Deep Sentiment Analysis on Tumblr

Anthony Hu

A dissertation submitted in partial fulfilment
of the requirements for the degree of
Master of Science in Applied Statistics

Department of Statistics
University of Oxford
Oxford, United Kingdom

September 2017

Declaration

The work in this thesis is based on research carried out at the Department of Statistics, University of Oxford. No part of this thesis has been submitted elsewhere for any other degree or qualification and it is all my own work unless referenced to the contrary in the text.

Acknowledgements

I would like to thank my supervisor Seth Flaxman who always had great insights and ideas. I would also like to thank my parents for giving me the opportunity to study here in Oxford, and for their unfaltering support.

Deep Sentiment Analysis on Tumblr

Anthony Hu

Submitted for the degree of Master of Science in Applied Statistics
September 2017

Abstract

This thesis proposes a novel approach to sentiment analysis using deep neural networks on both image and text. Deep convolutional layers extract relevant features on Tumblr photos and word embedding summarise the textual information to accurately infer the emotion of the post.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Chapter 2

Tumblr data

2.1 Overview of the data

Tumblr posts were retrieved using the official API, here is an example of a post:

Figure 2.1: An example of a Tumblr post

The tags are valuable as they indicate the user state of mind when writing that post. Queries were made searching for six different emotions appearing in the tags: happy, sad, angry, surprised, scared and disgusted. Each post contains the following information:

1. The text, in the example above: *When dogs are back home!*
2. The picture.
3. The associated emotion: one among the six classes.

The data extraction took several weeks due to the API's limitations: 1,000 requests per hour and 5,000 requests per day, with each request containing 20 posts. The final dataset has about 1 million posts.

2.2 Data preprocessing

In some posts, the tag was also appearing in the text itself, for instance:

"When you're on vacations and there is a rainstorm. #fail #sad".

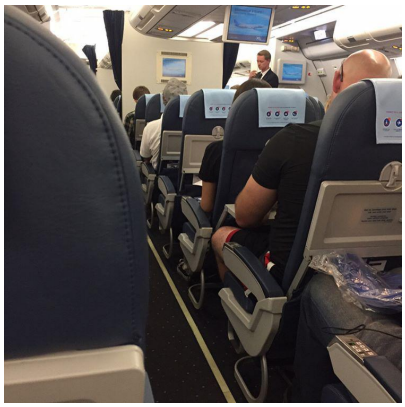
Keeping the *#sad* would bias the learning process and the neural network would simply learn to detect the presence or absence of that tag. To ensure that the network is actually learning something, we removed the hashtags containing the emotion to be predicted.

Also, Tumblr is used worldwide, therefore posts not written in english had to removed from the training data.

Need to talk about preprocessing non-english posts/longest post, smallest post/ + final dataset size?

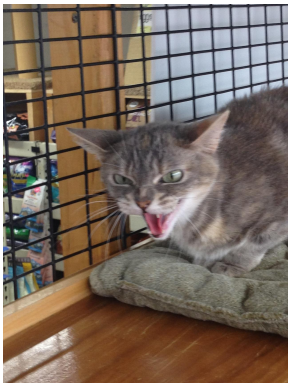
Here are examples of posts with their associated emotions:

(a) **Happy**: "Just relax with this amazing view #bigsur #california #roadtrip #usa #life #fitness (at McWay Falls)"

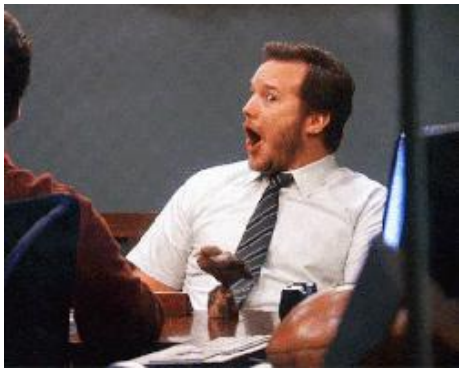


(b) **Scared**: "On a plane guys! We're about to head out into the sky to Paris, France #Paris #trip #kinda #nervous #fun #vacations"

(c) **Sad**: "It's okay to be upset. It's okay to not always be happy. It's okay to cry. Never hide your emotions in fear of upsetting others or of being a bother If you think no one will listen. Then I will."



(d) **Angry**: "Tensions were high this Caturday..."



(e) **Surprised**: "Which Tea? Peppermint tea: What is your favorite gif right now?"

(f) **Disgusted**: "Me when I see a couple expressing their affection in physical ways in public"

Figure 2.2: The 6 emotions in Tumblr posts [1]

Chapter 3

Visual recognition

The pictures are valuable to accurately determine the emotion of the user. For instance, happy photos might contain sunny landscapes and sandy beaches while sad pictures might contain darker colors. To analyse the images, we'll use convolutional neural networks, which achieve state-of-the-art performances in many visual recognition tasks. First we'll explain how they work and then we'll dive into the architecture we've used for deep sentiment analysis.

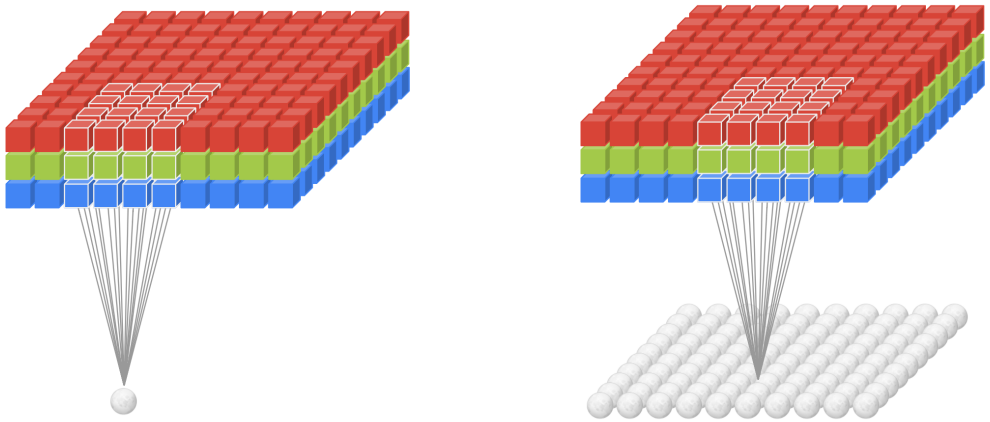
3.1 Convolutional neural networks

A convolutional neural network, often called ConvNet, can be seen as a simulation of the human visual cortex, that is to say an aggregation of plenty of receptive fields. (some illustrations might be helpful here)

3.1.1 Convolutional layer

Take an image of dimension $(h, w, 3)$ with h the height, w the width and 3 representing the number of channels (red, blue, green). If you simply flatten that image and transform it into a vector of size $h \times h \times 3$ and feed that vector to a neural network, you'll get average results as you've thrown away all the spatial information. Convolutions extract that spatial information and work the following way:

- Each convolution is described by a filter F of size $(f, f, 3)$, f usually being equal to 3, 5, or 7.



(a) One operation of convolution

(b) The result of the convolution

Figure 3.1: A convolution, each neuron been a 'receptive field' [2]

- Position the filter on the upper left of the image and element-wise multiply with the filter, then sum those numbers to obtain a 'neuron'.
- Slide across the image, one pixel at a time horizontally and vertically, and repeat the previous operation (see Figure 3.1).

By sliding through the image, you will get a new matrix of dimension $(h_{new}, w_{new}, 1)$, with $h_{new} = h - f + 1$ and $w_{new} = w - f + 1$. However, we usually don't want to reduce the size of our input image that fast, as we want to stack several convolutions. To ensure that the image has the same size after each convolution, zero-padding is used: we add p zeros to the borders of the input image to preserve the spatial size of the input. (illustration needed, before and after zero-padding)With zero-padding, h_{new} becomes: $h_{new} = h + 2p - f + 1$, and we want that h_{new} to be equal to h:

$$h + 2p - f + 1 = h \tag{3.1}$$

Therefore, $p = \frac{f-1}{2}$. Besides, zeros are used instead of any other number because you want the filter to activate on the pixels of the image only, therefore, setting the added border to zeros ensure that the resulting neuron will not be influenced by the border.

A convolution extracts information about the image such as edges or blotches of some color (Figure 3.2). The grayscale and edges filters were hardcoded but

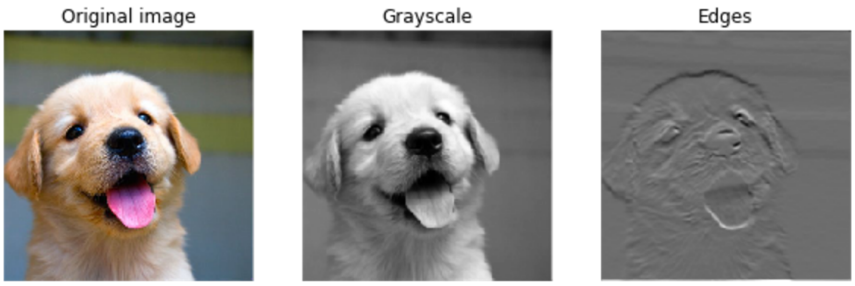


Figure 3.2: Examples of convolution

in a ConvNet setting, the weights of the filter F are learned through optimising a loss function – in our case, a metric measuring how accurate our predictions of the emotions are. The network will learn weights that will detect features that will be most relevant to our specific task. A convolution also has a depth parameter d: simply repeat the operation described above d times with d independent filters of the same size $(f, f, 3)$, to create a new tensor of dimension (h, w, d).

We can then apply convolutions on that new tensor. First layers will detect simple features such as edges or aggregation of colors, and deeper layers might recognise more complex features such as faces or wheels.

3.1.2 ReLU layer

Stacking convolutions is nice, but as it is, we are only creature features that are linearly dependent of the input pixels: we could in fact replace all the convolutions with a single matrix multiplication. In order to learn more interesting functions, we have to add non-linearities. Historically, the popular choice was the sigmoid function defined as:

$$\text{sigmoid}(x) = \frac{1}{1 + e^{-x}} \tag{3.2}$$

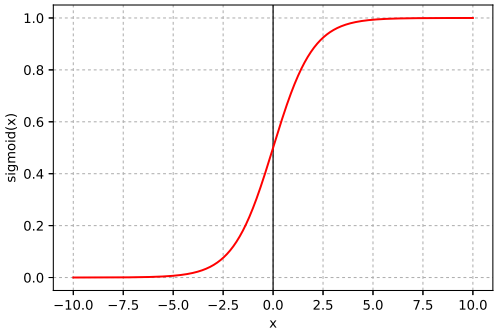


Figure 3.3: Sigmoid function

The sigmoid function is the simplest function having values between 0 and 1 mimicking the biological neurons 'firing' in reaction to their inputs. However, when the network is learning to minimise a loss function through backpropagation, the gradients tend to vanish to zero as the sigmoid's derivative goes to zero for high negative and positive values. The most popular choice of non-linearity is now the Rectified Linear Unit (ReLU) defined as:

$$\text{ReLU}(x) = \max(0, x) \tag{3.3}$$

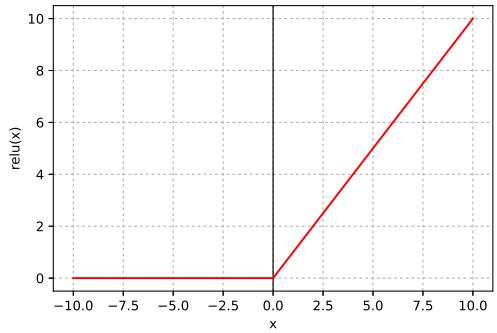


Figure 3.4: ReLU function

The ReLU's gradient is non-saturating for highly excited neurons which turns out to be a nice property to learn faster. In the network, each layer of convolution is followed by a ReLU layer, that simply applies the function $\max(0, x)$ to each neuron.

3.1.3 Pooling layer

There is a lot of spatial redundancy in an image, we don't need all the pixels to be able to identify what's in a picture. For example we can perfectly identify the animal in Figure 3.5 by reducing the number of pixels by two.

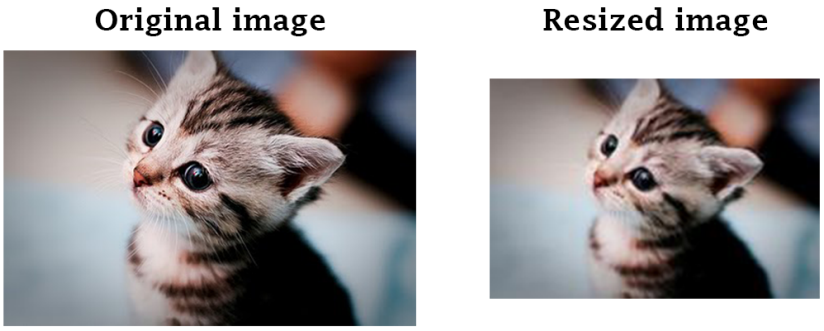


Figure 3.5: A kitten, and the same kitten with half the pixels

The same idea applies to convolved images, we might not need all the neurons that were created. The pooling operation downsamples the image in the following way:

- Pick a channel among the d ones.
- Position yourself on the top-left 2x2 square of the image and take the max.
- Repeat by sliding through the image vertically and horizontally with a stride/step of 2 (see Figure 3.6).

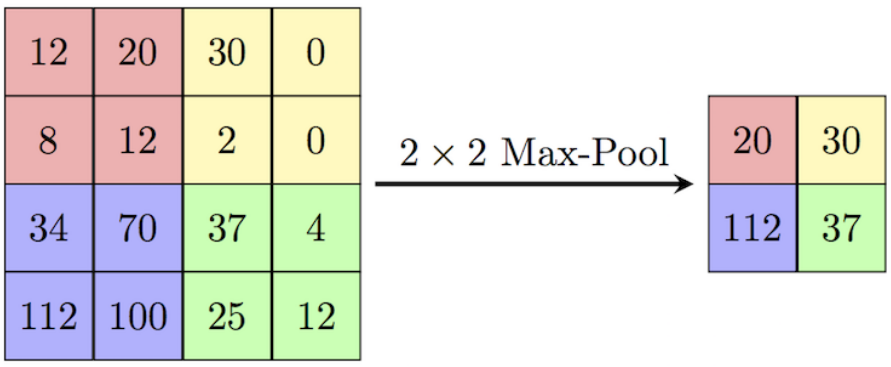


Figure 3.6: Max pooling [3]

After applying max pooling to each channel, the resulting image dimension is $(\frac{h}{2}, \frac{w}{2}, d)$ and we have discarded 75% of the neurons (as in each max-pool operation, we only keep the maximum neuron among the fours), effectively reducing the number of parameters and controlling overfitting.

You could wonder why max-pooling and not average pooling (taking the mean value of the four neurons)? The convolutions allow us to see if a certain feature is in the image when a neuron fires, and we only want to know if that feature is there in a certain region. Therefore taking the max of the four neurons is sufficient to know whether that feature is there or not in that particular region.

In practice, after a few iterations of convolutions, inserting pooling layers in-between convolutional layers might be a good idea to control the spatial complexity of the network.

3.1.4 An example of convolutional network

Here is an example of a convolutional neural network with an input image of size $(224, 224, 3)$:

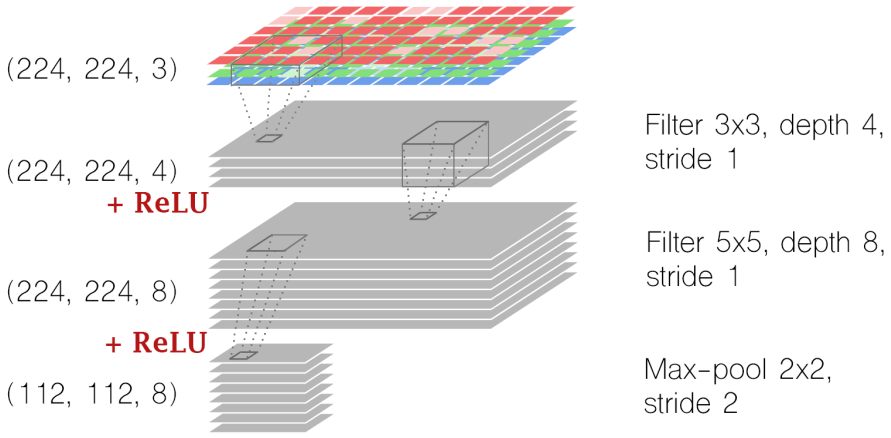


Figure 3.7: An architecture of a neural network [2]

- A first convolution with a filter of size 3×3 is applied, with depth 4, stride 1 and zero-padding of 1.
- A ReLU layer.

- A second convolution with a filter of size 5×5 is applied, with depth 8, stride 1 and zero-padding of 2.
- A ReLU layer.
- Max-pooling of size 2×2 with stride 2, reducing the height and width by 2.

After the last operation, the neurons are reshaped into a vector that can be fed to the traditional fully connected layers of neural networks.

3.1.5 Deep convolutional networks

Best results are achieved using deep convolutional networks, that is to say by stacking many layers of convolutions/ReLU/max-pool. But what exactly is 'many'? Let us have a look at the main Computer Vision competition: ImageNet Large Scale Visual Recognition (ILSVR).

1. **AlexNet** [4]: The first popular convolutional network, developed by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton, that outperformed the other competitors at ILSVR 2012 by a large margin: top-5 error of 16% compared to the runner-up with 26%. AlexNet has 5 convolutional layers (followed by ReLU), 3 max-pool layers and 3 fully-connected layers, producing a 8-layer deep network (not counting the max-pooling as it doesn't have any parameters).

2. **GoogLeNet** (also known as Inception) [5]: This is the winner of ILSVR 2014 with a top-5 error of 6.7% . This 22-layer architecture used the 'Inception Module' which allowed to drastically reduce the number of parameters: from 60M for AlexNet to 4M for GoogLeNet.

3. **ResNet** [6]: The winner of ILSVR 2015 with a top-5 error of 3.6% thanks to an astonishing 152 layers convolutional network. This architecture features 'skip connections' allowing this ultra-deep network achieve such results.

   (explain Inception module, talk about computational considerations (Stanford CS231n))

3.1.6 Transfer Learning

Training a convolutional network from scratch can be difficult as a large amount of data is needed and plenty of different architectures and hyperparameters need to be tried before finding a decent model. To circumvent that issue, you can take advantage of the pre-trained models available that learned to recognize images through near 1.2M training examples and a deep architecture that took weeks to train on multiple GPUs.

More specifically, the pre-trained networks learned to recognise features on a picture that allow it to classify the latter among the 1000 classes there are on the ImageNet dataset. Those features are combined in the final fully connected layer to make a decision. Suppose that instead of classifying an image into 1000 classes we want to label it according to 6 different emotions (happy, sad, angry, scared, surprised, disgusted). The same features can be combined in a different way to let the network take a decision on the emotion label of the image.

The process described above is called *Transfer Learning*: you chop off the last layer of the network and add your own layer given how many classes you have. You then freeze the weights of the other layers and only backpropagate through the newly created layer when training the network on your examples. If you have enough data, you can unfreeze more higher-level layers and backpropagate through them. One way to see why this works is that earlier features of ConvNets contain more generic features (such as edges or color blobs), while later features become more specific to the details of the classes present in the dataset. For example in ImageNet, there are many dog breeds and the later representational power might be used to distinguish those [7].

We will be using Google's Inception network and fine-tune the last fully-connected layer.

3.1.7 Google Inception network

After AlexNet proved that convolutional networks outperformed traditional machine learning models, the trend to achieve even better results was to build wider (more

units per layer) and deeper (more layers) networks and to add dropout to address overfitting. However, bigger networks are more expensive to train (more parameters) and could not be usable for real-time prediction if a forward-pass takes more than a second for instance. Moreover, if the increased capacity is not used efficiently, for example if most added weights are close to zero, then the network added depth will be completely wasted.

To address this problem, we could replace the fully connected layers by sparse ones, even inside convolutions [5]. Not only would it mimic more closely biological systems, but it would also have more theoretical ground thanks to the work of Arora et al. [8]. They proved that if the probability distribution of a dataset can be represented by a large, very sparse neural network, then it's possible to build the optimal network layer after layer by clustering highly correlated neurons of the preceding layer.

Current networks architectures are not using sparse layers as the libraries are heavily optimised for dense matrix multiplication. Training sparse layers would incur considerable overhead that are not handled by today's computing infrastructures. However, the Inception module is an elegant solution to add more expressiveness to your network while keeping the number of parameters low.

Inception module: At each layer, you'd normally be facing the dilemma of choosing between 1×1, 3×3, 5×5 convolution or max-pooling:

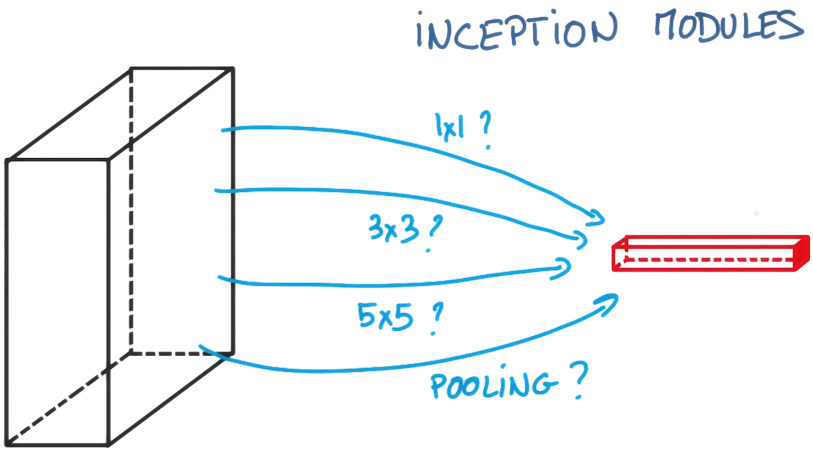


Figure 3.8: Which layer to choose? [9]

In the Inception module, you perform all of the above operations and let the network decide for you. Each operation is done in parallel before being concatenated and fed to the next layer. This allows to capture both local features via small convolutions and more high-level features via large convolutions.

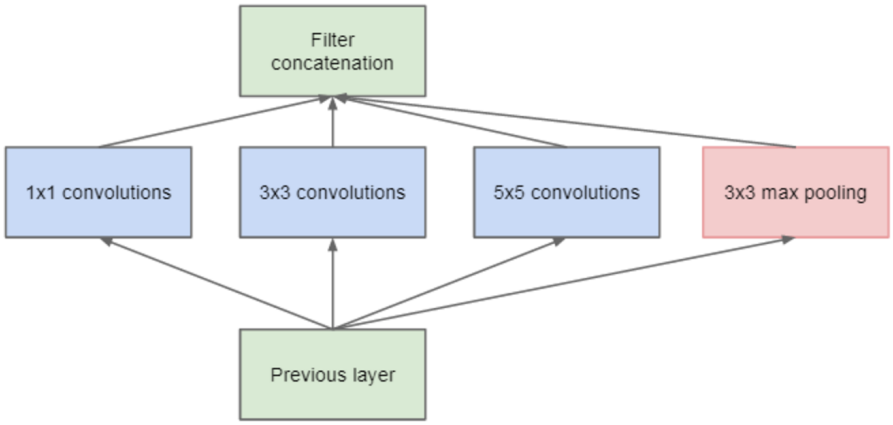


Figure 3.9: Naive Inception module [5]

But wait.. How is this supposed to keep the number of parameters low? This is actually the naive implementation of the Inception module. One key component are the 1×1 convolution for dimensionality reduction before the large convolutions. Let's take an example to understand: suppose at an arbitrary layer, your input is size $(14, 14, 480)$.

- **5×5 convolution, depth 48**: requires $(14^2)(480)(5^2)(48) = 112,896,000$ operations (supposing stride 1 and zero-padding).
- **1×1 convolution, depth 16 followed by 5×5 convolution, depth 48:** requires $[(14^2)(480)(1^2)(16)] + [(14^2)(16)(5^2)(48)] = 5,268,480$ operations.

That's more than twenty times faster! The number of parameters is also reduced by twenty as the reduction is the same (to get the actual number of parameters, you only need to divide the above by 14^2). The final Inception module is the following:

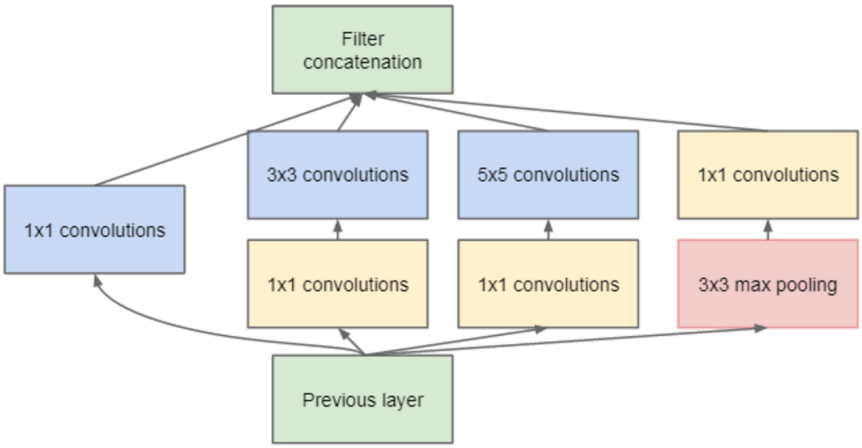


Figure 3.10: Inception module [5]

And the complete GoogleNet architecture being:

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figure 3.11: GoogleNet architecture [5]

The '#3 × 3 reduce' and '#5 × 5 reduce' refer to the dimensionality reduction with the 1×1 convolution. The 'pool proj' refers to the depth of the 1×1 convolution following the 3 × 3 max-pool with stride 1 and zero-padding 1.

In our model, we got rid of the last linear layer and replaced it with another linear layer of size $(1, 1, 6)$ for the 6 different emotions.

3.1.8 Results

Are the images enough to accurately determine the emotion conveyed by the user? We tried several machine learning models on the raw images, that were resized to a fixed size $(224, 224, 3)$. For now (need more time to train more data), the train set has 100 000 images and the test set has 20 000 images.

- **Softmax regression**: with L_2 regularisation of 0.01.
- **Random Forest Classification**: with 1000 trees and max depth of 5.
- **5-layer Convolutional Neural Network**: (each convolution is with stride 1, zero-padding to keep image size and followed by ReLU)
  - 3×3 convolution, depth 8
  - 3×3 convolution, depth 16
  - 2×2 max-pooling with stride 2
  - 3×3 convolution, depth 32
  - 2×2 max-pooling with stride 2
  - Fully connected layer
  - Fully connected layer
- **Inception fine-tuned**: As described in 3.1.6, we retrained the final layer of the Inception model with:
  - 10 epochs
  - Mini-batch size of 32
  - Adam optimizer with learning rate 1e−5

Here are the results, the measured accuracy being the fraction of correctly classified images on the test set:

Model	Parameters	Test accuracy
Softmax regression	L_2 reg: 0.01	0.40
Random Forest	1000 trees	0.50
5-layer ConvNet	architecture	0.60
Inception fine-tuned	architecture	0.65

Table 3.1: Comparison of models using raw images

Chapter 4

Natural Language Processing

Text analysis

4.1 Section 1

Chapter 5

Recurrent Neural Networks for text generation

5.1 Section 1

Chapter 6

Useful maths in LaTeX

6.1 Equations related

$$\frac{\partial u_1}{\partial t} = \Delta w_1 \quad \text{in } \Omega, t > 0, \tag{6.1}$$

$$\frac{\partial u_2}{\partial t} = \Delta w_2 \quad \text{in } \Omega, t > 0, \tag{6.2}$$

where

$$w_1 = \frac{\delta F(u_1, u_2)}{\delta u_1}, \tag{6.3}$$

$$w_2 = \frac{\delta F(u_1, u_2)}{\delta u_2}, \tag{6.4}$$

$$\begin{aligned} F(u_1, u_2) = {} & b_1 u_1^4 - a_1 u_1^2 + c_1 |\nabla u_1|^2 \\ & + b_2 u_2^4 - a_2 u_2^2 + c_2 |\nabla u_2|^2 \\ & + D \left(u_1 + \sqrt{\frac{a_1}{2b_1}} \right)^2 \left(u_2 + \sqrt{\frac{a_2}{2b_2}} \right)^2 . \end{aligned} \tag{6.5}$$

$$U_1^n = \sum_{i=1}^{J} U_{1,i}^n \eta_i, \quad W_1^n = \sum_{i=1}^{J} W_{1,i}^n \eta_i, \tag{6.6}$$

$$U_2^n = \sum_{i=1}^{J} U_{2,i}^n \eta_i, \quad W_2^n = \sum_{i=1}^{J} W_{2,i}^n \eta_i, \tag{6.7}$$

We also use the following notation, for $1 \leq q < \infty$,

$$L^q(0,T;W^{m,p}(\Omega)) := \left\{ \eta(x,t) : \ \eta(\cdot,t) \in W^{m,p}(\Omega), \int_0^T \|\eta(\cdot,t)\|_{m,p}^q \, dt < \infty \right\},$$
$$L^\infty(0,T;W^{m,p}(\Omega)) := \left\{ \eta(x,t) : \eta(\cdot,t) \in W^{m,p}(\Omega), \ \operatorname*{ess\,sup}_{t\in(0,T)} \|\eta(\cdot,t)\|_{m,p} < \infty \right\},$$

Cases

$$|v|_{0,r} \leq C|v|_{0,p}^{1-\mu}\|v\|_{m,p}^{\mu}, \quad \text{holds for } r \in \begin{cases} [p,\infty] & \text{if } m - \frac{d}{p} > 0, \\ [p,\infty) & \text{if } m - \frac{d}{p} = 0, \\ [p, -\frac{d}{m-d/p}] & \text{if } m - \frac{d}{p} < 0. \end{cases} \tag{6.8}$$

6.2 Writing

Lemma 6.2.1 Let $u, v, \eta \in H^1(\Omega)$, $f = u - v$, $g = u^m v^{n-m}$, $m, n = 0, 1, 2$, and $n - m \geq 0$. Then for $d = 1, 2, 3$,

$$\left| \int_\Omega f g \eta dx \right| \leq C|u - v|_0 \, \|u\|_1^m \, \|v\|_1^{n-m} \, \|\eta\|_1. \tag{6.9}$$

Proof: Note that using the Cauchy-Schwarz inequality we have

$$|(u)^m v^{n-m}|_{0,p} \leq \begin{cases} |u|_{0,2mp}^m \, |v|_{0,2(n-m)p}^{(n-m)} & \text{for } \ n - m \neq 0, \ \text{ and } \ m \neq 0, \\ |u|_{0,mp}^m \ \text{ or } \ |v|_{0,(n-m)p}^{(n-m)} & \text{for } \ m = 0, \ \text{ or } \ n - m = 0 \ \text{ respectively.} \end{cases}$$

Noting the generalise Hölder inequality and the result above we have

$$\begin{aligned} \left| \int_\Omega f g \eta dx \right| &\leq |u - v|_0 \, |u^m v^{n-m}|_{0,3} \, |\eta|_{0,6}, \\ &\leq |u - v|_0 \ \ |\eta|_{0,6} \begin{cases} |u|_{0,6}^2 & \text{for } \ m = 2, \\ |u|_{0,6} \, |v|_{0,6} & \text{for } \ m = 1, \\ |v|_{0,6}^2 & \text{for } \ m = 0, \end{cases} \\ &\leq C|u - v|_0 \, \|u\|_1^m \, \|v\|_1^{n-m} \, \|\eta\|_1, \end{aligned}$$

where we have noted (6.8) to obtain the last inequality. This ends the proof. □

We consider the problem:

(**P**) Find $\{u_i, w_i\}$ such that $u_i \in H^1(0,T;(H^1(\Omega))') \cap L^\infty(0,T;H^1(\Omega))$ for *a.e.* $t \in (0,T)$, $w_i \in L^2(0,T;H^1(\Omega))$

$$\left\langle \frac{\partial u_1}{\partial t}, \eta \right\rangle$$

Chapter 7

Conclusions

Bibliography

[1] **Tumblr photos:**
http://fordosjulius.tumblr.com/post/161996729297/just-relax-with-amazing-view-ocean-and
http://ybacony.tumblr.com/post/161878010606/on-a-plane-bitchessss-we-about-to-head-out
https://little-sleepingkitten.tumblr.com/post/161996340361/its-okay-to-be-upset-its-okay-to-not-always-be
http://shydragon327.tumblr.com/post/161929701863/tensions-were-high-this-caturday
https://beardytheshank.tumblr.com/post/161087141680/which-tea-peppermint-tea-what-is-your-favorite
https://idreamtofflying.tumblr.com/post/161651437343/me-when-i-see-a-couple-expressing-their-affection

[2] **Convolution images:** M. Gorner, Tensorflow and Deep Learning without a PhD, Presentation at *Google Cloud Next '17*
https://docs.google.com/presentation/d/1TVixw6ItiZ8igjp6U17tcgoFrLSaHWQmMOwjlgQY9co/pub?slide=id.g1245051c73_0_2184
The slide on the convolutional neural network was adapted to our architecture.

[3] **Max pooling:** Cambridge Spark
https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

[4] A. Krizhevsky, I. Sutskever and G. Hinton, ImageNet Classification with Deep Convolutional Neural Networks. In *NIPS*, 2012.

[5] C. Szegedy et al., Going deeper with convolutions. In *CVPR*, 2015.

[6] K. He et al., Deep Residual Learning for Image Recognition. In *CVPR*, 2016 .

[7] A. Karpathy, L. Fei-Fei, J. Johnson, Transfer Learning. In *Stanford CS231n Convolutional Neural Networks for Visual Recognition*, 2016.

[8] S. Arora et al., Provable Bounds for Learning Some Deep Representations. In *ICML*, 2014.

[9] Video explaning Inception Module, https://www.youtube.com/watch?v=VxhSouuSZDY.

Appendix A

Basic and Auxiliary Results

A.1 Basic Results