UKRAINIAN CATHOLIC UNIVERSITY

BACHELOR THESIS

Recommender systems for travel industry using visual information

Author:
Marko BURDA

Supervisor:
PhD Taras FIRMAN

A thesis submitted in fulfillment of the requirements for the degree of Bachelor of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2022

Declaration of Authorship

I, Marko BURDA, declare that this thesis titled, "Recommender systems for travel industry using visual information" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Faculty of Applied Sciences

Bachelor of Science

Recommender systems for travel industry using visual information

by Marko BURDA

Abstract

When choosing the right establishment to order from, we rely on the average ratings of those places. The ratings are calculated based on numeric values left by different users who have had experience with the establishment. The problem with such ratings is that having a high score does not guarantee one's satisfaction with a meal. It is a mere number evaluated based on other people's experiences. However, every human being is an individual with specific tastes, and the concept of good food differs for everyone. With the unexpected strike of the COVID-19 pandemic, there was also a sharp spike in need for food delivery services. For applications like Glovo or its competitors Rocket and Bolt Food, the only way of approximating the food quality is a score averaged from other users' ratings, which makes it, quite often, a hit-or-miss experience. Such services, unfortunately, neither offer written reviews nor any personalized recommendations based on your order history. This project is built around this topic and is aimed to train a model that learns from visual information and user behavior and produces recommendations based on given information.

Acknowledgements

I would like to thank my project advisor, Taras Firman, for supervising this work.

Contents

List of Figures

List of Abbreviations

ML	**M**achine **L**earning
CF	**C**ollaborative **F**iltering
CNN	**C**onvolutional **N**eural **N**etwork
ViT	**Vi**sion **T**ransformer
ResNet	**Res**idual **Net**work
CV	**C**omputer **V**ision
RGB	**R**ed **G**reen **B**lue
MLP	**M**ultilayer **P**erceptron
BST	**B**ehavioral **S**equence **T**ransformer

Dedicated to my loving family

Chapter 1

Introduction

As the internet became globally popularized in the 21st century, more than half the population of this planet is interconnected via an invisible network. In this day and age, thanks to modern technology, we are able to perform most of our daily tasks even without leaving our desks. Every citizen with a smartphone and an internet connection is able to communicate with close people and strangers from their homes, read books, watch movies, order delivery from restaurants, and even get groceries from the store. Many of our basic needs now require significantly less amount of effort, thanks to the internet.

Arguably, another major breakthrough in the pursuit of automation and improvement of quality of life was the widespread adoption and availability of machine learning technologies. The integration of artificial intelligence has skyrocketed over the years in the business industry. [Research, 2021] The global Artificial Intelligence (AI) in food and beverage market size was USD 3.33 Billion in 2020 and is expected to register a CAGR of 44.4% during the forecast period.

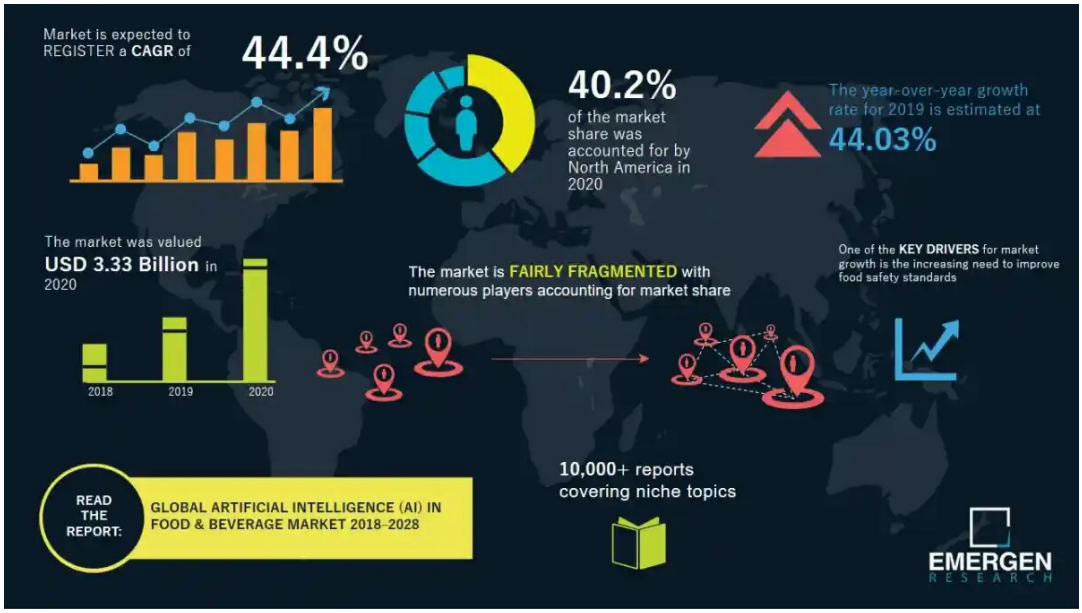

FIGURE 1.1: Artificial Intelligence in food and beverage market scope

Adopting machine learning allows businesses to optimize and automate processes, account for human error, and reduce labor costs. In Food and Beverage industry, it is used for various tasks like:

- Food supply chain optimization, price and inventory management predictions, machinery condition monitoring, and predictive maintenance are tasks that

can be replaced with the use of Artificial Intelligence, thus getting rid of the need for human labor in certain fields.

- Online services and applications. Automated customer service and relevant recommendations are both keys to improving the overall user experience.
- Market analysis. Tracking, categorizing, analyzing, and updating current trends and customer preferences make for an invaluable source of information for a business strategy.

With the rapid development of AI and its use in the food industry, many competing reviewing platforms are present on the internet. As mentioned before, analyzing the market and adapting to customers' tastes is an important factor in running a successful business. Thus recommender systems are extremely important for delivering the right content to the end-user.

Chapter 2

Background research

2.1 Types of Recommender Systems

Recommender systems can be mainly classified into two types: Collaborative Filtering and Content-based. In CF, users' previous interactions with items are used to yield new recommendations. CF can be further divided into User-based CF and Item-based CF.

Consider an example of User-based CF. Let there be an n $\times$ m matrix with n users u_i, $i = 1 \dots n$ and m items m_j, $j = 1 \dots m$. respectively. A value r_{ij} indicates a rating given by user u_i to item m_j. To predict ratings for items that have not been rated by a user u(i), users with similar ratings are taken, and a weighted average of ratings from these users is calculated to decide whether to recommend a certain item or not [Rocca, 2019].

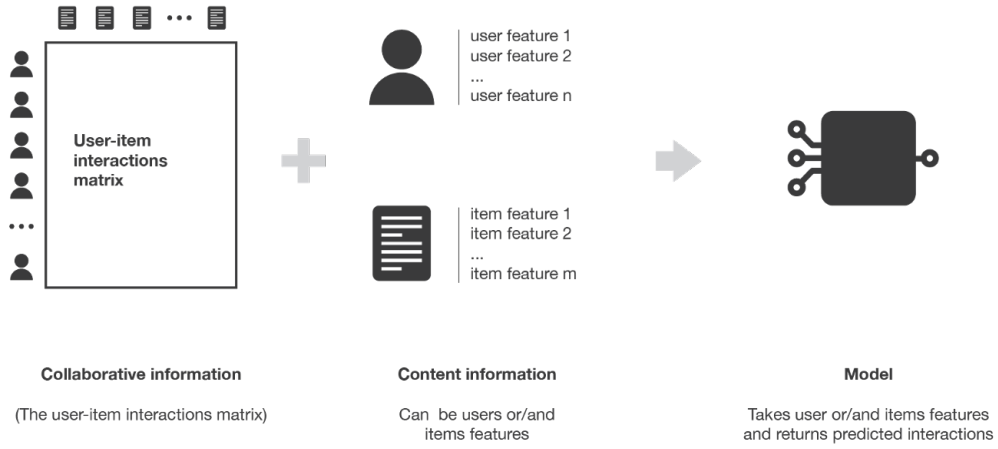

FIGURE 2.1: Collaborative Filtering overview

In the case of Item-based CF, similarities are instead discovered between items by looking at similar ratings given by the same user. By discovering pairs of similar items, the system is able to recommend items from these pairs if the user liked at least one of the items in the pair [Sarwar et al., 2001].

Of course, CF suffers from a cold-start problem. This means that at the early stages, such RS is obsolete since it lacks data about user's interactions to build its predictions on [Ricci, Rokach, and Shapira, 2011]. For such cases, platforms usually recommend new and popular items to new users, as shown in the example below.

Content-based filtering, on the other hand, uses item features to recommend other similar items to users.

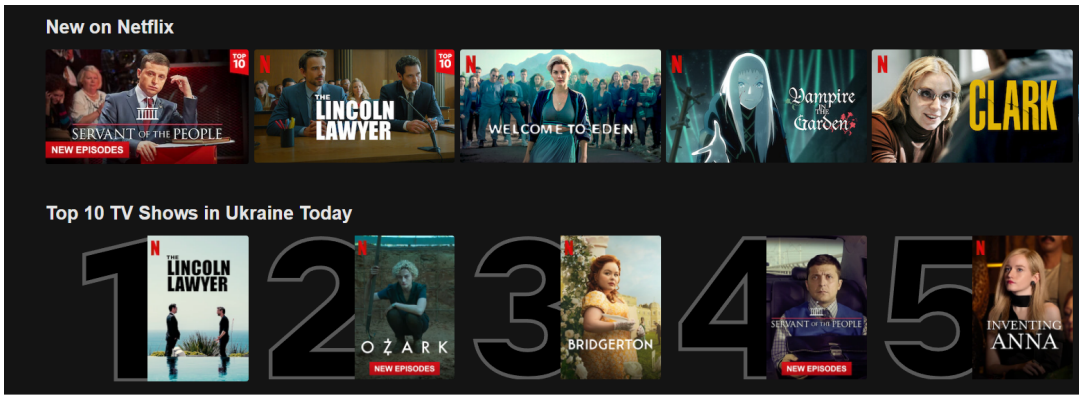

FIGURE 2.2: Netflix's "New & Popular" recommendations tab

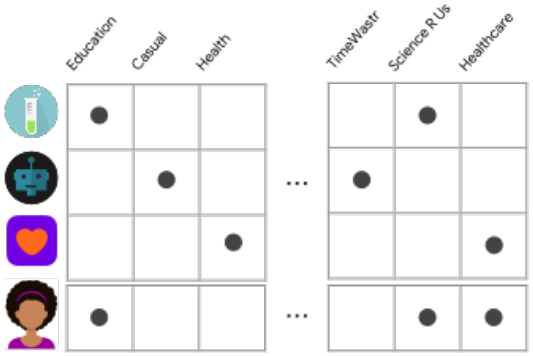

FIGURE 2.3: Content-based filtering matrix example

An easy example of that would be categories. The matrix in figure 2.3 represents different apps in columns and categories in rows, where each category assigned to an app is marked with a dot in its respective cell (this would be usually represented using binary in a real matrix)

By knowing what categories a user prefers, we can recommend other items based on some similarity metric. Features can be both implicit and explicit. For instance, music streaming services often prompt new users to select artists that they like to assign features like music genres to those users.

Main differences between CF and content-based filtering are:

- Content-based method requires a lot of information about features on the developer's end to discover similarities. CF does not need explicit categories since it relies solely on users' interaction history and sets underlying features implicitly.
- Content-based filtering does not require other users' data since recommendations are user-specific, which poses restrictions on further recommendations depending on existing user data.
- CF suffers from the cold-start problem severely when there is little interaction with an item. New items don't get recommended until someone rates them.

There are countless proposed solutions to solve the shortcomings of both approaches. Hybrid systems combine CF and content-based methods to use either of them when most suitable.

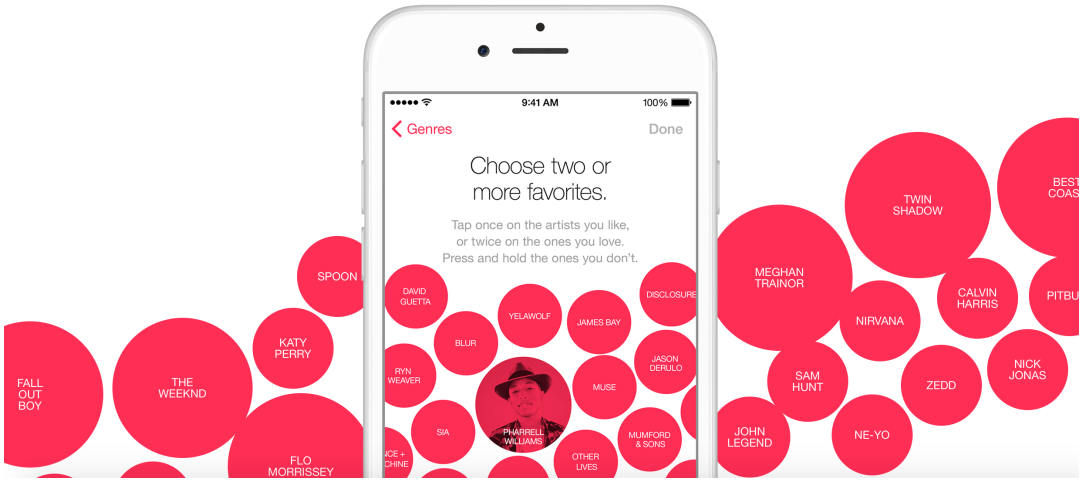

FIGURE 2.4: Favorite artists prompt for new users in Apple Music

2.2 Image Features Extraction

An embedding is a low-dimensional representation of a high-dimensional vector. Processing similarities between two images by using their raw pixel intensities is computationally intensive and thus highly ineffective. A few simple methods will be demonstrated in this paper to explain how feature extraction from images works.

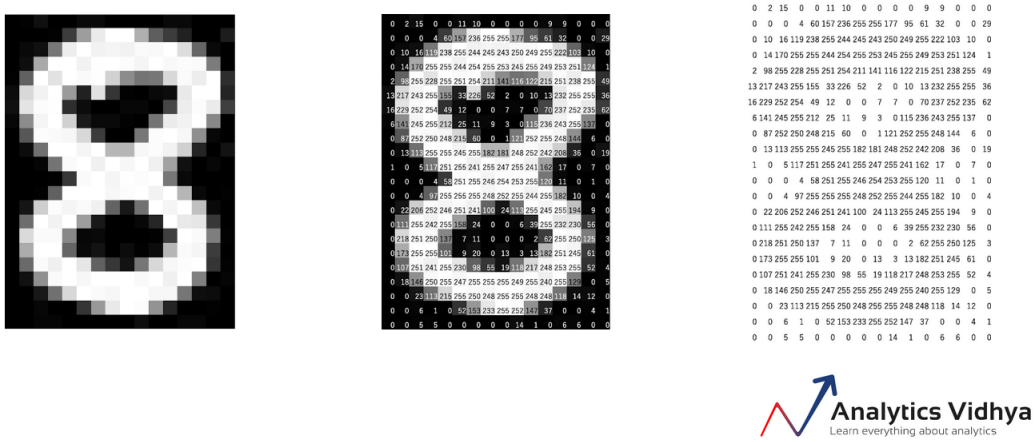

FIGURE 2.5: Image pixel intensity denoted in numbers

The dimensions of the image in 2.5 are 22 x 16 pixels. Here, pixel intensity values range between 0 (meaning black) and 255 (meaning white).

In the case of RGB or colored images, the same concept applies. However, the number of matrices is increased to 3, each representing red, green, and blue channels.

Going back to the grayscale image, the simplest way to create a feature vector would be to arrange all pixels values sequentially, thus creating a 1D array of length 352 or a vector with 352 features.

If the image were RGB, the number of features using the previously mentioned technique would be multiplied by the number of channels in the photo, in this case, 3, resulting in an array of length 1056. Again, the simplest solution to this rapid

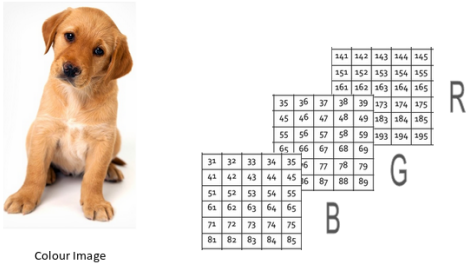

FIGURE 2.6: Using matrices to represent an RGB image

growth in size would be using only one matrix, with its values being a mean value of pixels in every channel, as shown in 2.7 [Singh, 2020].

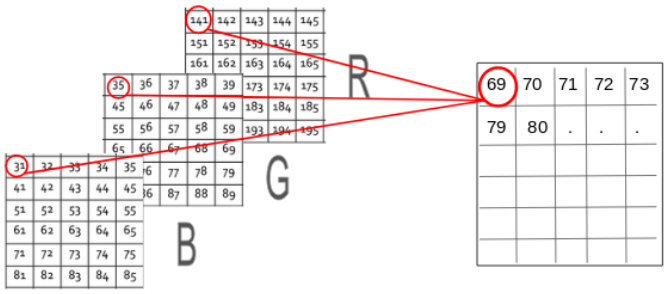

FIGURE 2.7: Translating 3 matrices with pixel values into 1

A more sophisticated technique of feature extraction is edge detection. Edge detection can be achieved by highlighting a pixel in a matrix and calculating the differences between pixels surrounding it. Various operators exist to compute an approximation of the gradient image. For instance, the Prewitt operator uses two 3x3 kernels for horizontal and vertical derivative approximations (see 2.8).

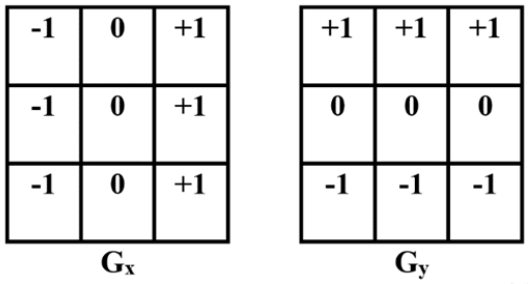

FIGURE 2.8: Prewitt horizontal and vertical kernels

The figure below demonstrates the output of combining vertical and horizontal edge detection masks using the Prewitt operator.

Other traditional Computer Vision techniques include, but not limited to:

- Harris Corner Detection
- Shi-Tomasi Corner Detector

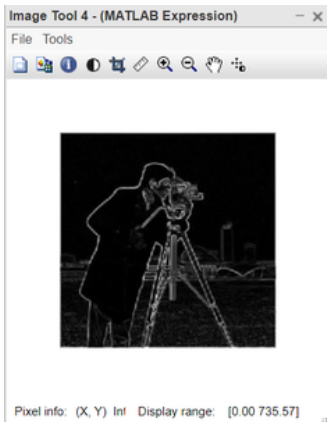

FIGURE 2.9: Image output after applying Prewitt operator

- Scale-Invariant Feature Transform
- Speeded-Up Robust Features
- Features from Accelerated Segment Test

2.3 Deep Learning for Features Extraction

Traditional approaches, however, can be replaced by Deep learning since it performs much better on CV tasks. Convolutional Neural Networks have been proved to provide significantly better efficiency and higher accuracy and demonstrate solid ability in learning context-specific features. Size, lighting, angle, and deformations, for example, are all part of an image context.

CNN allows extracting a set of features by detecting them in different image patches. The first layers of a CNN may extract low-level features like edges, which are then used to find mid-level features like shapes, leading to extracting high-level features, objects, for example. Kernels are used to detect features in an image patch (like the Prewitt kernel mentioned above for edge detection). Manual feature detection requires domain knowledge and handcrafted kernels, which is inefficient from a time perspective. CNNs are the solution to this problem since they learn important features from an image and apply necessary kernels (filters) to detect these features automatically [Bezdan and Bacanin, 2019].

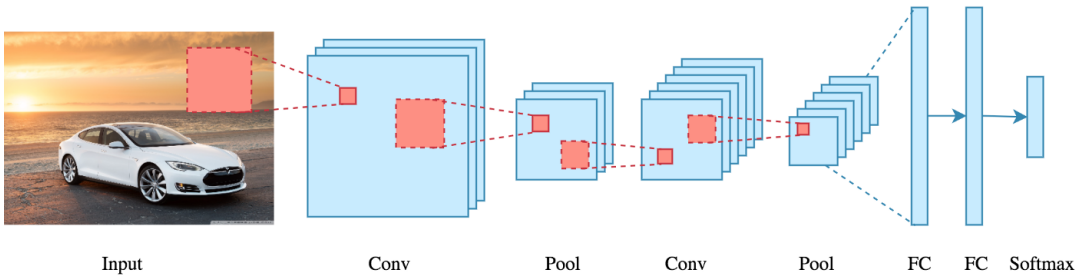

FIGURE 2.10: General architecture of a CNN

As shown in 2.10, an average CNN model consists of convolutional layers, pooling layers, fully connected layers at the end, and an activation function (in this case, softmax for multiclass classification).

Convolution layers use learnable filters to convolve on the input image and output activation (or feature) maps.

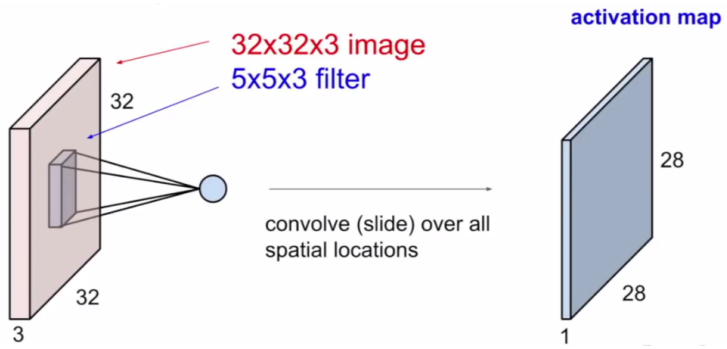

FIGURE 2.11: Convolution using a 5x5x3 filter

Pooling layers are used to perform pooling to reduce dimensionality and computational intensity and prevent overfitting. This is done by downsampling obtained feature maps by sliding pooling windows over patches of each feature map. Pooling operations include max pooling, average pooling, and global pooling [Dertat, 2017].

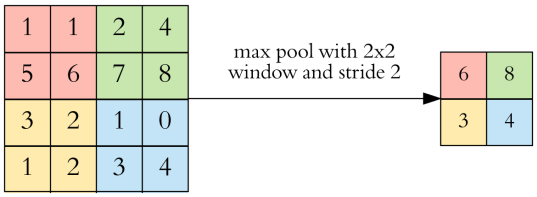

FIGURE 2.12: Max pooling with a 2x2 pooling windows

After these layers, the output is then flattened and passed through fully connected layers that act as a classifier for detected features.

2.4 Deep learning degradation problem

In 2012, a CNN-based architecture AlexNet, which contained eight neural network layers, further subdivided into five convolution layers and three fully connected layers, was the first to show superior results over traditional feature learning by winning the ImageNet competition [Krizhevsky, Sutskever, and Hinton, 2012]. The architecture of this model became a foundation for subsequent traditional CNNs and created a notion that using more layers leads to better results and a lower error rate. This belief, however, was disproved by the following experiment.

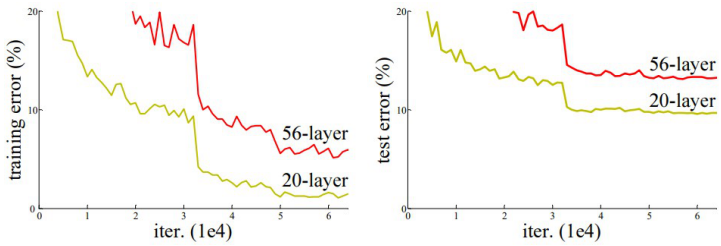

FIGURE 2.13: 20-layer and 56-layer architectures trained on CIFAR-10

As shown in figure 2.13, the 56-layer architecture gives more error rate than its smaller counterpart. It is not a case of overfitting since it tends to occur when training errors are significantly lower than test errors. Thus, the authors concluded that the failure could be attributed to a vanishing gradient problem. [He et al., 2015], [Ebrahimi and Abadi, 2018].

When a network generates output, it uses backpropagation to minimize the loss function. This involves calculating the gradient of the loss function with respect to each weight. With the chain rule, the gradient of the loss function can be represented as a product of gradients of all activation functions with respect to each of their weights.

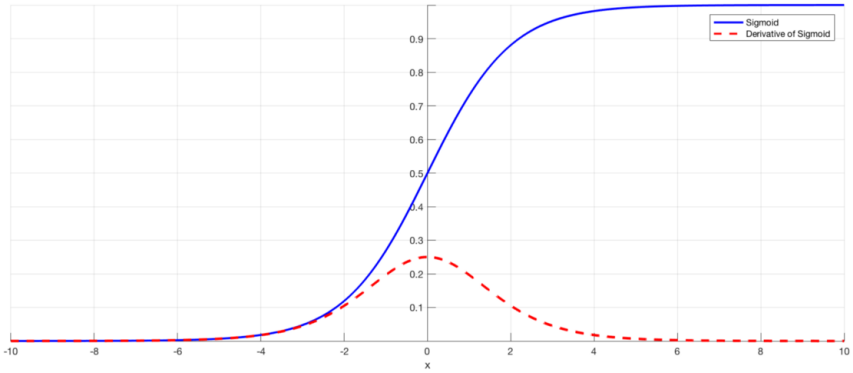

FIGURE 2.14: The sigmoid function and its derivative

As shown in 2.14, the derivative of a sigmoid can reach a maximum value of 0.25. The bigger the input, the smaller the derivative of this function is. Each additional layer in a network means an additional multiplication operation on a small derivative, which leads to the gradient decreasing exponentially [Wang, 2019].

2.5 ResNet as a solution to the degradation problem

Suppose we have a neural network with n layers that give a training error x. Let us consider one more neural network, a deeper one, with $m > n$ layers. Naturally, we expect the deeper network to perform better or at least as good as its shallower counterpart. Since the first n layers in the deeper network will produce the same results, the remaining $m - n$ layers will either learn a more complex representation or act as an identity function to carry the output to the m_{th} layer.

Thus, we can conclude that the deeper network should give a training error y that is no greater than x. Empirically, this does not happen, which proves that the remaining layers are unable to learn the identity mapping. In residual networks, instead of hoping that these layers fit an underlying mapping, we explicitly let them fit a residual mapping. If we denote the desired mapping as $H(x)$ and let the layers fit mapping $F(x) = H(x) - x$, the original mapping is recast to $F(x) + x$ [He et al., 2015].

The residual block, shown in 2.15, features "shortcut connections," which are essentially identity mappings that allow our input to skip one or more layers. With these shortcut (or skip) connections, we are able to formulate $F(x) + x$. Another advantage of these skip connections is that they add no computational complexity and allow for training by SGD with backpropagation. To compare the difference, let us take a "plain" network (in other words, a network without residual blocks). In

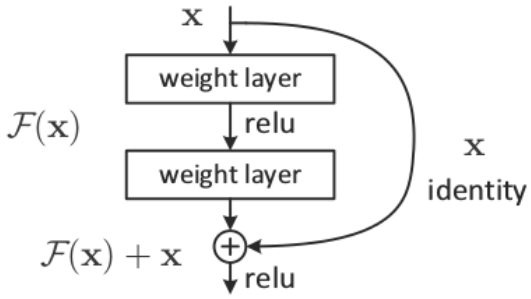

FIGURE 2.15: Residual block

the case of a plain network, the problem of learning an identity function would be solving the equation $H(x) = F(x)$. On the other hand, to solve $H(x) = F(x) + x$, the network has to make $F(x) = 0$, which is an easier task to accomplish [He et al., 2015].

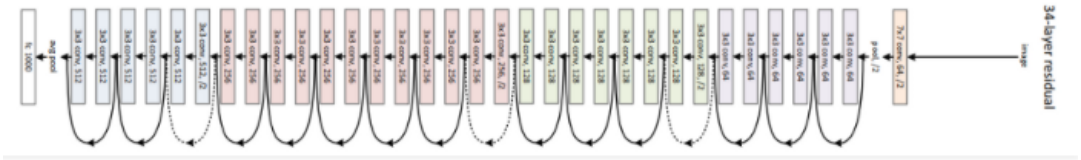

FIGURE 2.16: plain network with shortcut connections

Figure 2.16 shows a plain network with 34 parameter layers is shown with shortcut connections inserted into every few layers. A building block $y = F(x, W_i) + x$. can only be used if the input and the output are of the same dimensions. To achieve this, either the skip connection is padded with extra zero entries to increase its dimensions, or a linear projection W_s is used on the shortcut connection (the projection is made by adding $1x1$ convolutional layers to the input.

2.6 Vision Transformers

CNNs are known to have been dominating the field of CV in image recognition tasks. However, recently, a new alternative to CNN has arisen. To understand how ViT work, let us first take a look at transformers, which served as a foundation for this CNN alternative.

Initially, in sequence-to-sequence problems, the solutions were based on Recurrent Neural Networks that worked sequentially to preserve the order of the sentence, thus requiring each layer to have access to the previous output. As a result, LSTM computations were performed sequentially, posing serious limitations to the model: with long sentences, if the decoder only accesses the last output, it loses information about the first elements in the sequence. To combat those limitations, the attention mechanism was introduced. Attention allowed to extract a weighted sum of all encoder states, thus extracting information from the whole sequence. This enabled the decoder to focus on the most important element of the input to predict the output. Nevertheless, the mechanism still fell short on large sequences due to processing each element at a time requiring time and adding to overall computational complexity.

Transformers, on the other hand, use a self-attention mechanism to figure out the importance of other words in a sentence relative to the last-mentioned word. The self-attention mechanism is a sequence-to-sequence operation that takes a weighted average over all the input vectors to produce output vectors of the same dimensions. In this mechanism, every input vector has three representations: key, query, and value. Maxime, 2020

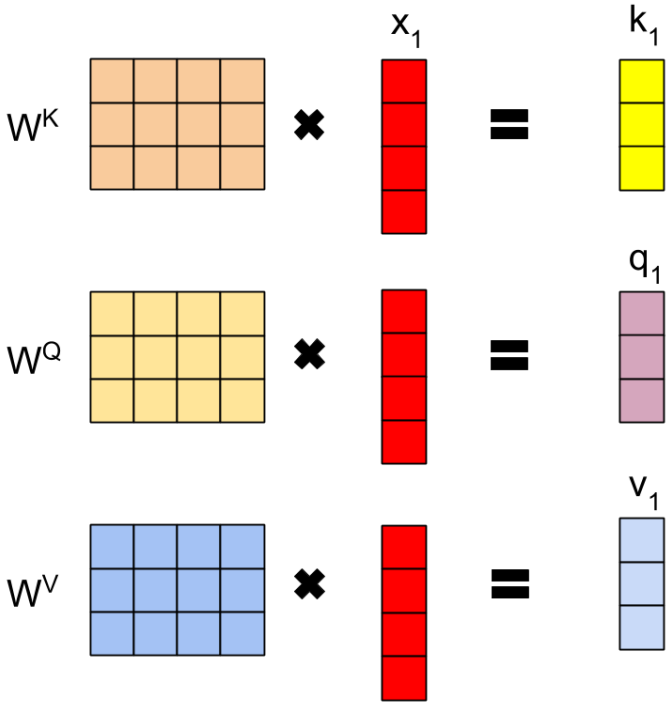

FIGURE 2.17: Deriving key, query and value representations from input vectors

Figure 2.17 shows an example of deriving these representations from an input vector of dimension 4. In this example, the representations all have a dimension of 3. Each of these is a result of a matrix-vector product, as shown above.

To calculate an attention score for a word at a certain position, we calculate the dot product of the selected input query and all keys, including self. A softmax function is then applied to the output to obtain the score. This is described in the paper by the following formula:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{d_k}})V \tag{2.1}$$

Where Q, K, V are the query, key, and value matrices, respectively. d_k represents the dimensions of keys, and we use its square root as a scaling factor for our softmax function [Vaswani et al., 2017].

For multi-head attention, the individual attention scores are concatenated and multiplied by an additional weights matrix to match the output dimension for the feed-forward layer 2.18.

The self-attention mechanism is permutation invariant. This implies that regardless of the words' positions in the same sentence, the outcome stays unaltered. To combat this problem, we create a representation of the position of the word and add it to the word embedding. This process is called positional encoding. In the paper, a

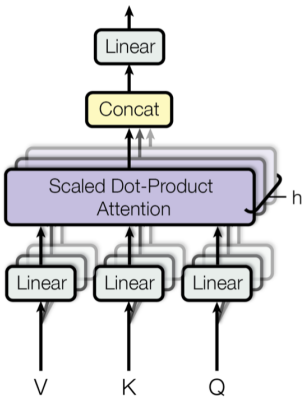

FIGURE 2.18: Multi-head attention

sinusoidal function is used.

$$PE(pos, 2i) = \sin(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.2}$$

$$PE(pos, 2i+1) = \cos(\frac{pos}{10000^{2i/d_{model}}}) \tag{2.3}$$

Where d_{model} is the dimensionality of the embedding vector. Thus, for even positions we use the cos function, and sin for even positions.

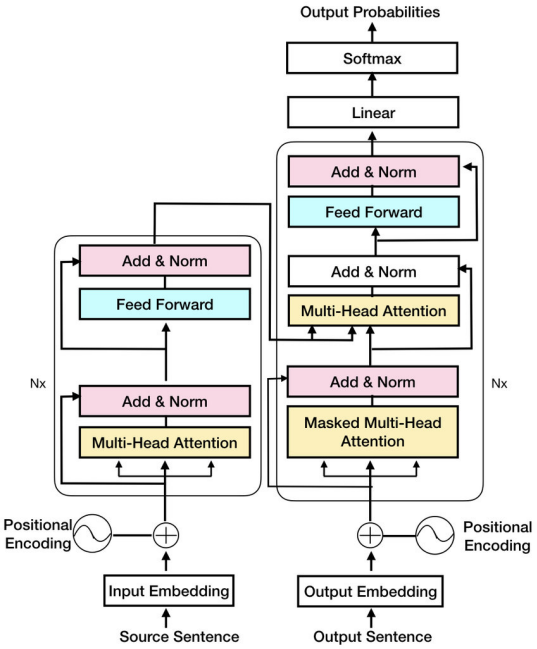

FIGURE 2.19: Transformer encoder and decoder

Let us now walk through the encoder components. The encoder incorporates:

1. Word tokenization and embeddings
2. Adding positional encodings to the embeddings
3. Passing the resulting vectors to the first encoder block

Each encoder block consists of N layers, which in turn comprise the following sub-layers:

- A multi-head attention mechanism
- A normalization layer
- A linear layer
- Second normalization layer

There are two residual connections in a block around each of the two sub-layers. Both the encoder and the decoder share most of the similarities, with a few key differences. As can be observed in 2.19, the first sub-layer is Masked Multi-Head Attention, which allows disregarding unknown outputs by masking the next word embeddings. We do this by setting their value to $-\infty$.

$$MaskedAttention(Q, K, V) = softmax(\frac{QK^T + M}{\sqrt{d_k}})V \tag{2.4}$$

Where M is the matrix of zeros and $-\infty$.

The Multi-Head Attention (also Encoder-Decoder attention) takes the Key and Value matrices from the output of the encoder and the Query matrix from the Masked Multi-Head Attention output. The outputs of N decoders are then passed through the linear layer resulting in logits (i.e., a vector of scores). Softmax is used to turn those scores into probabilities, and the word corresponding to the highest probability is chosen as the output.

Now, we can apply the given knowledge in CV to construct a vision transformer.

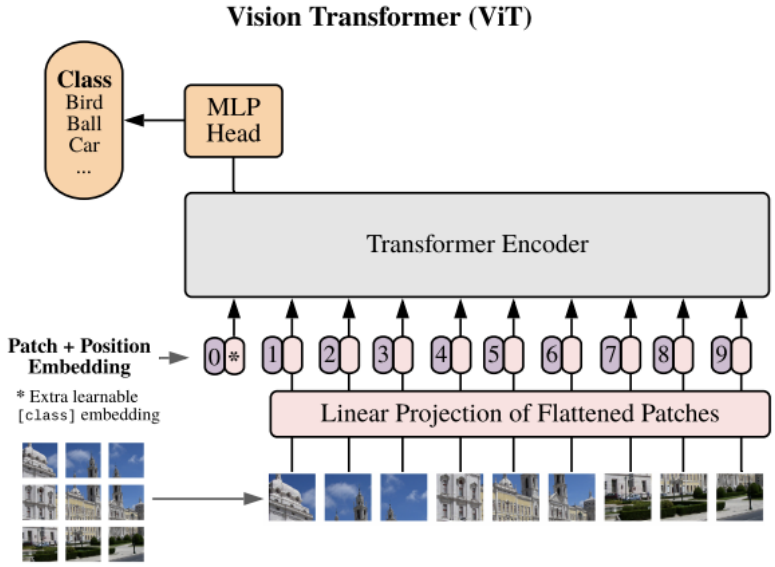

FIGURE 2.20: Vision transformer model overview

The model follows the original Transformer as closely as possible [Dosovitskiy et al., 2020]. Thus, following the same principles as with sentences, we split the input image into fixed-size patches (equivalent to tokens) and flatten those patches. A learnable embedding is added to the sequence of patches. Position embeddings

are then added to the patch embeddings to maintain positional information. Finally, those are fed to the transformer encoder [Bhojanapalli et al., 2021], [Sarkar, 2021].

Main differences between a ViT and a CNN:

- Transformers lack some of the inductive biases present in CNNs, such as translational equivariance and locality, and have to learn these properties from the given data. Translational equivariance means that the activation in a feature layer of a CNN will translate accordingly to the image translations. In other words, CNN is able to recognize an object even when its position has changed.
- Unlike a CNN, a transformer is permutation invariant by design.
- Due to the lack of the aforementioned inductive biases, transformers require huge training datasets to provide higher accuracy results than ResNet, for instance. (huge implying datasets that comprise at least 14M images)
- Transformers provide higher computational efficiency and scalability. That is, again, if we train it on large datasets

Chapter 3

Solution Overview

3.1 Image embeddings

Several methods of producing image embeddings exist, all varying in difficulty. Let's go into more specifics about several of these.

1. If we print out children of a ResNet18 model in Pytorch (this specific model was chosen for brevity), we will see the following:

```
Children Counter:  0  Layer Name:  conv1
Children Counter:  1  Layer Name:  bn1
Children Counter:  2  Layer Name:  relu
Children Counter:  3  Layer Name:  maxpool
Children Counter:  4  Layer Name:  layer1
Children Counter:  5  Layer Name:  layer2
Children Counter:  6  Layer Name:  layer3
Children Counter:  7  Layer Name:  layer4
Children Counter:  8  Layer Name:  avgpool
Children Counter:  9  Layer Name:  fc
```

   To obtain the output of our last layer (namely layer4), we take the modules in our model and discard the average pooling and fully connected layers that come after layer4. Using the remaining modules, we can replace our pretrained model with a new Sequential model. The drawback here is, of course, the creation of two virtually identical models, which take up twice as much memory until the initial one is freed [Manna, 2021b].

2. Second way of doing this involves just one instance of a model. This is done by first creating a subclass of a ResNet class in Pytorch and assigning only the layers we need to it. We then obtain the output from layer4 by overriding the _forward_impl method of the ResNet class to return the output. This method is easier on the memory but requires creating an instance from our subclass and loading the pretrained weights [Manna, 2021a].

3. Attaching hooks. Pytorch provides one more, arguably, the most convenient method for extracting activations. The method register_forward_hook, as one may have guessed, enables registering hooks on desired layers. When the forward() method is triggered, the modules inputs and outputs get are passed to the hook. This method was chosen for the implementation due to its intuitive design.

3.2 Dataset

Yelp is a crowd-sourced local business review directory. Yelp was a compelling choice for producing a dataset for this task since it features its own API called Yelp Fusion. Using an API would allow for great customizability in choosing what data the dataset is comprised of.

Yelp fusion provides a few useful endpoints that could contribute to our dataset:

- Business search, featuring querying by search terms (e.g., "food" or "restaurants"). This endpoint provides some useful information that we could use, for example, the business' id, location, rating, categories, and review count.
- Business details, providing much more information about a certain business than the aforementioned endpoint, namely photos of the business.
- Reviews endpoint returns reviews' id, text, rating, timestamp, user's id, and user's name for a given business id. The timestamp is crucial for creating a time-based sequence of reviews from one user.

Nevertheless, Yelp Fusion suffers from a few major drawbacks.

- By default, a client is limited to only 5000 API calls per 24 hours. This can be used up pretty quickly if several endpoints are used in conjunction to produce the final dataframe. Of course, one can register a few API keys on different accounts, but this process makes gathering data so much more inefficient.
- The API features no endpoints to retrieve reviews by a specific user id. Hence, forming user-based sequences can be a little tricky.
- For each business, only up to 3 reviews can be returned. This makes the task of forming a complete dataset based on just one city seem highly impractical.
- Every business query by term returns a maximum of 50 results, allowing up to 1000 by setting the offset value in the request parameters. This means that for every 1000 businesses per term, 20 API calls have to be executed.

Having said that, this is still one of the best options for obtaining restaurant-specific information with photos, so an attempt was made to make use of the API. First, three scripts were created to obtain and process the necessary data. To extract the maximum amount of information about businesses in one city, the script used 3 of the available sorting methods (best match, rating, review count) and a dozen terms related to our topic. For each 50 business ids obtained from this search, 50 more requests performed a search for business details (i.e., photos) by business id, and the same amount for reviews for a given business. Seconds script grouped all obtained reviews by $user_{id}$, and the third one merged all these reviews into one file. The duplicate reviews, obtained from performing numerous searches with large intersections, were then dropped when operating with the DataFrame object. After some time of experimenting with the API, it was concluded that pulling a sufficient amount of reviews is possible, but forming a fair amount of considerably long sequences (at least five reviews per user) is not.

Fortunately, Yelp provides its own dataset to work with. The numbers specified in its description are as follows:

- 6,990,280 reviews

- 150,346 businesses
- 200,100 pictures

This may seem a lot, but if only one city is accounted for, the amount of data might not be perfect for achieving the highest accuracy results possible. Still, it is definitely enough to experiment with the model.

This dataset provides four files (business.json, review.json, user.json, photo.json) that may be of use for our case. Unfortunately, the user file contains little information that can be applied for context learning (i.e. no person-related features like age or sex), so this file was omitted in the resulting dataframe.
After converting these files to .csv format and stripping all the unnecessary data, here are the samples of two resulting dataframes for photos and reviews, respectively.

	user_id	business_id	rating	timestamp
0	mh_-eMZ6K5RLWhZyISBhwA	XQfwVwDr-v0ZS3_CbbE5Xw	3	2018-07-07 22:09:11
1	OyoGAe7OKpv6SyGZT5g77Q	7ATYjTIgM3jUlt4UM3IypQ	5	2012-01-03 15:28:18
2	8g_iMtfSiwikVnbP2etR0A	YjUWPpI6HXG530lwP-fb2A	3	2014-02-05 20:30:30
3	_7bHUi9Uuf5__HHc_Q8guQ	kxX2SOes4o-D3ZQBkiMRfA	5	2015-01-04 00:01:03
4	bcjbaE6dDog4jkNY91ncLQ	e4Vwtrqf-wpJfwesgvdgxQ	4	2017-01-14 20:54:15
...	...	...	...	...
6990275	qskILQ3k0I_qcCMI-k6_QQ	jals67o91gcrD4DC81Vk6w	5	2014-12-17 21:45:20
6990276	Zo0th2m8Ez4gLSbHftiQvg	2vLksaMmSEcGbjI5gywpZA	5	2021-03-31 16:55:10
6990277	mm6E4FbCMwJmb7kPDZ5v2Q	R1khUUxidqfaJmcpmGd4aw	4	2019-12-30 03:56:30
6990278	YwAMC-jvZ1fvEUum6QkEkw	Rr9kKArrMhSLVE9a53q-aA	5	2022-01-19 18:59:27
6990279	6JehEvdoCvZPJ_XIxnzIIw	VAeEXLbEcI9Emt9KGYq9aA	3	2018-01-02 22:50:47

FIGURE 3.1: Sample of the reviews dataframe

	photo_ids	business_id
0	N3tTxsgpFH91wzo4gZm19A	0
1	IbkOXfgXq12U5Pye_ySGFw	1
2	oewWFZoEf7AVYyWJ614amw	2
3	6Tm1_c_r3-TtrFpEWI6RTg	3
5	zh0zAoJYe_LGrZmatx5AFA	5
...	...	...
5841	ktCxuBAe9Pa65_m4ZVjVaQ	5841
5844	Pgs8tAT_gqTbfXJPzD-glw	5844
5845	mCF-hPRPx5YwqVx738QGAw	5845
5849	gilz_mJqz8t0tGr1XCJuXw	5849
5851	igkn2yYqmoFf7sM-to-byA	5851

FIGURE 3.2: Sample of the photos dataframe

Some columns have been dropped in these samples but contain extra information like image labels (inside, outside, drink, food, menu) and business categories that can be used as extra features to further experiment with training our model.

3.3 Behavior Sequence Transformer

The base architecture for the model in this paper will be the Behavior Sequence Transformer developed by Alibaba Search and Recommendation Group.
The paper proposes that we use the Transformer model to "capture the sequential

signals underlying users' behavior" for recommendations. The problem this model is trying to solve is that the standard embedding and MLP paradigm, despite being successful, fails to take into account a very important factor of sequences, i.e., users' clicked items in an order.

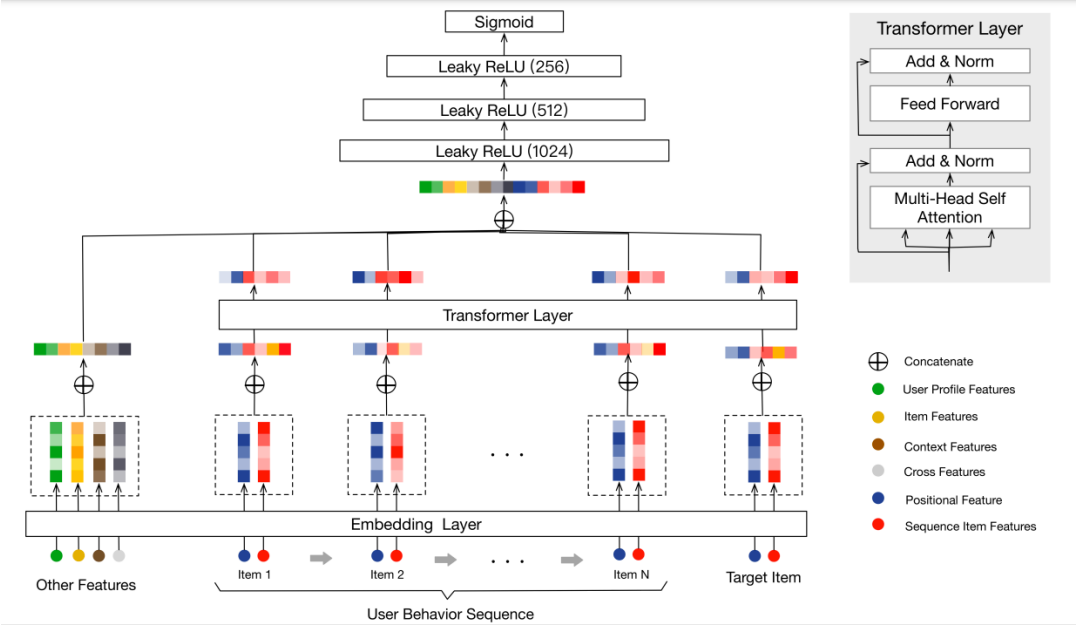

FIGURE 3.3: Behavior Sequence Transformer architecture

The embedding layer embeds all input features into low-dimensional vectors of fixed size. Other Features denotes features related to the user (e.g., gender, age, city), item clicked (category, shop, tag), contextual features exclusive to the website, and cross features (e.g., age * item). These features are concatenated and embedded into a low-dimensional vector, as can be seen in 3.3.
For the rest, each item from a sequence is embedded, as well as a target item. An item is represented by "Sequence Item Features" and "Positional Features," where the first includes the item id and its category. In the paper, position values are computed as $pos(v_i) = t(v_t) - t(v_i)$, where $t(v_t$ is the recommending time of item v_i, and $t(v_i$ is the time when user clicks on the item. This method claims to outperform the sinusoidal and cosinusoidal functions used in the original Transformer paper. The scaled dot-product attention is described as

$$S = MH(E) = Concat(head_1, head_2, \ldots, head_h)$$

$$W^H = Attention(EW^Q, EW^K, EW^V),$$

where W^Q, W^K, W^V are the projection matrices, Q, K, V are the Query, Key and Value matrices respectively, E is the embedding matrices of all items, and h is the number of heads. To avoid overfitting, dropout and LeakyRELU are used both in self-attention and Point-wise Feed Forward Network. Chen et al., 2019

To summarize, the key modifications made to the original Transformer in this model are:

- Other features embeddings are added and concatenated to the Transformer Layer output

- Item features are used for the Embedding Layer
- The sigmoid function is used to generate the final output (for the recommendation)

Chapter 4

Implementation and experiments

4.1 Model Implementation

The implementation is based on the Behavior Sequence Transformer model. Since user data contains no personal information on the user, only the user's id is used in place of Other Features of the original model, and thus, no cross features are implemented.

In the original paper, the position is calculated using $pos(v_i) = t(v_t) - t(v_i)$, where $t(v_t)$ is the recommending time of the item and $t(v_i)$ is the time when user click the item v_i. In our scenario, we use the timestamps of the reviews to sort them in order of posting the review and represent positions of items as their respective positions in the created sequence. In other words, positions in a sequence of length 8 are represented with an array [0, 1, 2, 3, 4, 5, 6, 7].

For items, we use extracted image features. For this purpose, we created a class to pass normalized images from the dataset to a pretrained ResNet model and extract features from the avgpool layer. The best performance was achieved when converting the images to tensors and saving them to the dataframe, rather than performing feature extraction during the training phase.

As this is not a sequence to sequence problem, only the Encoder is used in the model. Fortunately, the Pytorch machine learning framework already provides a built-in $TransformerEncoderLayer$ class that is made up of self-attention and feed-forward network following the "Attention is all you need" paper.

In the original Transformer paper, the Adam optimizer is used for the training phase. We use AdamW optimizer for this model, which is essentially the same optimizer but includes decoupling the weight decay from the optimization step.

The model is a linear regression for rating predictions in the range of 1 to 5. For training, the loss function is Mean Squared Error (MSE) and is computed as

$$\frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Where n = number of items, Y = observed values, $\hat{Y}$ = predicted value. The square part of the function allows it to put larger weights on errors, preventing outlier predictions.

4.2 Metrics

Metrics are divided into two sections. In training, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) are used.

$$\frac{1}{n}\sum_{i=1}^{n}|Y_i - \hat{Y}_i|$$

The following metrics are present in the test: precision, recall, and F1-score. Additionally, a confusion matrix is plotted for a better visual interpretation of the results.

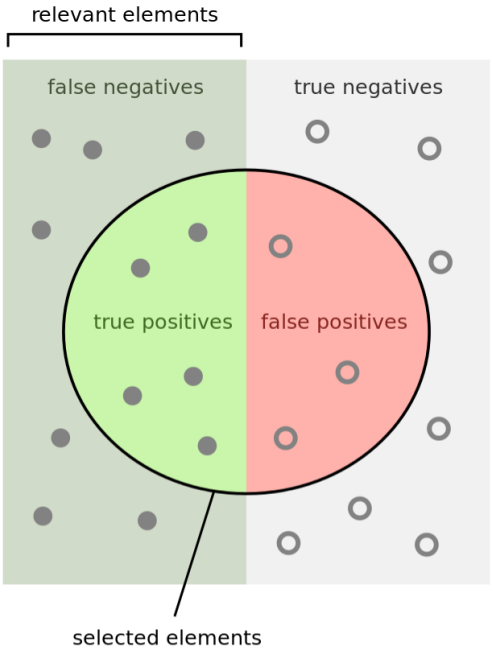

FIGURE 4.1: Precision and Recall

Figure 4.1 is a good explanation of the underlying principle of recall and precision metrics. Precision is defined as

$$\frac{TP}{TP + FP} \tag{4.1}$$

With TP = True Positive, FN = False Positive respectively. Precision measures the fraction of relevant instances out of all retrieved instances. Recall is calculated using

$$\frac{TP}{TP + FN} \tag{4.2}$$

Where FN = False Negative. Recall determines the accuracy of the model in terms of correctly identifying True Positives.
F1-score is a harmonic mean of precision and recall and is calculated as follows:

$$F_1 = 2\frac{precision * recall}{precision + recall} \tag{4.3}$$

To simplify, F1-score combines recall and precision to rate the overall accuracy of our model.

4.3 Experimenting with the model

Following options were considered during the experiments:

- Choosing batch sizes between 32 and 256 with increments of power of 2
- Limiting image processing exclusively to images of same label (inside, outside, drink, food, menu)
- Choosing between AdamW and AdaGrad optimizers
- Adding ReduceLROnPlateau learning rate scheduler on top of the existing optimizer
- Varying learning rate from 0.00005 to 0.01
- Using Pytorch Lightning automatic learning rate finder utility
- Processing images inside batches vs. preprocessing images to dataset directly
- Applying normalization before and after in the Encoder layer
- Using positional encodings with a sinusoidal function instead of pytorch embeddings

Using Pytorch Lightning tuner to automatically find the optimal learning rate proved partially useful. The initial learning rate suggested by the system (by running warmup steps in the range 100-200) closely corresponded to manual findings (that is, the learning rate of values from 0.005 to 0.0013 for batches of sizes 64 and 128). Experimenting with the learning rate was a necessity due to the tuner malfunctioning and causing the error rate to grow exponentially (this feature is, indeed, described as "bleeding edge").

AdamW showed slightly better performance than AdaGrad, and thus was used in its favor with parameters specified following "Attention is all you need" (with parameters $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-8}$). Using a Resnet-34 pretrained model instead of ResNet-18 demonstrated worse accuracy and needed additional fine-tuning.

Encoding images to vectors prior to feeding them to the transformer vastly improved training time. Image vectors were added to the training dataframe to be directly accessed by our model and ignore unnecessary loading times.

Over several dozens of runs showed that the minimum loss value that could possibly be achieved with a given dataset size was 1.1-1.3 (MSE). Due to the absolute value of error exceeding 1, the testing phase was converted to a binary classification problem rather than multiclass since discretizing values would cause a serious fraction of predictions to be off by a unit.

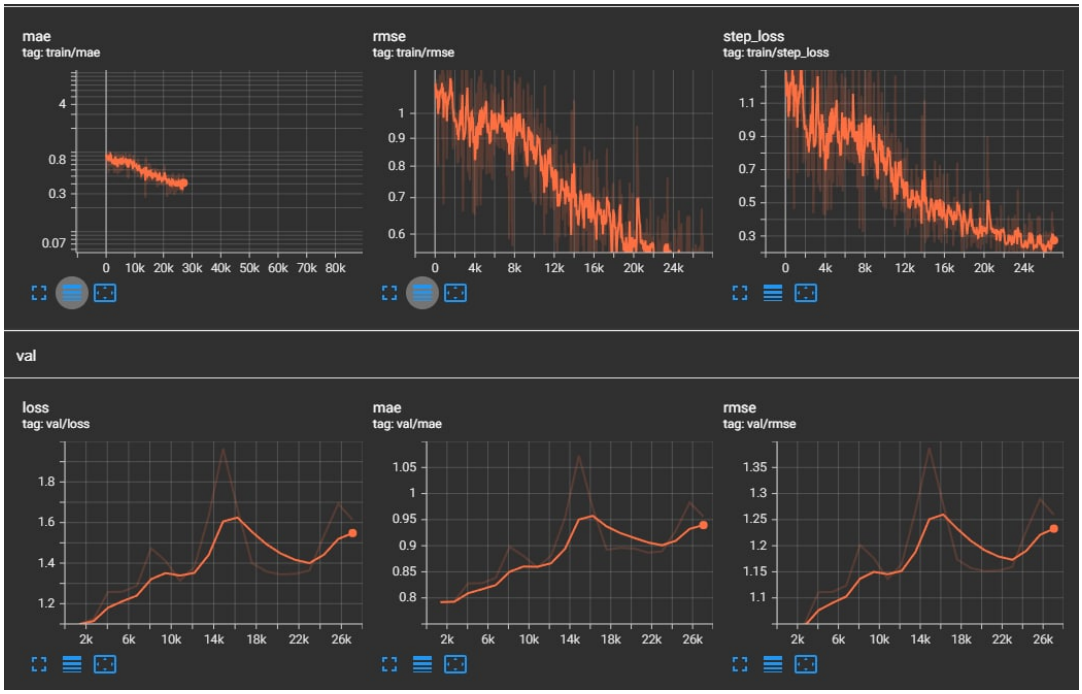

FIGURE 4.2: Tensorboard metrics for training

Initial training was performed exclusively on one city from the dataset, comprising 3299 images and 92399 sequences. Each modification was tested on batches of sizes 32, 64, and 128 (with some exceptions, including 16 for tuning learning rate). Other experiments on achieving the lowest error rate were conducted on a bigger training dataset of 14075 images and 379312 sequences. Some runs included the entire dataset but were insufficient to achieve a higher convergence rate.

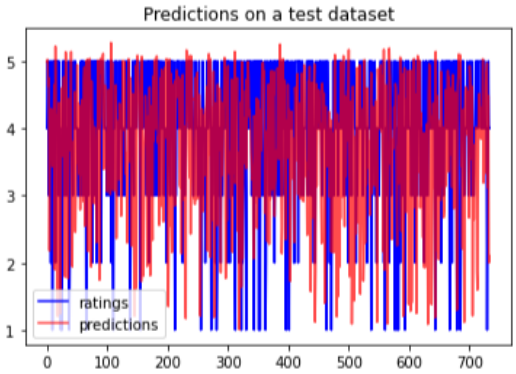

FIGURE 4.3: Visual overlay representation of predictions over target

Chapter 5

Conclusions

5.1 Summary

In this work, we checked the possibility to use a recommendation system based on deep learning approaches for trip planning using visual information from places photos. We tested the architecture of attention-based NN as a baseline and adapted it for working with images input. The model demonstrated relatively poor performance and the reason of that can be a big variation of input data. We proved that guessing user tastes is very complex task and can be depended not only on entire place's features but also on other features that we cannot easily include into the model.

5.2 Future work

Our goal in the future would be to increase a set of external factors that can help to extend environment vector and most probably increase a performance of NN.

Bibliography

Bezdan, Timea and Nebojsa Bacanin (Jan. 2019). "Convolutional Neural Network Layers and Architectures". In: pp. 445–451. DOI: 10.15308/Sinteza-2019-445-451.

Bhojanapalli, Srinadh et al. (2021). "Understanding Robustness of Transformers for Image Classification". In: *CoRR* abs/2103.14586. arXiv: 2103.14586. URL: https://arxiv.org/abs/2103.14586.

Chen, Qiwei et al. (2019). "Behavior Sequence Transformer for E-commerce Recommendation in Alibaba". In: *CoRR* abs/1905.06874. arXiv: 1905.06874. URL: http://arxiv.org/abs/1905.06874.

Dertat, Arden (2017). *Applied deep learning - part 4: Convolutional Neural Networks*. URL: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

Dosovitskiy, Alexey et al. (2020). "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale". In: *CoRR* abs/2010.11929. arXiv: 2010.11929. URL: https://arxiv.org/abs/2010.11929.

Ebrahimi, Mohammad Sadegh and Hossein Karkeh Abadi (2018). "Study of Residual Networks for Image Recognition". In: *CoRR* abs/1805.00325. arXiv: 1805.00325. URL: http://arxiv.org/abs/1805.00325.

He, Kaiming et al. (2015). "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E Hinton (2012). "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira et al. Vol. 25. Curran Associates, Inc. URL: https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

Manna, Siladittya (2021a). *Extracting features from an intermediate layer of a pretrained resnet model in PyTorch (easy way)*. URL: https://medium.com/the-owl/extracting-features-from-an-intermediate-layer-of-a-pretrained-model-in-pytorch-easy-way-62631c7fa8f6.

— (2021b). *Extracting features from an intermediate layer of a pretrained resnet model in pytorch (hard way)*. URL: https://medium.com/the-owl/extracting-features-from-an-intermediate-layer-of-a-pretrained-model-in-pytorch-c00589bda32b.

Maxime (2020). *What is a Transformer?* URL: https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04.

Research, Emergen (2021). *Artificial Intelligence (AI) in food and beverage market by end-use (hotel and restaurant, Food Processing Industry, Beverage Industry), by application (consumer engagement, Quality Control, and safety compliance), and by region, forecast to 2028*. https://www.emergenresearch.com/industry-report/artificial-intelligence-in-food-and-beverage-market.

Ricci, Francesco, Lior Rokach, and Bracha Shapira (2011). "Introduction to Recommender Systems Handbook". In: *Recommender Systems Handbook*. Ed. by Francesco Ricci et al. Boston, MA: Springer US, pp. 1–35. ISBN: 978-0-387-85820-3. DOI: 10.

1007/978-0-387-85820-3_1. URL: https://doi.org/10.1007/978-0-387-85820-3_1.

Rocca, Baptiste (2019). *Introduction to recommender systems*. URL: https://towardsdatascience.com/introduction-to-recommender-systems-6c66cf15ada.

Sarkar, Arjun (2021). *Are transformers better than CNN's at image recognition?* URL: https://towardsdatascience.com/are-transformers-better-than-cnns-at-image-recognition-ced60ccc7c8.

Sarwar, Badrul et al. (2001). "Item-Based Collaborative Filtering Recommendation Algorithms". In: *Proceedings of the 10th International Conference on World Wide Web*. WWW '01. Hong Kong, Hong Kong: Association for Computing Machinery, 285–295. ISBN: 1581133480. DOI: 10.1145/371920.372071. URL: https://doi.org/10.1145/371920.372071.

Singh, Aishwarya (2020). *Image feature extraction: Feature extraction using python*. URL: https://www.analyticsvidhya.com/blog/2019/08/3-techniques-extract-features-from-image-data-machine-learning-python/.

Vaswani, Ashish et al. (2017). "Attention Is All You Need". In: *CoRR* abs/1706.03762. arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

Wang, Chi-Feng (2019). *The vanishing gradient problem*. URL: https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484.