SAPIENZA
UNIVERSITÀ DI ROMA

Artificial Intelligence and Robotics

Machine Learning

2019 Fall

Nijat Mursali | 1919669

HOMEWORK 2

Rome, Italy

Table of Contents

List of Figures

Abstract

This report describes solving image classification problem in two modes which are solving it from scratch by defining CNN and training it and secondly applying transfer learning and fine tuning from a pre-trained model.

Introduction

Before introducing our algorithm, we have to first understand what exactly classification means. Classification is of the ways of predicting the class of given data points in a sense of labelling classes into categories. In classification we must be sure about the task of approximating mapping function which takes input variables and converts into discrete output variables. Examples for classification problems can be speech recognition, document classification, spam recognition and so on. In speech recognition it takes the sound waves as input and through neural network it gives the output of plain text at the end. In spam recognition, as it is binary classification, we care only if the document is spam or not. As we have mentioned we will be using dataset in our algorithms, so this classification problem is one category of supervised learning which means we take labelled dataset with the input data. For solving these kind of problems, there are several classification algorithms already invented such as Support Vector Machines, Decision Trees, Neural Networks as so on. In our previous homework report we have implemented SVM and Decision Trees, but in this report, we will be implementing CNN which stands for Convolutional Neural Network, transfer learning and fine tuning in order to solve our problem.

Design

Convolutional Neural Networks

Convolutional networks are one type of neural network in order to process the data. To be more specific, convolution is the operation on two functions of a real valued argument.[1]. As an example, we can show the tracking the position of the spaceship where we assign variables of a single output *x(t)*, the location of the ship with. Thus, we can find the new *s* by writing:

$$s(t) = \int x(a)w(t-a)da$$

We call this operation convolution which we need to have *w* to be valid probability density function. In our formula, the first argument is normally called **input** and the second argument is called **kernel**. In the end, we call the multiplication of input with kernel the **feature map** as output.

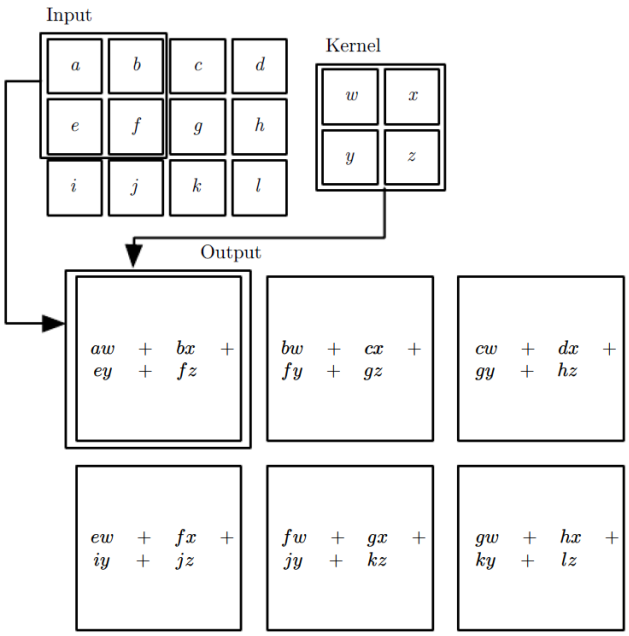

Fig 1.1 Example of 2-D convolution without kernel-flipping

Convolution leverages three important ideas that can help improve a machine learning system: **sparse interactions**, **parameter sharing** and **equivariant representations.**[2].

To make it short, sparse interactions can be achieved by making the kernel smaller than the input which means we can find or detect the edges with kernels that occupy only hundreds of pixels. By reducing the number of pixels, we just stick on the important ones which improves statistical efficiency. Parameter sharing happens when we use the same parameter for more than one model.

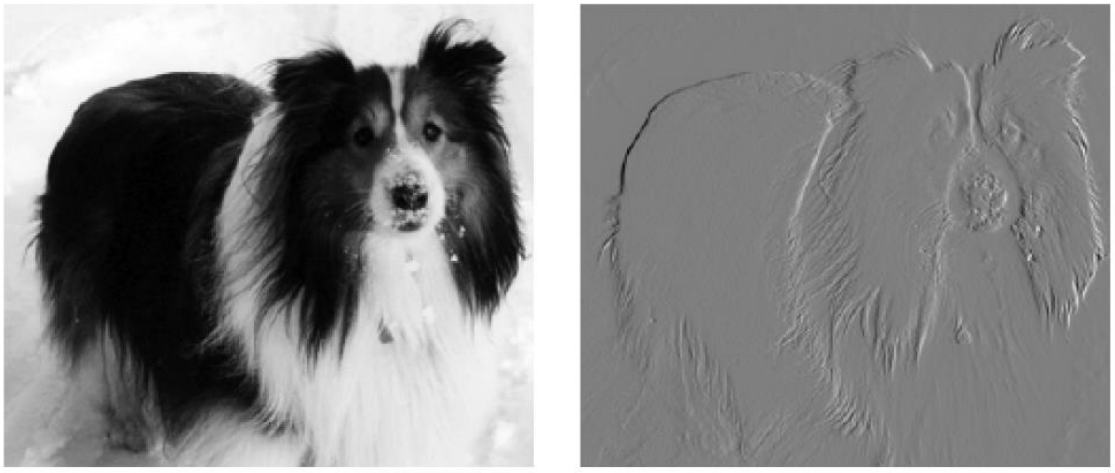

Fig 1.2 Efficiency of edge detection

As shown in *Fig 1.2*, the image in the right side was formed by taking each pixel in the original image and subtracting the value of its neighbouring pixel on the left.

Transfer Learning

Transfer learning is one of the machine learning methods where the model is being reused as the starting point for the model on a second task. It is often used in deep learning where pre-trained models are used as the starting point on computer vision and natural language processing tasks given the vast compute and time resources required to develop neural network

models on these problems and from the huge jumps in skill that they provide on related problems. [3]

The fundamental question here is how we can use transfer learning? There are usually two common approaches for this one. First approach is to develop model approach and secondly, we have pre-trained model approach. In model approach, we have to select the base source task, develop the source model, reuse and tune the model. In pre-trained model approach, we don't need the source model because it is chosen from available models.

Fine Tuning

Fine tuning is one of the machine learning methods where we reuse the network model that has already been trained beforehand and we make it perform second similar task. In our case, using fine tuning will be giving benefits to us in a sense of feature extraction which happens in front layers of the network. To be more specific, in our dataset we have more than one thousand images in four classes: haze, rainy, snowy and sunny. We have to set the trainable value to false for all other lower layers as they have already been trained.

Implementation

As we have discussed before, we have to implement to approaches of image classification in our report: convolutional neural networks from scratch and training data using transfer learning and fine tuning. In the following paragraphs, we will be starting to talk about the implementation of CNN first and then going to deeper in training.

Implementation of CNN from scratch

To solve these problems, we have a combination of four classes of images which are haze, cloudy, snowy and rainy. In each of the classes, we have more than 100 images that are ready to be evaluated. Firstly, we have implemented feature extraction and exported it to *txt* file and

added it to implement CNN. After that, we have created bunch of list variables in order to add for train/test images and labels and we have added the extracted features to corresponding variables. Then, we have added those lists into NumPy arrays in order to go to further steps. In next steps, using the *Sequential* Keras library we have implemented the CNN and using the *model.fit* we have found the accuracy which we will be mentioning in further paragraphs.

Implementation transfer learning and fine tuning

In order to solve the image classification problem with transfer learning we needed to add the dataset and apply transfer learning and fine tuning. In order to do this, we have used *'MWI-Dataset-1.1_400'* dataset which holds two thousand images in overall in four classes. To make our life easier, we first tried to create the *python* file in order to resize the images by decreasing the size of them. As we have discussed in our introduction part, we needed to do this just to get rid of huge amount of computation steps and decrease the time.

```
Found 320 images belonging to 4 classes.
Found 80 images belonging to 4 classes.
Image input (118, 224, 3)
Classes: ['HAZE', 'RAINY', 'SNOWY', 'SUNNY']
Loaded 320 training samples from 4 classes.
Loaded 80 test samples from 4 classes.
```

Fig 2.1 Loading Data

Firstly, we have added important libraries such as NumPy, Tensorflow and Keras into our code and after that we set the directories for train and test data. We have used *ImageDataGenerator* in order to generate batches of tensor image data with real-time data augmentation where the data will be looped over in batches. Then, we have trained both train and test sets using *flow_from_directory*. In next steps, we have created several methods in order to apply transfer learning. Firstly, we needed to define input tensor, load a pretrained model and extract the features using Adam optimizer. We will be talking about the results such as accuracy in following paragraphs.

Evaluation

Evaluation of CNN from scratch

In order to evaluate the CNN, we have chosen 10 epochs which are the number of iterations to train the data. To evaluate the CNN, we have used *Sequential* model and Conv2D, MaxPooling2D layers from Keras. When we look at the documentation of *Sequential* model, we can see that all layers we mentioned above are in the code. For the input shape we have chosen 150x150x3 to decrease the computation time for our train data. At the end we have chosen the model and fit it using 10 epochs.

Evaluation of Transfer Learning

In order to evaluate the approaches for image classification, we have used *sklearn* library with *classification_report* and *confusion_matrix.* In order to do that, we have chosen predictions from transferNet and using the test generator we have chosen tests for confusion matrix. We have run 20 epochs to train the model. Epochs are integers which determines the number of integers to train the model. Simply saying ,it is an iteration over the entire data provided.

```
5/5 [==========================>] - ETA: 22:56 - loss: 1.5544 - accuracy: 0.9475
```

Fig 2.2 Last iteration for training

Results

As we have implemented our algorithm, the most important part is to check the results and see how accurate our algorithm is. As described enough above, we have implemented both CNN and transfer learning to check the accuracy of our algorithm within our dataset. With the help of *sklearn* library we have used *classification_report* and *confusion_matrix* to check the

accuracy of the algorithms. Firstly, as we have seen from evaluation part, the accuracy of the CNN is 0.90. Secondly, we also calculated the accuracy for transfer learning and got the accuracy of 0.94 in the twentieth epoch.

Conclusion

This paper has been created and made exhaustive research on implementing our algorithm to solve the image classification problem by applying CNN, transfer learning. Till the end of this project, we have successfully made a lot of research on these topics and tried to explore and understand how all of them works. In this report, we have presented how exactly those classification algorithms work and how we exactly implemented them. The accuracy of our algorithm is higher than 90% for both CNN and transfer learning. However, for the future work we will be trying to find ways to increase the accuracy for our algorithm.

References

1. Ian Goodfellow, Yoshua Bengio, Aaron Courville – Deep Learning
    - https://www.amazon.it/Deep-Learning-Ian-Goodfellow/dp/0262035618
2. Transfer Learning
    - https://machinelearningmastery.com/transfer-learning-for-deep-learning/
    - https://keras.io/models/sequential/