Transfer Learning and Convolutional Neural Networks (TLCNN) by Dan Hoogasian

Abstract

The popular task of image classification utilizes some state-of-the-art deep neural networks to achieve many goals. With the more readily available increased computing power, more and more individuals are entering the field, and this has led to a need for more customizable machine learning APIs. Convolutional neural networks excel at image classification and are the most widely used model type for such tasks. There are many already developed and pre-trained CNNs available for use, but many people prefer to build their own models.

This paper explores some of the fundamental building blocks of CNNs and what makes them ideal for image classification. It then presents two example models built for different datasets and discusses the results. Finally, some advantages and limitations of implementing transfer learning are explored.

Introduction

The potential applications of image classification have grown rapidly. Applications range from medical images (e.g., MRIs, X-rays, and CT scans), quality control in product manufacturing, forestry management, and automotive transportation. Many already developed and pre-trained models exist that can be applied to a variety of applications, but there is a growing demand of people to build their own models. The ability of APIs to accommodate this demand is limited and currently presents numerous challenges, one of which is the ability to apply a user-designed deep neural network architecture to a variety of differently sized images.

Building a high-quality model from scratch (i.e., without using the architecture or weights from a prebuilt model) is time consuming and requires a comprehensive understanding of the field. With the recent increase in computing power, more people have experimented with the potential of such models. This paper investigates what is required to build a quality model from scratch and how to apply such models to images of different sizes.

Datasets

Models for two datasets were developed for this project. The first dataset is the popular CIFAR-10 dataset, which is a subset of the Tiny Images dataset originally created by the MIT Computer Science & Artificial Intelligence Laboratory. This subset contains 60,000 labeled 32x32 color images of the following ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. The dataset was accessed directly through Keras but is also available through the department's website [5].

The second is the STL-10 dataset made available by the Stanford University Computer Science department. It contains 13,000 labeled training and testing images, and 100,000 unlabeled images. The dataset is primarily used for unsupervised learning but was used here because of the similarity it has to the CIFAR-10 dataset (the unlabeled data were excluded for this project). The data consists of 96x96 color images of the following ten classes: airplane, bird, car, cat, deer, dog, horse, monkey, ship, and truck. The mutually exclusive monkey and frog classifications were dropped from both datasets, to improve model transferability. This dataset is available for download through the department's website (the script supplied on the webpage was used for the download) [2].

Methods and Descriptions

The Keras API in conjunction with TensorFlow were used, as they are one of the most popular and customizable machine learning solutions. Many users have successfully built state-of-the-art deep learning model through Keras. Keras also has many built-in datasets and models, making it a good option to research and test various models.

A Convolutional Neural Network was used as the model type since it is the most popular machine learning model for image classification [6]. It is popular for numerous reasons, most notably, its speed. One of the biggest hurdles in image classification is the training time, since each pixel must go through many calculations until an optimal weight is reached. This also makes the model slow once deployed and placed into use.

Using a traditional deep neural network like a multi-layer perceptron can work for many applications, but they will often take so long that it isn't practical. A CNN incorporates multiple concepts that greatly speed up the process. CNNs quickly pull out the most identifying features by dividing an image into much smaller parts, called filters or kernels, often of a 3x3 or 5x5 size. This filter is slid across an image in small steps or strides, performing convolution and backpropagation until an optimal weight is reached. This is done for each filter, for each layer and after completion, only the filter with the highest value is kept (i.e., the most important feature). The number of filters is often 32, 64, or 128, but it can get higher or smaller. These are called the convolution layers, of which there are usually at least four-of, and sometimes up to 20. This greatly reduces the total number of parameters (or neurons) required to represent the image, reducing the memory requirements and the number of computations [3].

Another technique CNNs use is pooling. In pooling, a pooling kernel (often of a 2x2 or 4x4 size) is slid across the image. At each step, calculations are made and only one value or pixel number is kept, reducing the image resolution.

For the first model, a rather simple CNN was used. It was adapted from a blog posting on efficient CNNs for the CIFAR-10 dataset (the smaller of the two datasets) [1]. It consisted of four convolutional layers, each followed by a pooling layer and a batch normalization layer, all followed by a hidden dense layer, followed by dropout, then a dense output layer.

For the STL-10 dataset, a more complex approach was taken, with the goal of achieving the highest accuracy. To do this, hyperparameters were chosen via a random search through a Keras tuner. The hyperparameters that were chosen were the learning rate, the optimizer, the number of filters, the number of neurons, the number of convolution layers, and the number of hidden dense layers.

Results

The evaluation metrics for both models were categorical accuracy for the accuracy and categorical cross-entropy for the loss. For the final training, each model was run for 30 epochs. The CIFAR-10 model had an accuracy of about 80% and a loss of about 1.0. The STL-10 model's accuracy was about 71% with a loss of about 1.18. Although these were not impressive scores, they were sufficient for the main objectives of the project.

Transfer learning is where the main architecture and trained weights of a model are taken from one application and transferred to another. Many tasks in the industry are similar enough where this can be an efficient and accurate method. Unfortunately, choosing an efficient and compatible model can be a hard task. Data often have differences (e.g., image size, image color, file formats, and label types) that hinder model transferability. Many models available for transfer learning have a parameter where one can simply change the input image size. A good example is the input_tensor parameter for the VGG16 model. This parameter likely resizes images with a layer like tf.keras.layers.Resizing. Resizing is simple, but many image details can be lost in the process. This project explored the possibility of retaining the original image size, hoping to not use some weights (if the new images were smaller than the originals), reusing the same weights multiple times per filter (if the new images were larger), or something of the likes.

Since CNNs break an image into numerous smaller components (i.e., filters), like discussed previously, this seems doable. Unfortunately, implementing this was not easy and not achieved. This is partially because all layers depend upon the input shape given to the beginning layers and models are saved in their entirety, making them difficult to dissect later [4].

Numerous methods were either explored, developed, and tried, but none were successful. The first method attempted was changing the input layer (and its input shape). The second method was creating a new input layer to accommodate a differently sized input. The third method was including an input shape parameter in the model itself. Lastly, was an attempt to convert the model to a JSON file, updating the model, then converting it back to a Keras model. The third method seemed the most probable, but this would likely require creating a model class, with more customizable parameters, which is essentially what the prebuilt model applications in Keras are.

Discussion

Being able to develop a tailored model from scratch is a valuable capability. Some example scenarios are if the dataset and application are unique, the user wants to achieve high performance, or it is being used for educational purposes. Transfer learning is also tremendously valuable, for both user-built models and pre-built models. Image size is important to think about if considering transfer learning, for both applications, and is a good topic for further research and development for CNNs.

References:

[1] Brahmane, A. (2020, October 26). *Deep Learning with CIFAR-10*. Towards Data Science. https://towardsdatascience.com/deep-learning-with-cifar-10-image-classification-64ab92110d79

[2] Coates, A., Lee, H., & Ng, A. Y. (2011). *An Analysis of Single-Layer Networks in Unsupervised Feature Learning*. AISTATS. https://cs.stanford.edu/~acoates/papers/coatesleeng_aistats_2011.pdf

[3] Geron, A. (2023). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems* (3rd ed.). O'Reilly Media.

[4] Kyrkou, C. (2019, August 20). *Changing input size of pre-trained models in Keras*. Medium. https://ckyrkou.medium.com/changing-input-size-of-pre-trained-models-in-keras-3dfbe3ca3091

[5] Krizhevsky, A. (2009). *Learning Multiple Layers of Features from Tiny Images*. [Master's Thesis, University of Toronto]. http://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf

[6] Le, J. (2018, October 7). *The 4 convolutional neural network models that can classify your fashion images*. Towards Data Science. https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d