Convolutional Neural Networks and Image Classification

Sagaf Youssouf

Université Paris-Est Créteil (UPEC)
International Master of Biometrics and Intelligent Vision
https://www.international-master-biometrics-intelligent-vision.org/

1 Introduction

In computer vision, classification is a fundamental task that aims to assign a specific label for a given input. When it comes to image classification, convolutional neural networks are the most widely used. CNNs are a type of neural network that trying to build higher-level features from a set of pixels of an image. To work well, CNN should be trained using lots of data. During the training of the model, we can face a lot of problems including overfitting when we don't have large volumes of data. In that case, there are many solutions to solve the problem and one of them is data augmentation. In this report, we will briefly look at how to load a dataset efficiently and identify overfitting problems. We expose a critical analysis of a simple flower classification exercise illustrated by the official image classification tutorial with Keras.

2 Loading images off disk

The objective of the tutorial is to classify five types of flowers (daisy, dandelion, roses, sunflowers, and tulips). Google colab service based on the Jupypter Notebook has been used for this. Loading images out of the directory is done by calling `image_dataset_from_directory` utility. This will create a `tf.data.Dataset` object. The dataset is split, 80% of the images are used for training, and 20% for validation.

Fig. 1: Images examples of flowers to classify

Table 1

Type of flower	Daisy	Dandelion	Roses	Sunflowers	Tulips
Number of images	633	898	641	699	799

For each type of flowers, the number of occurrences in the dataset is given in table 1. We can see that the dataset is unbanlanced. As RGB channel values are in range `[0, 255]`, we should standardize the data to in `[0, 1]` range by using a `Rescaling Layer` as following :

```
normalizedLayer=layers.experimental.preprocessing.Rescaling(1./255)
```

3 Model that overfits

In the tutorial, a model that overfits is created. It consists of three convolution layers with a max pool layer in each of them. This is followed by a fully connected layer with 128 units activated with `relu` function. Accuracy and loss for training and validation set after 20 epochs with a batch size of 64 is given in plots 2. The total number of parameters for the model is `3 989 285`.
The training accuracy is increasing over the time whereas validation accuracy turns around 60%. As we can see the training loss continues to decrease and almost reaches zero at epoch 20. This is normal because the model is trained to fit trained data. This phenomenon is called overfitting. This often happens when you train a model with a lot of parameters on a dataset with few examples.

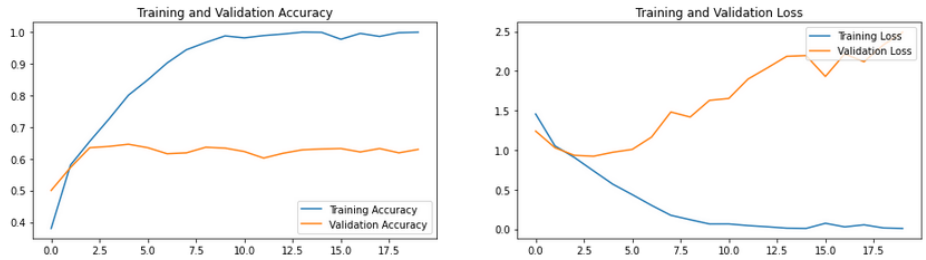

Fig. 2: Overfitting model accuracy and loss

4 Handling overfitting

There are many techniques that can be applied to mitigate overfitting. Among these techniques, there are *Data Augmentation* and use of *Dropout layers*.

4.1 Data augmentation

Data augmentation is a technique used to increase the diversity of the training set by performing random transformation such as image translation and rotation. This yields to believable-looking image and will help to expose the model to more aspects. In this way, the model generalizes better. Data Augmentation is implemented as in the following code using `Keras Preprocessing Layers`. This will be included inside the model as others layers.

```
data_augmentation = keras.Sequential([
    layers.experimental.preprocessing.RandomFlip("horizontal",
                                                 input_shape=(h, w, 3)),
    layers.experimental.preprocessing.RandomTranslation(0.2, 0.2),
    layers.experimental.preprocessing.RandomRotation(0.3),
    layers.experimental.preprocessing.RandomContrast(0.35),
    layers.experimental.preprocessing.RandomRotation(0.1),
    layers.experimental.preprocessing.RandomZoom(0.1),
  ]
)
```

Examples of resulting images examples when data augmentation is applied are illustread in the following figure.

Fig. 3

4.2 Use *Dropout layers*

Dropout is a form of regularization. It will randomly remove a certain number of outputs units from the layer at each update of the training phase. Dropout function takes as argument a fractional: `layers.Dropout(0.2)`

4.3 Train on the full train data and evaluation

Data augmentation is performed and two *dropout* layers have been added as illustrated in the following code :

```
model = Sequential([
  data_augmentation,
  layers.experimental.preprocessing.Rescaling(1./255),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.5),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(128, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(256, activation='relu'),
  layers.Dense(num_classes)
])
```

After 40 epochs, accuracy and loss curve for training and validation for the new model are given in 4. As we can see, applying *Data Augmentation* and using *dropout* layers yield to significant performance improvements. The validation accuracy approachs the asymptote. For loss and accuracy, the two curves are roughly overlapped (closer aligned).
The resulting final accuracy on the validation data is 71.66%. And the total number of parameterq is `2 080 677`. We notice that reducing number of parameters helps a lot. A simple benchmark is given in table 2. The model using data augmentation and dropout is the one that has high accuracy on the validation data.

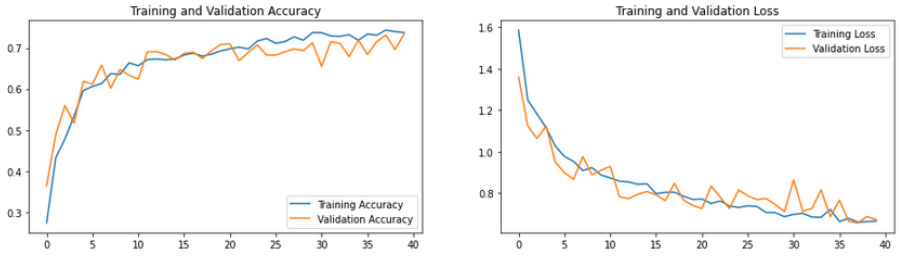

Fig. 4: Loss and accuracy of the model with data augmentation and dropout

Table 2

Models	Validation accuracy
Model with data augmentation and dropout	72.66 %
Model with data augmentation and no dropout	71.93 %
Model with no data augmentation and no dropout	68.53 %

5 Conclusion

As shown above, Data Augmentation and dropout help to reduce overfitting. We manage to increase the accuracy on validation data substantially without provoke underfitting. Some other options may help mitigate overfitting. We can try to reduce the network's capacity by removing the number of elements in the hidden layers. We can also apply regularization by adding a cost to the loss function for large weights.

References

1. Image classification, `https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/images/classification.ipynb`.