Classification of Image Data

Younggue Kim
McGill ID 260831176

Negin Moslemi
McGill ID 260939867

Abstract

The goal of this work is to analyze different machine learning models, implement and test them on an image database. This paper serves as a report for a project of COMP 551 (Applied Machine Learning), 2020 taught by Professor Reihaneh Rabbany and Professor Mohsen Ravanbakhsh at McGill University [9].

1. Introduction

In general, image classification is the task of assigning an input image one label from a fixed set of categories. The goal of image data recognition is recognizing different categories of objects present in the image. Humans perform well in such tasks as we learn object classes since our birth by analyzing the world around us and learn our native language for describing the world and objects in it. Yet, categorizing a significantly large set of textual data can be exhaustive and time-consuming for a human being. This immediately suggests investigating the possibility of automation of the text categorization process. However, computers lack the ability to learn by default and thus have to be trained to do so.

The problem of image data classification that is proposed to be solved in this work can be defined as developing several machine learning algorithms for estimating whether each analyzed image belongs to some particular category. There are two main challenges in answering this question. The first one embeds enabling the algorithm to ignore irrelevant factors of an image (*e.g.* background, noise, obstacles, etc.). The second challenge is making the algorithm ignore insufficient differences between objects (*e.g.* it must be able to tell that two similar cars are actually cars, ignoring the fact that there are no two completely identical cars in the world).

The work is organized as follows. Section 2 covers the theoretical background of the models used in this work. Section 3 describes the methods used for data pre-processing and the process of learning for each model. Section 4 concludes this work with comparison of the models used in this work.

2. Models structure

The two models that are used in this work are the Multilayer Perceptron (MLP) and the Convolutional Neural Network (CNN). The MLP in this work was developed from scratch while the CNN is adapted from the Keras library for Python [6]. Moreover, the MLP training algorithm was enhanced such that the model can produce more accurate results. On the other hand, after comparing Pytorch [8] and Keras accuracy, it was decided that the CNN should be developed according to the Kaggle tutorial by using Keras [6].

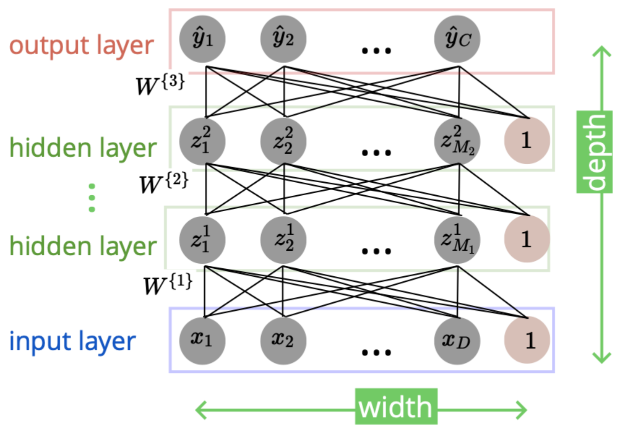

Figure 1: A general scheme of an MLP network. Here x is a D-dimensional input vector of features that forms the input layer (0-layer), $W^{\{l\}}$ is a weight matrix between layers $(l-1)$ and l, $z^{\{l\}} = h(W^{\{l\}} z^{\{l-1\}})$ is the l-th hidden layer where h is an activation function of that layer, $\hat{y}$ is the predicted C-dimensional target vector with C being the number of classes that forms the final layer, adapted from [9]

Most of the model descriptions are adapted from the lecture notes of the Applied Machine Learning course at McGill University and are described in the following subsections [9].

2.1. The multilayer perceptron

The multilayer perceptron (or the feedforward network) is an extension to the classical perceptron model. The general framework of the MLP is illustrated in Figure 1. In this work, the following notation is used: x is a D-dimensional input vector of features that forms the input layer (0-layer), $W^{\{l\}}$ is a weight matrix between layers $(l-1)$ and l, $z^{\{l\}} = h(W^{\{l\}} z^{\{l-1\}})$ is the l-th hidden layer where h is an activation function of that layer, $\hat{y}$ is the predicted C-dimensional target vector with C being the number of classes. There are $M^{\{l\}}$ hidden units in the l-th layer. Note that dimensions of hidden layers should coincide, i.e. $W^{\{l\}} \in \mathbb{R}^{M_l \times M_{l-1}}$. Ultimately, from it follows that $W^{\{1\}} \in \mathbb{R}^{M_1 \times D}$ and $W^{\{L\}} \in \mathbb{R}^{C \times M_{L-1}}$ where L is the total number of layers.

In this work, the feedforward network of the MLP is assumed to be fully connected (dense), i.e. every element of the l-th layer is connected to every element of the $(l+1)$-th layer.

2.2. The convolutional neural network

CNN is a neural network with convolutional layers so as we see it's actually a special type of MLP [9]. In fact, CNN is a simple NN which use convolution in place of general matrix multiplication in at least one of their layers [2]

CNN is designed to automatically and adaptively learn spatial hierarchies of features through backpropagation by using multiple building blocks, such as convolution layers, pooling layers, and fully connected layers (Figure 2). The first two, convolution and pooling layers, perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output, such as classification. A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of linear operation [10].

3. Implementation

This work is carried out using Google Colaboratory so that every team member could contribute to the work in a concurrent and agile way [3]. Thus, the Google cloud servers were used for computational purposes.

The Python implementation for the MLP and the CNN come as an attachment to this work in the ZIP file `code.zip` and are named `COMP_551_MLP_Image_Classification .ipynb` and `MiniProject3_2_Keras.ipynb`, [6] respectively. Moreover, there is a file named `comp_551_mlp_image_classification.py` which serves as an alternative to the Jupyter Notebook (but Jupyter Notebook has higher interactivity thus recommended for use).

3.1. Data description

The CIFAR-10 dataset was adapted for experiments [7]. The dataset consits of 50,000 train images and 10,000 test images with 32×32 resolution. Every image is colored and belongs to one of the ten categories: *plane*, *car*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*. Figure 3 shows 5 first images from the train dataset.

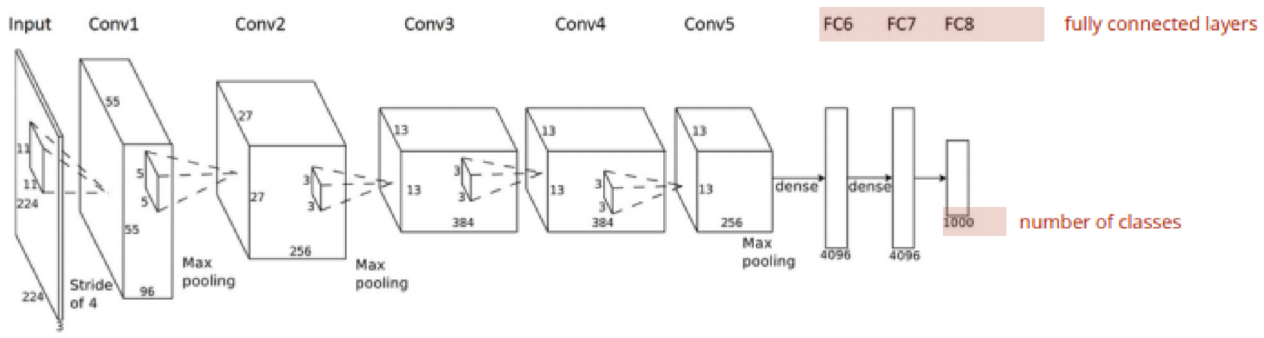

Figure 2: An example of CNN with five convolutional layers, three pooling layers, and three fully connected layers [9]

Figure 3: First 5 images from the CIFAR-10 train dataset with their categories organized from left to right as follows: *frog*, *truck*, *truck*, *deer*, *car*

3.2. Data pre-processing

First, note that every image in the CIFAR-10 dataset has $32 \times 32 \times 3$ or 3072 features in total, as it is essentially a patch of 32×32 pixels with 3 channels corresponding to fundamental colors: red, green, blue (RGB). Every feature x_i takes some value between 0 and 255.

It was decided to normalize the data by converting each feature so that it would belong to an interval between 0 and 1, i.e. dividing every feature by 255. Moreover, for MLP, every image was flattened, so instead of a $32\times32\times3$ matrix a 3072-dimensional vector is used. On the other hand, for CNN, by implementing one-hot encoding, we transformed y to be an array of 10 by the help of keras.utils.tocategorical.

These pre-processing techniques are applied to both train and test datasets.

3.3. Learning

It was decided to apply the k-fold cross-validation for splitting the train dataset in several folds, one of which is used for validation for every model. The number of folds is chosen to be $k = 5$.

The MLP model heavily depends on the structure of its network, i.e. on its number of layers, the size of each layer, and the activation functions. These dependencies can be considered as hyperparameters of the MLP model.

After running a series of tests with different number of layers, the size of each layer, and the activation functions, it was decided to use a fully connected network with 2 hidden layers in between the 3072-dimensional input layer x and the 10-dimensional output layer y. it was decided to use a rectified linear unit (ReLU) function defined in Equation 1 as the activation function in all the layers instead of the standard logistic function. The reason for this choice lies in the gradient of the logistic function approaching zero for a significant value of z, while the ReLU function's gradient remains the same constant non-zero value when it is active and zero when it is inactive.

$$h(z) = \max(0, z). \tag{1}$$

As for the loss function, it was decided to apply the cross-entropy loss function, but with the softmax function instead of the logistic function as the multi-class task is to be solved. This loss is defined as follows:

$$L_{CE}(y, w^\top x) = -y\log(\hat{y}) - (1-y)\log(1-\hat{y}), \tag{2}$$

where $\hat{y}_c = \text{softmax}(x)_c = \frac{e^{z_c}}{\Sigma_{c'=1}^{C} e^{z_{c'}}}$ and $z = w^\top x$ is a logit, so $\Sigma_c \hat{y} = 1$.

It was decided to take 20% of the train data for validation purposes and use 15 epochs to predict target values of the validation set. First, we tried larger number of echos. we took a look at the validation and training losses and tracked their values and realized that 15 is our ideal number of epochs because after the 15th epochs the accuracy decreases constantly.

In addition to that, it was decided to apply a special training technique which takes its inspiration from the k-fold cross validation. In this approach, the train dataset was subdivided into $k = 5$ folds and then 15 epochs were run on every k split, i.e. the total number of epochs is 75. The idea behind this is in making the learning network to respond more accurately to the unseen data by enforcing different unseen data. The assumption is that this approach would make the MLP more adaptive than it was without it. This method is called the **Adjusted MLP** or **AMLP** in this work.

As for the CNN, after running different number of layers and activation functions, it was decided to implement a model with 3 convolutional and max pooling layers with the use of ReLU function as the activation function. Technically, we used Convolution2D in order to make a convolutional network that deals with images and MaxPooling2D to add the pooling layers. We also used some other functions such as Dropout for preventing the overfitting and Flatten for flattening the input and converting the pooled feature map to a single column [4]. Then we compiled the CNN using Compile function which needs three parameters; the optimizer, the loss function, and the metrics of performance. After running different optimizer like sgd, adam, rmsprop, we used adam optimizer which provided higher accuracy.

4. Discussion and Results

The resulting accuracy of all implemented models on the test portion of the CIFAR-10 dataset is shown in Table 1.

As mentioned in Subsection 3.3, it was decided to use ReLU as the activation function instead of the logistic function. Validation tests indicated the accuracy of the MLP with the logistic function to be 11% lower than of the MLP with ReLU.

The MLP was trained over 15 epochs with Figure 4 showing the training trend-line. Note that the validation accuracy tends to be lower than the train accuracy due to a slight overfit to the train data.

The accuracy of 56.19% obtained from running the tests with the MLP model called for some improvement. In attempt to do so, it was decided to apply the AMLP described in Subsection 3.3.

Model	Accuracy on the train set, %	Accuracy on the validation set, %	Accuracy on the test set, %
MLP	57.48	51.03	56.19
AMLP	83.66	72.56	81.44
CNN	76.18	73.09	73.20

Table 1: Accuracy of the models on the CIFAR-10. The winner is highlighted in green. Note the the accuracy on the validation set is an average accuracy accross $k = 5$ folds, for MLP and AMLP, and highest one amongst all epochs.

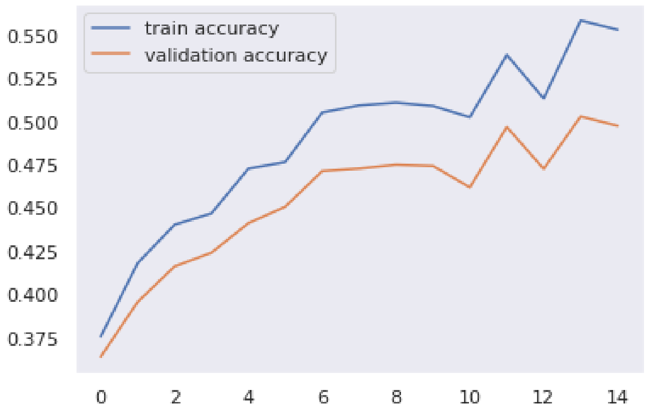

Figure 4: Training trend-line of the MLP model running showing the dependency of the train and validation accuracy on the number of passed epochs

Although the accuracy of the AMLP is significantly higher than of the regular MLP and reaches the encouraging 81.44%, it is not completely clear whether it is only due to the 5 times increase of the number of epochs or splitting the data into k folds actually assits the model and makes it more accurate. This calls for further investigation of this approach. Figure 5 showing the training trend-line.

On the other hand, the CNC was trained over 15 echos with Figure 6 showing the trend-line. After running the CNC model on the test dataset, the accuracy of 73% was resulted .

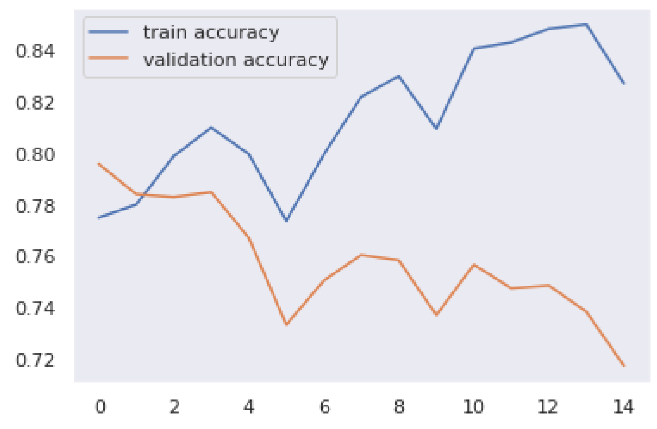

Figure 5: Training trend-line of the last set of 15 epochs of the AMLP model running showing the dependency of the train and validation accuracy on the number of passed epochs

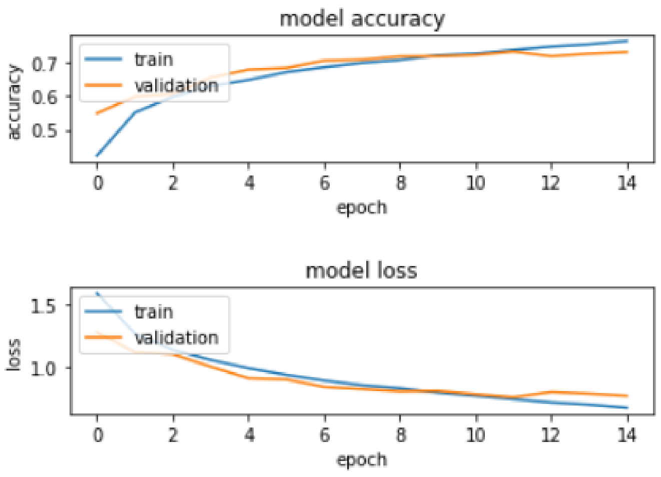

Figure 6: Training trend-line of the CNC model running showing the dependency of the train and validation accuracy/loss on the number of passed epochs

There are another possible ways to improve the performance of the models. For example, a better tuned feature detection algorithm could be applied to the input images, *e.g.* the entropy-based an entropy-based feature detector of Kadir and Brady [5] could be used for selecting regions and their scale within the image.

Applying principal component analysis (PCA) could potentially improve the computational efficiency of the methods. PCA is based on finding an optimal lower dimensional subspace in which to represent data of high dimension.

These potential improvements were adapted and have proven to be extremely efficient by Fergus *et al.* [1].

Summarizing, the AMLP shows the best results on the CIFAR-10 dataset among the others presented models with the accuracy of 81.44%.

References

[1] R. Fergus, P. Perona, and A. Zisserman. Object class recognition by unsupervised scale-invariant learning. In *Proc. CVPR*, volume 2, 2003.

[2] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[3] Google. Colaboratory. https://research.google.com/colaboratory/faq. Accessed: 2020-03-10.

[4] Heartbeat. Convolutional neural networks: An intro tutorial. https://heartbeat.fritz.ai/a-beginners-guide-to-convolutional-neural-networks-cnn-cf26c5ee17ed. Accessed: 2020-04-18.

[5] T. Kadir and M. Brady. Scale, saliency and image description. *IJCV*, 45(2):83–104, 2001.

[6] Kaggle. CNN from scratch. https://www.kaggle.com/aninditapani/cifar-10-cnn-from-scratch. Accessed: 2020-04-08.

[7] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[8] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[9] Reihaneh Rabbany and Siamak Ravanbakhsh. McGill University COMP-551, Lecture notes: Applied machine learning, Winter 2020.

[10] Rikiya Yamashita et al. Convolutional neural networks: an overview and application in radiology. insights into imaging 9.4. 2018.