***Neural Network Graph Classifier of Probability Distributions***

Patrick Spohr

589909

20.02.2024

Master’s Program

Finanzmathematik, Aktuarwissenschaften und Risikomanagement

Department

Informatik, Kommunikation und Wirtschaft

Course

Deep Learning Seminar

ProfessorDr. Alla Petukhina

Table of Contents

[Motivation 1](#_Toc158119807)

[Graph Classifier 1](#_Toc158119808)

[Literature Review 2](#_Toc158119809)

[Objectives 3](#_Toc158119810)

[Main Goals 3](#_Toc158119811)

[Initial Problems 3](#_Toc158119812)

[Data 6](#_Toc158119813)

[CIFAR-10 Images 6](#_Toc158119814)

[Scraped Graphs 6](#_Toc158119815)

[Generated Graphs 7](#_Toc158119816)

[Methods 9](#_Toc158119817)

[Overview 9](#_Toc158119818)

[Feed-Forward Neural Networks 9](#_Toc158119819)

[Convolutional Neural Networks 10](#_Toc158119820)

[CNNs vs. FNNs 10](#_Toc158119821)

[CNN Layers 11](#_Toc158119822)

[Max Pooling Layers 12](#_Toc158119823)

[Rectified Linear Units 12](#_Toc158119824)

[Cross-Entropy 12](#_Toc158119825)

[Adam Optimizer 12](#_Toc158119826)

[Python, Keras, and Libraries 12](#_Toc158119827)

[Simple Graph Classifiers 12](#_Toc158119828)

[Distribution Graph Classifiers 13](#_Toc158119829)

[Results 14](#_Toc158119830)

[Simple Graph Classifiers 14](#_Toc158119831)

[Distribution Graph Classifiers 14](#_Toc158119832)

[Conclusions and Further Research 14](#_Toc158119833)

[References 15](#_Toc158119834)

# Motivation

## Graph Classifier

In the age of information, those that can successfully and quickly interpret large amounts of data will have an advantage. The field where converting data into visual aids is called descriptive statistics and the most common method to do this is with graphs. This follows the adage, “A picture says a thousand words.” With just a glance someone can understand and make connections about the data without needing to scan through a table of numbers. There are many different kinds of graphs: bar charts, pie, charts, line graphs, scatter plots, and radar charts to name a few. Within those charts they can be divided up into more types, like stacked bar charts or donut charts. From science to business charts are the main way key parts of the data are shown to other researchers, customers, or investors.

The McDonald’s 2022 Annual Report \cite{mcdonalds2022} has a total of 73 pages and 8 graphs. Each graph displays important information to the investor, like the figure which shows total revenues by segment on page 13. One can see that from 2020 to 2022 the revenue from international operated markets rose and fell, but overall, the revenue share from the different segments have remained static. Wouldn’t it be beneficial to have access to these graphs immediately without needing to browse through the whole report? The ability to identify and then classify graphs instantly in the report without needing to read through it and snip out the graphs manually would save time and effort. The overall flow of this system would have three parts: Identify, Locate, and Organize. Identify would simply see if it is a graph or not and classify it into a type of graph. Once that is finished then the graphs can be located on a page and extracted. The last part would organize the charts for easy viewing. My focus is on the Identify part of this system and will be limited to building a model that answers two questions: Is this image a graph? What kind of graph is it? Regarding the second question, I’ll limit the scope even further because of the multitude of different graphs. The focus will be on classifying the probability distribution in a graph. This was chosen because of the novelty of the problem – no similar projects were found – and because of my current study of stochastics.

## Literature Review

I’ll need to briefly mention the literature I tried to find and the literature that guided this project. As mentioned earlier, this project is quite novel in its end goal since I did not find any research that specifically tried to classify graphs into probability distributions. There were projects that classified graphs into bar charts, pie charts, etc., but even then, those projects were not part of published research. When one tries to search for a neural network graph classifier or a similar pattern the first results are about Graph Neural Networks or GNNs, which are unrelated. As for image classification itself, there are a plethora of papers and resources. I personally learned a lot from a Stanford University course \cite{cs231n2023} and \textit{An Introduction to Convolutional Neural Networks} from O’Shea and Nash \cite{oshea2015}.

# Objectives

## Main Goals

As stated previously, my focus is on the identification and classification of graph images. This can be accomplished in two types of models where the first one classifies a graph image from a natural image (non-graph) and the second one classifies a graph image into its probability distribution. There are many probability distributions to choose from, but I selected the normal distribution, log-normal distribution, exponential distribution, and uniform distribution. I primarily choose these distributions because they are common and relatively distinct from each other. The reasoning behind this two-model process is to be able to produce a baseline result where it is known that graphs can be distinguishable from natural images. I predicted early on that graphs would be quite distinguishable from natural images and the accuracy of this model would be high, but as it will be discussed later, there is a second component to this problem specific to my data. These hurdles will be expanded upon in the next section.

In summary, the workflow of the project was split into four steps: get data, create graph classifiers, create distribution classifiers, and evaluate the models. Getting data had the potential to be the hardest step of this project because of the novelty of its goals to classify graph images into their probability distributions and I was partly correct. The next major obstacle was choosing the right model to build the machine learning models and the method in which each image would be classified. In the “Methods” section, I will go in depth to explain the process of how the models were built and the images classified.

## Initial Problems

As image classification is a vibrant and bustling field, there is no shortage of natural images of various types of things – cars, airplanes, horses, cats, dogs, clothes, etc. One of the most common datasets for natural images is the CIFAR-10 \cite{ krizhevsky2009} as it is well labeled and contains various images. The task of finding graph images was much more difficult. However, there was a suitable dataset of graphs on Kaggle from SunEdition \cite{sunedition2021} which met my requirements to build the first model, which simply classifies an image as a graph or not a graph. Unfortunately, this dataset does not contain classes with specific probability distributions and cannot be used for the second model – classifying the four chosen distributions in graphs.

The biggest issue was getting a dataset with the normal distribution (norm), log-normal (lognorm) distribution, exponential distribution (exp), and uniform distribution (unif) as its four classes. As I saw it, this could be solved in two possible ways. First, I could have scraped the graphs from various online sources. This leads to a major issue of not only where to scrape the graphs but how to label them time efficiently. The dataset from SunEdition on Kaggle has some major drawbacks when wanting to classify the images into more specific classes. These would be apparent in my own scraped graphs as well. Many graphs have more than one graph type in the image. That makes the correct classification impossible. Additionally, the dataset includes ‘graph-like’ images which are depictions of graphs but have no numerical value and are closer to art or natural images. I could also mention that even in a graph type, like bar charts, there are subcategories – stacked bar charts, horizontal bar charts, vertical bar charts, etc. Were I try to scrape the graphs, then I would lose control of the variables and spend most of my time labeling each graph which would be impossible for the timeframe of the project. This leads to the second solution to my problem. I could generate the graphs.

Generating graphs with the aid of a graphing library was the most time effective solution to acquire thousands of graphs with the four distributions in already-labeled classes. This is not the ideal solution since it biases the dataset towards the design conventions of the chosen graphing library and it lacks the variety seen online, but it can be highly randomized to mimic the potential graphs in various sources and the variables can be controlled to ensure all graphs have the potential to be classified. The most important consideration in this method is the time saved by generating the graphs. The daunting task of scraping and labeling 10,000 graphs can now be minimized into writing a script using a graphing library. The specifics of the three datasets used will be further explained in the subsequent section.

# Data

## CIFAR-10 Images

The CIFAR in CIFAR-10 stands for the Canadian Institute for Advanced Research and the 10 represents the 10 classes in the dataset: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck \cite{krizhevsky2009}. The data comes in six batches of 10,000 images with a total of 60,000 images. The 10,000 images in each batch are made up of 10 classes with 1,000 images per class. The images are colored and 32x32, which means they have a shape of (32,32,3). The first two values represent the dimensions and the 3 stands for the red, green, and blue channels. This data structure is known as a tensor. However, the original data type is a flat NumPy array of (3072,), so each image needed to be reshaped to fit in the model with the graphs.

The updated dataset has 10,000 images (first-batch) that are all 32x32 and in JPG format. Also, the 10 classes have been combined into one natural image class. I have only selected 10,000 because there are only 7,753 suitable graphs in the scraped graph dataset. Had I selected more that would have led to an imbalanced dataset in the model. In my project, the CIFAR-10 dataset will be labeled as CIFAR or natural.

## Scraped Graphs

Just like the CIFAR-10 dataset, the scraped dataset \cite{sunedition2021} needed to get updated to work with the simple graph classifier model as well. The original dataset has 15,786 images and 8 classes. The classes are just image, bar chart, diagram, flow chart, graph, growth chart, pie chart, table. As you can see ‘just image’ refers to non-graphs or natural images, so it needed to be removed. Other classes like table, flow chart, and growth chart were not necessary and were removed as well.

In the end, four classes were kept, bar chart, diagram, graph, and pie chart, so the total images now used were 7,753. The images came in various dimensions and were resized to the natural images 32x32 and converted to JPG format. I’ll note here that JPG was chosen because it only has 3 color channels and no alpha channel. If PNG is used then it could have a shape of (32,32,4) which adds unnecessary parameters for the scope of my objects.

## Generated Graphs

Lastly, 8000 graphs were generated in a 460x345 dimension and JPG format. The graphing library used was Matplotlib in Python, but I’ll discuss the programs and libraries in more detail in the “Methods” section. There were 4 classes – norm, lognorm, exp, and unif – with 2,000 images each. The only thing updated were the dimensions which included 32x32, 115x86, and 153x115. For clarity, these dimensions follow the Windows format of width by height.

As for the design, almost every part of the graph was randomized. The color of the histogram bars, line, figure color (outside the box), face color (inside the box where the plot is), and text labels were changed, but the figure color and face color were biased to white by 20 percent because most graphs tend to be white. There was also some changes to ensure that the histogram bars, line, and face don’t have the same color. Regarding the histogram and line, both were visible or only one was visible. The line style was solid or a dash. Each graph had a title, y label, and x label which consisted of lorum ipsum, which is Latin gibberish and is commonly used in web design. The idea behind this was to avoid a bias to English if someone used this model to predict graphs with different languages. The text also had randomized size. Lastly, the density function parameters were randomized to ensure many variations of each probability distribution.

Overall, the graphs looked like something that could be found in a report, book, or online. The biggest issue with the generated graphs were the so-called ‘duds’ which looked blank because the color of the face, histogram, and line were too similar. This wasn’t a large amount of images, so I decided to keep them instead of manually deleting all of them. I reasoned that keeping the original generated the same is better for reproducibility, rather than someone, me, deciding if a graph is a dud or not since some graphs are on the border of being a dud or not. Ultimately, the seed of the graph generator remains the same and the dataset can always be recreated and used in the model.

Early on, I pre-identified possible misclassifications once I scanned through the set of generated graphs. The similarities between some of the graphs, especially the log-normal and normal distribution, were considerable. They were not ‘duds’ like previously mentioned – they looked typical, but because of the density functions’ parameters, they were similar in appearance to a point where it would become a 50-50 decision if I had to classify them myself. The cases when this happened are a minority, but it was important to keep this in mind as I evaluated the models later.

Normal Distribution Density Function

Log-Normal Distribution Density Function

Exponential Distribution Density Function

Uniform Distribution Density Function

# Methods

## Overview

As an introduction to the methods used, I’ll briefly list the steps taken to achieve my objectives. Firstly, I reviewed literature available on artificial neural networks or ANNs and convolutional neural networks or CNNs and then I familiarized myself with TensorFlow, Keras, and Pillow (Python library for images). After that I found natural images (CIFAR) and scraped images (SCP). The next step was generating the graph dataset (GEN) in 32x32, 115x86, 153x115 dimensions. With all the data prepared, I trained simple graph classifiers using CIFAR, SCP, GEN datasets using 32x32 images. Those models were then evaluated by using saved Keras model to predict the images. Subsequently, I moved on to train the distribution graph classifiers using 32x32, 115x86, 153x115 dimensions. In order to see where the models of the distribution classifier models misclassified, I input all the images into the saved Keras models again. Lastly, I tested untrained images with the distribution classifier 153x115 which were not used in any model previously.

## Feed-Forward Neural Networks

The ANN process used in my model is called a feed-forward neural network or FNN. The structure of the model consists of three parts: an input layer, one or many hidden layers consisting of neurons, and an output layer. The two main processes in the model are forward-propagation and back-propagation, and can be summed up by their respective goals, producing an estimate and minimizing the loss. Forward-propagation begins once the model is built and the data is split into the input values x and the target values y. In this scenario the goal is to use x to predict y.

The input data starts from the input layer and flows into the neurons in the hidden layers where it gets multiplied by a weight. The process can be defined as the dot product between the input values and their respective weights. The product then goes into an activation function in the neuron so that the neuron ‘fires’, returning a modified value, or ‘rests’ – gives zero. The process is repeated through every hidden layer until the output value from the final hidden layer enters the output layer, where it is again put into an activation to determine the final prediction or estimate. The estimate is then put into a loss function to calculate the loss or difference between the estimate and target value. From here a process called back-propagation starts which goes from the back to the front of the entire model. The loss function gets minimized by an optimizer using gradient descent to achieve the smallest possible loss and this is done by changing the weights between the neurons for every layer and every connection. Forward-propagation and back-propagation are repeated until the loss is minimized enough to suit the goal of the model.

## Convolutional Neural Networks

### CNNs vs. FNNs

In order to classify images, the FNN model needs to be adjusted to take the tensor shape of images into account. This is where convolutional neural networks or CNN come into play. The biggest difference between convolutional neural networks or CNNs and a more standard FNN is that they are mostly used in image classification. CNNs enable encoded image-specific features into the architecture (location + color), making the network more suited for image-focused tasks while also reducing the parameters required to set up the model \cite{oshea2015}. Simply put, the overall structure of the model doesn’t change, meaning the model still has an input layer, hidden layers, and an output layer. Additionally, the main processes of forward-propagation and back-propagation remain untouched, however the hidden layers get tweaked considerably whereby the neurons don’t take one value, but a tensor and these neurons only connect to a small region of the layer preceding it. These new hidden layers are now called convolutional layers. The next major addition is another layer called a pooling layer, which comes after a convolutional layer. It would be pertinent to mention that in many CNNs a ReLU activation function is used instead of a more typical sigmoid activation function in FNNs.

### CNN Layers

The most important part of a CNN is its convolutional layers since they are the means in which the original image gets ‘boiled down’ into its ‘essence’. I use this analogy because one can imagine a set of ingredients, pixels making up an image, that get thrown into a boiling pot where the water evaporates leaving only the important flavors behind. This visceral example can be summed up more literally, where the original image gets transformed layer by layer from the original pixel values to the final class scores. By the end, the full image will be reduced into a single vector of class scores, arranged along the depth dimension. What are these class scores? They indicate the estimate for each class or y target. As an example, the final output layer for CIFAR-10 would have dimensions of (1, 1, 10) since it has a total of 10 classes that an image can be sorted into \cite{cs231n2023} – airplane, dog, cat, etc.

What is a convolutional layer exactly? The four important parts are the filter, kernel, stride, and padding. These are the parameters that will determine how the convolution is performed. First, the number of filters are chosen, also known as output filters, and they determine how many times the image gets passed through the process. One filter means one pass through and one output tensor, while 10 filters mean ten pass throughs and 10 output tensors. The next choice is the size of the filter, which is known as the kernel. An example kernel size is (3,3,3) and this works just like image dimensions, where the height is 3, the width is 3, and the RGB channels are 3. The kernel in the example can be visualized as something that slides over the image and only focuses on a 3-by-3 area at a time. Adding on to the previous example, if the model has 10 filters, then that means there would be 10 output tensors, but that doesn’t mean the output tensor shape would be (3, 3, 3). One needs the last two parameters, stride, and padding, to calculate the output tensor shape. The stride is a number that tells the kernel how many pixels it should move at a time over the image. A stride of one means the kernel moves across the image space 1 pixel at a time. The last parameter padding, or zero-padding, tells the model how many zeros to place around the image so that no vital information is lost when the kernel slides over the image. With the parameters decided, one can now calculate the output volume size or shape of the output tensor.

Output Volume Size Equation

### Max Pooling Layers

The convolutional layer is not the only new layer added in a CNN. The layer that often proceeds a convolutional layer is a pooling layer, or in the case of my model, a max pooling layer. Pooling is also known as downsampling and this is where a lower resolution version of an input signal is created. This method is used to retain the large or important structural elements without the fine detail that may not be as useful \cite{brownlee2020}. The outputs of the convolutional layers record the precise position of features but following small movements in the position of the feature, like a cat head slightly turned, will result in worse predictions, so max pooling is utilized, where only the highest value in every feature map or output filter goes on to the next layer. It can be noted that another pooling method, average pooling, can be used. It functions the same but averages the values in the feature patch.

Max Pooling Function

### Rectified Linear Units

As stated earlier, the activation function simply tells the neuron to ‘fire’ or ‘rest’ which in the case of the activation function I used, rectified linear unit function or ReLU, the value returned is either 0 if x < 0 or x if x > 0 \cite{agarap2019}. ReLU is used as an activation function for a few key reasons. First, the simple nature of the function, f(x) = max(0,x), eliminates complex calculations and reduces the processing demands. This is especially important in convolutional layers where the amount of parameters can skyrocket and increase the time needed to learn. This works in practice by promoting sparsity in the model and sparsity refers to the scenario where most of the cell entries in a matrix, or in this case tensor, are zero \cite{giskard}.

Figure of the Rectified Linear Unit \cite{agarap2019}

### Cross-Entropy

The next important topic to cover is the loss function used since the learning process can only happen if the model has a loss function to calculate the error. The loss function in the models I built was the cross-entropy loss function. These functions, used in classification problems, calculate how accurate the machine learning or deep learning model is by defining the difference between the estimated probability with our desired outcome \cite{365team2023}. These functions come in two versions, one for binary classification and one for categorical classification. My model uses categorical cross-entropy because it is a multi-class classification problem, which means there is only one correct class per x input or image.

Binary Cross-Entropy Function

Categorical Cross-Entropy Function

### Adam Optimizer

To wrap up this section over CNNs, the optimizer used to minimize the loss function needs to be mentioned. The optimizer is called gradient descent and it can be described as a first-order iterative algorithm for finding a local minimum of a differentiable multi-variable function. This is apposed to gradient ascent where the algorithm tries to find the local maximum. This can be visually imagined as a rock falling from the mountain side where it will stop once it hits the lowest point or if it gets stuck in a low point between two high points – which will tried to be avoided. The models use the Adam Optimizer and that is “…an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments \cite{kingma2015}.” The parameters used are alpha, beta1, beta2, and epsilon. Alpha is also known as the learning rate and determines the stepsize, and is usually very small at around 0.001, which is important, because the algorithm shouldn’t skip over the local minimum. The major advantages of the Adam optimizer are that it only requires first-order gradients and has relatively little memory requirements.

## Python, Keras, and Libraries

The implementation of this project wouldn’t be possible without the various tools used. I’ll describe them all briefly and list out the versions used for the sake of reproducibility.

## Simple Graph Classifiers

## Distribution Graph Classifiers

# Results

## Simple Graph Classifiers

## Distribution Graph Classifiers

## Conclusions and Further Research

# References