Hybrid models as an alternative to traditional Convolutional Neural Networks for image classification

Jonatan H. Hanssen
Bachelor Student, Robotics and Intelligent Systems
Department of Informatics
The faculty of Mathematics and Natural Sciences
Email: jonatahh@ifi.uio.no

Eric E. Reber
Bachelor Student, Robotics and Intelligent Systems
Department of Informatics
The faculty of Mathematics and Natural Sciences
Email: ericer@ifi.uio.no

Gregor Kajda
Bachelor Student, Robotics and Intelligent Systems
Department of Informatics
The faculty of Mathematics and Natural Sciences
Email: grzegork@ifi.uio.no

1 Abstract

We found a dog driving a car down Hollywood Boulevard. It had sunglasses on, but didn't know where to go due to demensionality reduction. It was computing many computinations but due to severe liminations it had problems with binaly classifination and a large emesee.

2 Introduction

The task of image recognition and classification is a great challenge in machine learning, as there are a large number of features present in the data which slow down our training. For a relatively small one-channel image of 200 by 200 pixels, the naive approach of using every pixel as a feature becomes unfeasable, as we now have 40000 features. Clearly, some form of dimension reduction must be used to rectify this problem, and Convolutional Neural Networks are regarded as the gold standard for this. CNNs are able to reduce the dimensionality while extracting useful information in each layer, for example by finding edges in the first layer, combinations of edges that make a shape in the next layer, and combinations of shapes that resemble an object in the last layer. However, CNNs have many hyperparameters that need to be adjusted to achieve optimal performance, and a slow and computationally intensive training process. In this paper, we explore alternative models for image classification, using Principal Component Analysis and Random Forests in combination with parts of the traditional CNN model. We will investigate how these architectures perform for a simple image classification problem such as the MNIST dataset versus a more difficult classification problem in the form of pneumonia prediction from X-ray images.

More specifically, we will compare four different architectures. First we will use a traditional CNN, where a series convolutional layers is followed by a series fully connected layers. Then we will modify our CNN to create a CNN-RF model, where the features extracted by the convolutional layers are fed into a Random Forest instead of a series of fully connected layers. Finally, we will make use of Principal Component Analysis instead of convolution for dimensionality reduction, feeding the results of this stage into either a Neural Network or a Random Forest. In doing this, we hope to answer the following question: what is the best way to tackle the challenges associated with image classification?

First, we will give an explanation of the theory and method used in this paper, followed up with a detailed discussion of our results. Finally, we will arrive at a conclusion which summarizes our core results and lessons.

3 Method

3.1 Theory

This sections covers the theoretic background which underlies the methods used in this paper.

3.1.1 Convolutional Neural Networks

Convolutional Neural Networks are neural networks which are specialized to work on images, or other types of data where the structure of the data carries meaning. A normal neural network treats each input independently, and inputs are flattened such that any spatial information is lost. CNNs, on the other hand, explicitly take the spatial dimensions into account, by using convolutional kernels and pooling layers as part of their architecture.

The eponymous operation of convolution is a method which is used much in image processing. It is performed by using a convolutional kernel, which is applied to each pixel of the image. The kernel is a matrix of values, and the output of a particular pixel is the sum of all the pixels in its neighbourhood multiplied by the overlapping value in the kernel. As such, the value in any one position is dependent on the kernel, but also on the pixels around this position. This allows us to extract valuable information about images, such as edges or patterns. A basic kernel could be the Laplacian kernel (Eq. 1), which will output a high value when the pixel in the middle of the kernel has a value which differs greatly from the pixels in a 3 by 3 neighbourhood around it.

$$laplace: \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \tag{1}$$

The mathematical equation for the convolution of a kernel w on an image $f(x,y)$ (represented as a discrete function in two variables) is as follows:

$$w \star f(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} f(x+s, y+t)^{1} \tag{2}$$

In a convolutional neural network, the values of each kernel is initialized randomly, and training involves tuning these values. When performing convolution at the edges of an image, some values in the image will be undefined, i.e some values of the kernel will not overlap with the image. To solve this problem, we can either ignore any positions where the kernel is not fully contained within the image, or introduce padding, where we pad the image with values (usually just zeroes) so that more of the image can be convolved. By choosing the amount of padding, we can decide how close the dimensions of the result of our convolution are to the original dimensions. Another parameter which has an effect on the output dimensions is the *stride* parameter, which decides the size of the step between each convolution operation. By having a stride higher than one, we are no longer evaluating at every position. This reduces the computational complexity at the cost of not extracting our features as finely [Goodfellow et al., 2016, 343]. This will also downsample the result of our convolution to a smaller dimension than the input.

The convolution stage is often followed by an activation function, like the RELU function. After this, it is common to use a *pooling layer*. This layer replaces the output at a certain location with a summary statistic of the nearby outputs [Goodfellow et al., 2016, 335]. We could for example replace the value at a given position with the maximum value in a neighbourhood around this position, which is known as *max pooling*. By using pooling, we are making our output invariant to small translations of the input. This is useful if we care more about whether a feature is present rather than exactly where it is [Goodfellow et al., 2016, 336]. Furthermore, by using a stride parameter here as well, we are able to downsample our output.

After several layers of convolution, the input has been usually been downsampled, and important features about the input have hopefully been extracted by use of convolution and pooling. The output is then flattened, and the features are commonly sent to one or more fully connected layers.

[1] In reality, we should have $x-s$ and $y-t$, as correlation involves rotating the kernel by 180 degrees. However, the term convolution is often used for both the mathematical operations of correlation and convolution [Gonzales and Woods, 2018, 160]

3.1.2 Principal Component Analysis

Principal Component Analysis is a dimensionality reduction method which projects our dataset onto a lower dimension, while capturing as much of the variance of our data as possible. The central idea is to change the basis we use to represent our dataset, using vectors pointing in the principal directions of our dataset, instead of the unit vectors in feature space. The principal directions point in the direction of the variance of our dataset. By simply changing our basis to these vectors, called principal components, we have not achieved any dimensionality reduction, but by discarding directions of very little variance, we can reduce dimensionality without losing much variance (and thus information). By performing a Singular Value Decomposition we have everything we need to achieve this. The SVD is as follows:

$$X = UDV^T \tag{3}$$

Here, the coloumns U is known as the left singular vectors of X, D is a diagonal matrix containing the singular values of X (the square roots of the eigenvalues of X^TX) and the coloumns in V are known as the right singular values. The right singular values are actually the principal components we need, ordered by the variance in that direction. Thus, we can project our dataset down to a subspace spanned by a subset of these vectors (for example the first 3 vectors) and reduce the dimensionality of our dataset while losing as little information as possible. Furthermore, the coloumn vectors of V are ortogonal, and as such we can project our dataset down to this subspace with a single matrix multiplication:

$$\hat{X} = XV \tag{4}$$

3.1.3 Decision Trees

A decision tree is an intiutive way of performing both classification and regression tasks, by asking a series of yes or no questions about an instance in the dataset. These questions will split the original dataset, the root node, into child nodes which contain subsets of the dataset, and these nodes will be split again, creating a tree structure which terminates in leaf nodes which imply a choice of class.

For classification problems, the algorithm for creating a decision tree is quite simple. First, consider each feature of the dataset, and attempt to split all instances into two child nodes based on a simple true/false statement about this feature. Having done this for every feature, calculate the degree of homogeneity[2] of the two child nodes created, when considering the actual class of the instances. The feature which created the most homegenous split, is the feature which has the largest effect on the final class, and as such should be the first feature we use to split the dataset. After this, we have two subsets, and we perform the same algorithm on these subsets again. This algorithm terminates when it is not possible to split the data further, because all nodes only contain instances belonging to the same class. In practice, it is advised to stop the algorithm earlier by introducing limits to the depth of the tree, or limits to how few instances can be in one node. Once the algorithm has terminated, we have a series of leaf nodes containing subsets of the data.

Once a tree has been built, prediction is done by taking the instance and propagating it down the tree by answering the true/false statements that have been decided on in the creation of the tree. Once we have reached a leaf node, we predict that this instance is the class to which the majority of the instances in the leaf node belong to.

Decision trees are intuitive and computationally cheap to create, not requiring any training due to the deterministic nature of the algorithm. Furthermore, they require little tuning and have few hyperparameters. However, they are prone to overfitting.

3.1.4 Random Forests

A Random Forest is a type of ensemble method used to improve the performance of a single decision tree, by using bagging. Instead of using a single tree, a random forest consists of many similar trees, and the final class is determined majority voting by all the trees. Like decision trees, they are relatively simple to understand.

To create a random forest, we perform a bootstrap of the original dataset, creating many similar instances with small permutations introduced by resampling with replacement. We now create one decision tree per bootstrap, only using a random subset of the total amount of features every time we perform a split[3]. The result of this is that we create many slightly

[2]This can described by calculating the Gini-factor, or by calculating the entropy of the set

[3]The number of features used is often set to $\sqrt{p}$

different trees. These trees give predictions which are worse than simply creating a single decision tree using all the features and all the data, but when basing our prediction on the majority vote of all the trees, we are able to make predictions which are more generalizable and less prone to overfitting.

3.1.5 Datasets

Below is a description of each of the datasets used in this report.

3.1.6 MNIST

The MNIST dataset is a well known multiclass classification problem used to test machine learning algorithms. It consists of 70000 black and white images of hand drawn digits, each 28 by 28 pixels. Each image is labeled with one of ten classes, corresponding to the digit it depicts. The dataset is divided into 60000 training images and 10000 validation images. This dataset is essentially uniformly distributed, with all classes containing around $9 - 11\%$ of the total dataset.

3.1.7 Guangzhou Chest X-Ray Pneumonia Dataset

The Guangzhou Chest X-Ray Pneumonia Dataset consists of 5856 X-Ray images of children between the ages of one to five. The images were taken from the Woman and Children's Medical Center in Guangzhou, China. The dataset is divided into a training set of 5232 images and 624 validation images. The dataset is not evenly distributed, having 74.2% instances of pneumonia in the training set and 62.5% instances of pneumonia in the validation set.

3.2 Implementation

Our implementation can be found at `github.com/Gregz9/CNN_vs_weak.git`.

All neural network stages were implemented using the Tensorflow library. PCA was implemented as a custom layer using the Keras API, or by mapping the operation to the whole dataset using Tensorflow's `dataset.map` functionality, depending on the model. Random Forests were implemented using the Tensorflow Decision Forests library.

4 Results

We found so much stuff man

4.1 Traditional Convolutional Neural Network

4.2 Convolution followed by a Random Forest

4.3 Principal Component Analysis followed by a Neural Network

4.4 Principal Component Analysis followed by a Random Forest

5 Conclusion

I have realized that I am cooler than Eric.

References

[Gonzales and Woods, 2018] Gonzales, R. C. and Woods, R. E. (2018). *Digital Image Processing*. Pearson, New Nork, NY.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

Appendix

Here are some pictures of Eric Reber, formerly known as the criminal Ed "Gurgler" Bickley of Texas State Penistentiary.

Fig. 1. Eric Reber in his most devious form