential (), we indicate that we are going to start building a sequential model, that is, that the layers are going to be added in sequence. Then, with Rescaling (), we rescale the input in the range [0, 255] to the range [-1, 1] and also include the size we want the images to have on our network.

Once this is done, we start adding capable to the network.

The first one we add has 32 filters, a kernel size of 5x5, and has ReLU-activation and max-pooling. But what does all this mean?

On the one hand, filters are used to represent image characteristics.

On the other hand, the kernel is a filter that we are going to pass over the input image to find the convolution value. This kernel will be used to detect certain characteristics of the images.

Finally, we apply the ReLU activation function and max-pooling to eliminate data without greatly reducing the information.

The Dropout serves to prevent overtraining, and it does so by setting input units to 0 in the frequency that we indicate.

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Dropout, Dense, Rescaling, Flatten
from tensorflow.keras.callbacks import EarlyStopping

model = keras.Sequential()
model.add(Rescaling(scale=(1./127.5),
                    offset=-1,
                    input_shape=(150, 150, 3)))
model.add(Conv2D(16, kernel_size=(3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Conv2D(32, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))
model.add(Conv2D(64, (3, 3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.5))

model.add(Flatten())
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(9, activation='softmax'))

model.compile(loss=tf.keras.losses.categorical_crossentropy,
              optimizer=tf.keras.optimizers.Adam(1e-3),
              metrics=['accuracy'])
```

We also see how we use categorical_crossentropy as the loss function. This is a measure of the distance between probability distributions. It is used in classification tasks in which we have several classes, and in which an example can only belong to one of the possible categories, our model having to decide which of them. In other words, it is ideal for our case, since we classify an image in the nine possible categories that we have. This function can consider that an example belongs to a category with probability 1 and to other categories with probability 0, hence it is used for classification.

Given an image, the prediction is a probability vector that represents the probabilities that are predicted for each of the classes. These predictions are normally obtained when the softmax function is used as the activation function for the last layer. In addition, we have another one-hot vector, which indicates with a 1 in a position the category it represents. Therefore, for a certain category, we want the probability to be 1, while for other classes we want it to be 0.

Being a loss function, the lower the value of the loss, the better. In training, the model tries to achieve as little loss as possible.

For the cross-entropy function, we calculate the loss for each of the classes independently and then add them together. The loss is computed by multiplying the probability of the class in the one-hot vector multiplied by the logarithm of the class in the prediction vector. The sign of this operation is inverted, so that it tends to infinity or to 0. For the Categorical Cross Entropy function, we forget the rest of the classes, and we only calculate the loss for the class indicated with a 1 in the vector one hot (hot class). That is, the value of the function will be the logarithm of the probability.

As we can see, this function does not take into account if we have a warning sign for bumping and interprets it as a sign of dangerous curves with a 70% probability, but takes into account that there would be a remaining 30% to identify the correct class , in this case the loss value being very high. That is, you don't take into account where the rest of the probability goes, you only look at how well you identify the result class.

Therefore, the loss is 0 if the prediction is 1. On the other hand, the loss has infinity if the prediction is 0, which is the opposite of the result of our one-hot vector. Therefore, the further we are from the value of the one-hot vector, the faster the error function will grow.

Once the Categorical Cross Entropy function is explained, we see that we use an optimizer: "Adam". Throughout the versions, we have tested with other optimizers such as Adagrad, which adapts the learning rate to the parameters by carrying out small updates for the most common features and much larger updates for rarer features. On the other hand, Adadelta tries to reduce Adagrad's aggressive learning speed by restricting the cumulative gradient window. RMSProp also attempts to solve for Adagrad decreasing learning rates using a squared gradient average. Lastly, Adam works as a combination. of Adagrad and RMSprop.

Next, we begin to train our neural network. To do this, we define the number of epochs and use EarlyStopping to prevent overtraining. The patience value indicates the number of epochs to be expected in the case of deterioration and restore_best_weights allows us to return to the highest value obtained.

```
# TRAINING -----------------------------------------------

epochs = 100

es = EarlyStopping(monitor='val_accuracy', mode='max', verbose=1, patience=10, restore_best_weights=True)

h = model.fit(
        train_ds,
        epochs=epochs,
        validation_data=val_ds,
        callbacks = [es]
)
```

Finally, we examine the hit values that our neural network gives us and create the confusion matrix. Additionally, we have tested with images that the neural network has never seen to verify that it correctly recognized which category it belongs to. As we can see, the categories are numbered from 1 to 9 for simplicity.

The results that we have obtained are shown graphically by executing this code.

```
[ ]  import matplotlib.pyplot as plt

     plt.plot(h.history['accuracy'])
     plt.plot(h.history['val_accuracy'])
     plt.plot(h.history['loss'])
     plt.title('Model accuracy')
     plt.ylabel('accuracy')
     plt.xlabel('epoch')
     plt.legend(['training', 'validation','loss'], loc='upper right')
     plt.show()
```

To get the confusion matrix:

```
[ ] import numpy as np
    from sklearn.metrics import classification_report, confusion_matrix
    import seaborn as sns

    results = np.concatenate([(y, model.predict(x=x)) for x, y in val_ds], axis=1)

    predictions = np.argmax(results[0], axis=1)
    labels = np.argmax(results[1], axis=1)

    cf_matrix = confusion_matrix(labels, predictions)

    sns.heatmap(cf_matrix, annot=True, fmt="d", cmap="Blues")

    print(classification_report(labels, predictions, digits = 4))
```

Finally, we tested with our own images (taken with phones around the streets).

```
[ ] img = keras.preprocessing.image.load_img(
        '/content/drive/My Drive/Colab Notebooks/imagenes/imagen5.png', target_size=image_size
    )
    img_array = keras.preprocessing.image.img_to_array(img)
    img_array = tf.expand_dims(img_array, 0)  # Create batch axis

    predictions = model.predict(img_array)
    print(np.argmax(predictions[0]))
```


Authors

- Sebastián Fernández García - @sebastianfernandezgarcia
- Nahima Ortega Rodríguez - @nahimaort

Base code provided by Cayetano Guerra Artal