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 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 high-dimensional word embedding followed by a recurrent layer process the textual information to accurately infer the emotion of the post. The network architecture, named Deep Sentiment, can also be adapted to generate images and text given an emotion.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Sentiment analysis has been an active area of research in the past few years, especially on the readily available Twitter data, e.g. Bollen et al. [2] who investigated the impact of collective mood states on stock market or Flaxman et al. [1] who analysed day-of-week population well-being.

Contrary to Twitter, Tumblr's posts are not limited to 140 characters, allowing more expressiveness, and are not centered on the textual content but on the image content instead. A Tumblr post will almost always be an image with some text accompanying the latter. Pictures have become prevalent on social media and characterising them could enable the understanding of billions of users.

http://www.ifp.illinois.edu/ jyang29/papers/AAAI15-sentiment.pdf

We propose a novel method to uncover the emotional of an individual posting on social media. The ground truth emotion will be extracted from the tags, considered as the 'self-reported' emotion of the user. Our model incorporates both text and image and we aim to 'read' them to be able to understand the emotional content they imply about the user.

Chapter 2

Tumblr Data

2.1 Overview of the Data

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

Figure 2.1: An example of a Tumblr post

The tags '#chowchow # home #happy #bluetongue' are really valuable as they indicate the user's state of mind when writing that post. Ekman popularised the idea that there are six basic emotions [3]: hapiness, sadness, anger, surprise, fear, disgust. These emotions are said to be *basic* as they are hardwired regardless of the species. Basic emotions are innate, universal and automatic and induce fast reactions that are linked with a high survival rate.

To build our dataset, queries were made searching for each of the six emotions appearing in the tags. Adjectives were used as they were more commonly used by users: #happy, #sad, #angry, #surprised, #scared and #disgusted. Each post would then contain 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.

Note that sometimes, a post would contain several basic emotions such as '#sad #angry'. We simply selected the first hashtag written by the user as it can be deemed as the main emotion that the user first thought of.

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 also appeared in the text itself, for instance:

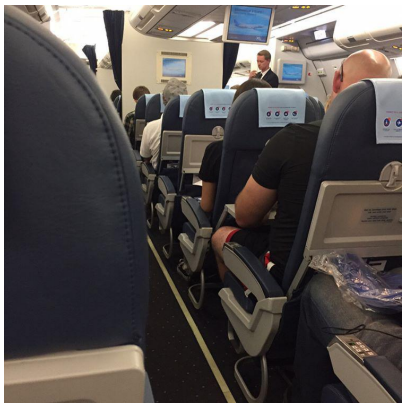
> *"When you're on vacations and there is a rainstorm. #fail #sad".*

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 be removed from the training data. Basically, if a post contains less than t, a threshold, English word, it is deemed as non-English and removed from the dataset. The threshold was set to 5 English words as it appears to filter out reasonably well the dataset. The vocabulary of english words was obtained from Word2Vec and will be detailed further in Section 4.

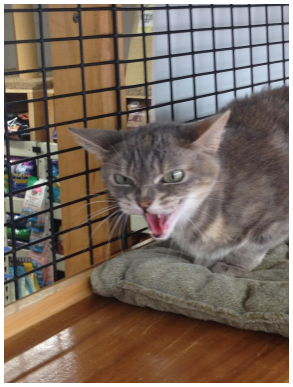
Here are examples of posts with their associated emotions:

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

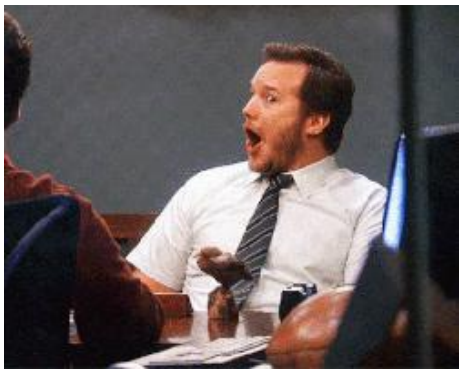


(b) **Scared**: "On a plane guys! We're about to head out into the sky to Paris, France #Paris #trip #kinda #nervous #fun #vacations"

(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 illustrated by Tumblr posts [4]

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.

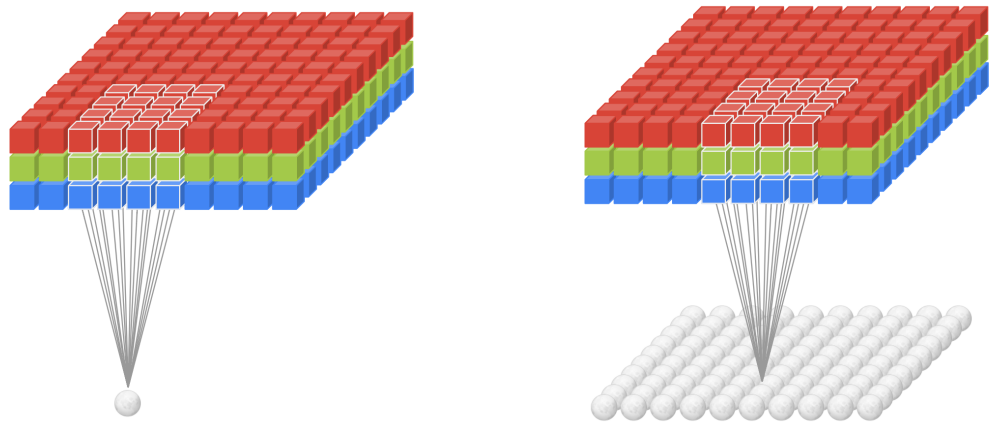
3.1 Convolutional Neural Networks

Convolutional neural networks, often called ConvNets, were inspired by the work of Hubel and Wiesel [**?**] on the human visual cortex. They found that our visual cortex operates as a complex arrangement of cells, where each cell is receptive to a small region of the visual field and is called a *receptive field*.

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 and green. If you simply flatten that image and transform it into a vector of size $h \times w \times 3$ and feed it to a neural network, you'll get poor 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.
- Position the filter on the upper left of the image and element-wise multiply the $f \times f \times 3$ chunk of image 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).



(a) One operation of convolution

(b) The result of the convolution

Figure 3.1: A convolution, each neuron been a 'receptive field' [5]

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 h_{new} to be equal to h:

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

Solving (3.1) gives $p = \frac{f-1}{2}$. 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

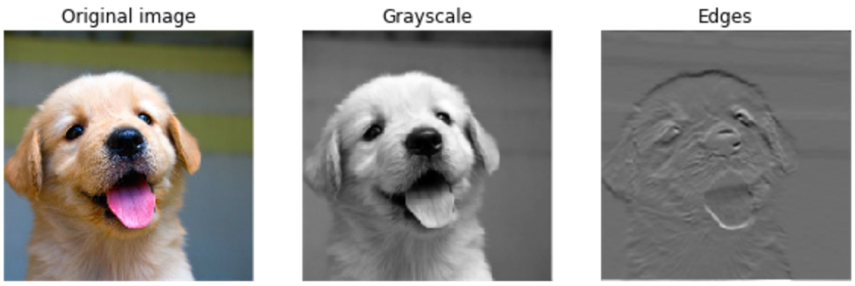


Figure 3.2: Examples of convolution

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 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 the 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.

3.1.2 ReLU Layer

Stacking convolutions is nice, but as it is, we are only creating 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 – that is to say transforming the tensor (h, w, d) with a non-linear function. Historically, the popular choice was the sigmoid function defined as:

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

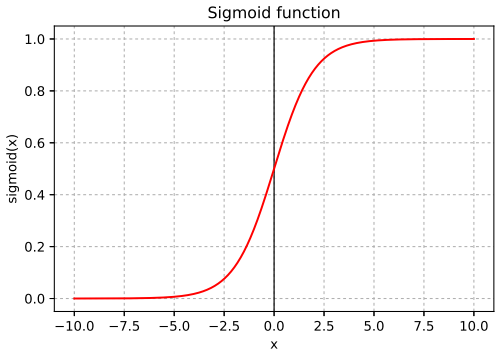


Figure 3.3: Sigmoid function

The sigmoid function is the simplest function to have 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 values that are highly negative or positive. The most popular choice of non-linearity is now the Rectified Linear Unit (ReLU) defined as:

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

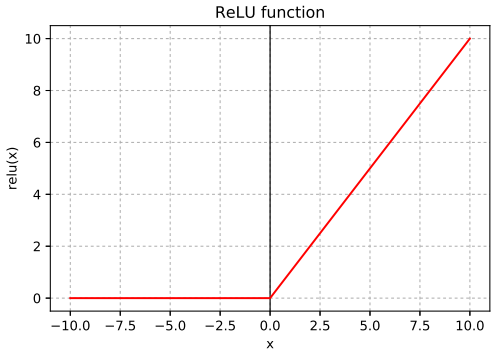


Figure 3.4: ReLU function

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.

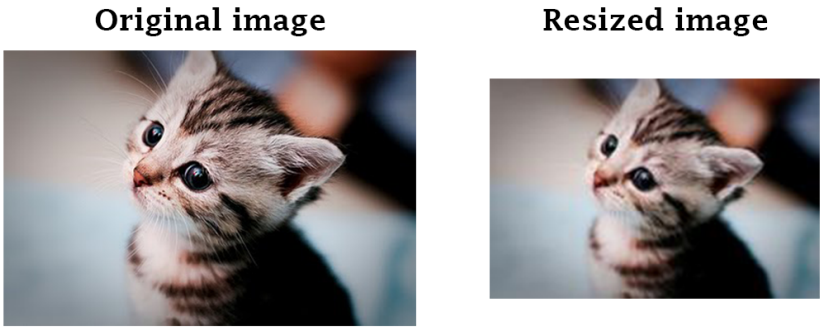


Figure 3.5: A kitten, and the same kitten with half the pixels

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.
- Start at 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).

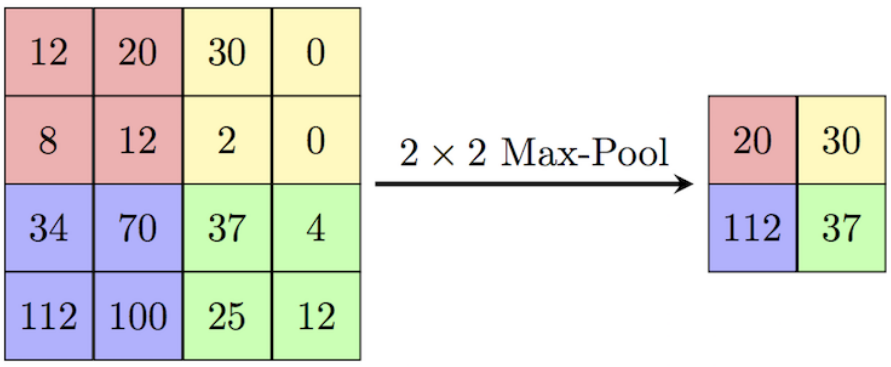


Figure 3.6: Max pooling [6]

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 $(224, 224, 3)$:

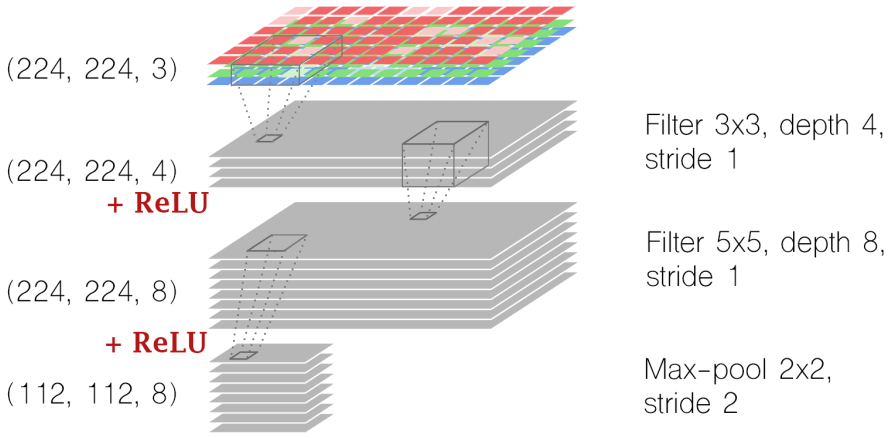


Figure 3.7: The architecture of a convolutional neural network [5]

- 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.

3.2 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** [7]: 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) [8]: The the winner of ILSVR 2014 with a top-5 error of 6.7% . This 22-layer architecture used the 'Inception Module' (that will be described shortly) which allowed to drastically reduce the number of parameters: from 60M for AlexNet to 4M for GoogLeNet.

3. **ResNet** [9]: 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.

3.3 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 from the ImageNet dataset, 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 in order to classify the latter among the 1000 classes on the ImageNet dataset. Those features are combined in the final output layer (of size 1000, each neuron being a class probability). 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 about what is the emotion of the image.

The process described above is called *Transfer Learning*: we chop off the last layer of the network and add our own layer given how many classes we have. We then freeze the weights of the other layers and only backpropagate through the newly created layer when training the network on our examples. If we have enough data, we can unfreeze more higher-level layers and backpropagate through them.

Earlier features of ConvNets contain more generic features (such as edges or color blobs) that can be used for any task, 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 [10]. We will be using Google's Inception network and fine-tune through the last 3 layers. (that number is subject to change)

3.4 Google Inception Network

3.4.1 Motivation

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 extra depth and width will be completely wasted.

To address this problem, we could replace the fully connected layers by sparse ones, even inside convolutions [8]. 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. [11]. 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 a network while keeping the number of parameters low.

3.4.2 Inception Module

At each layer, we would normally be facing the dilemma of choosing between 1×1, 3×3, 5×5 convolution or max-pooling:

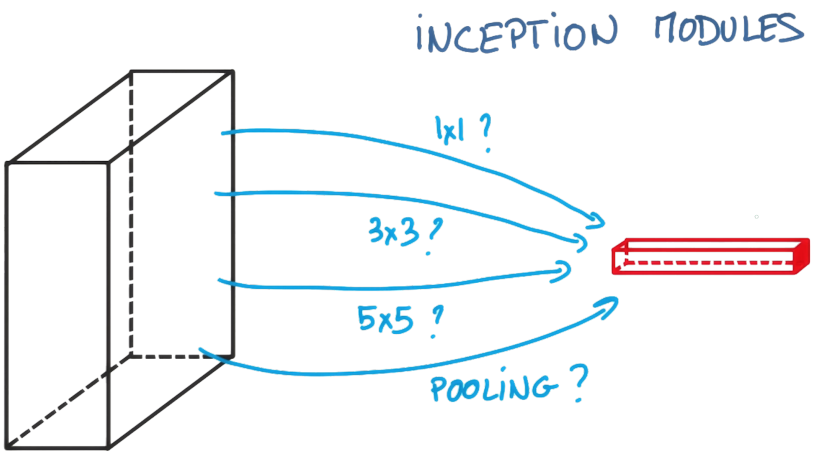


Figure 3.8: Which layer to choose? [12]

In the Inception module, we perform all of the above operations and let the network decide for us. 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.

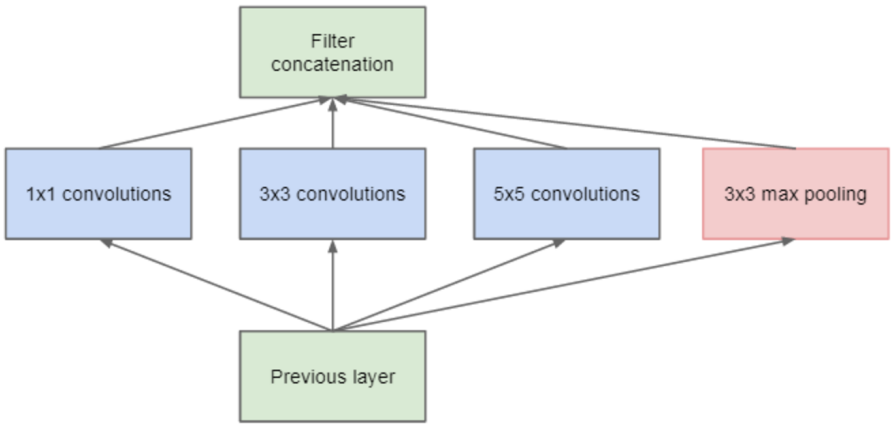


Figure 3.9: Naive Inception module [8]

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 convolutions, which behave like clustering the highly correlated neurons, before the large convolutions. Let's take an example to understand: suppose at an arbitrary layer, your input is size $(14, 14, 480)$.

- A **5×5 convolution, depth 48**: requires $(14^2)(480)(5^2)(48) = 112,896,000$

operations (supposing stride 1 and zero-padding).

- An **1×1 convolution, depth 16, followed by a 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 factor 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:

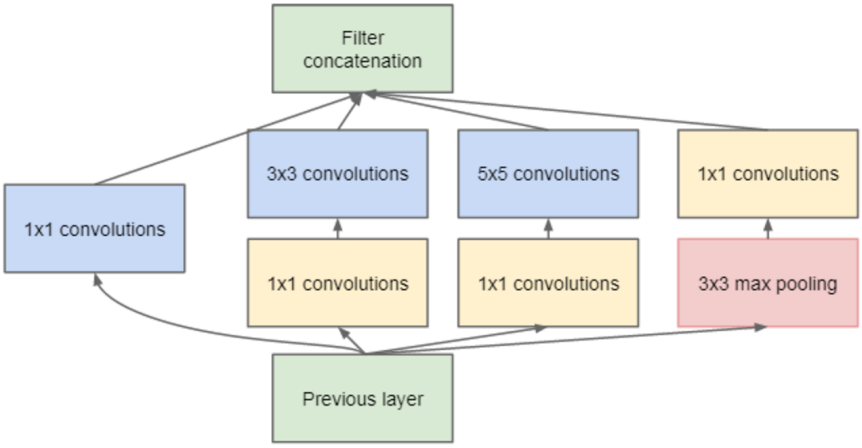


Figure 3.10: Inception module [8]

3.4.3 GoogleNet

The complete GoogleNet architecture is:

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 [8]

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.

3.5 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:
  - 20 epochs (1 epoch = 1 full sampling of the training data)
  - 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	0.18
Random Forest	1000 trees	0.20
5-layer ConvNet	see above	??
Inception fine-tuned	see above	??

Table 3.1: Comparison of models using raw images

Chapter 4

Deep Sentiment

Deep Sentiment is the name of the deep neural network architecture merging both visual recognition and text analysis.

4.1 The Architecture

An illustration is worth a thousand words:

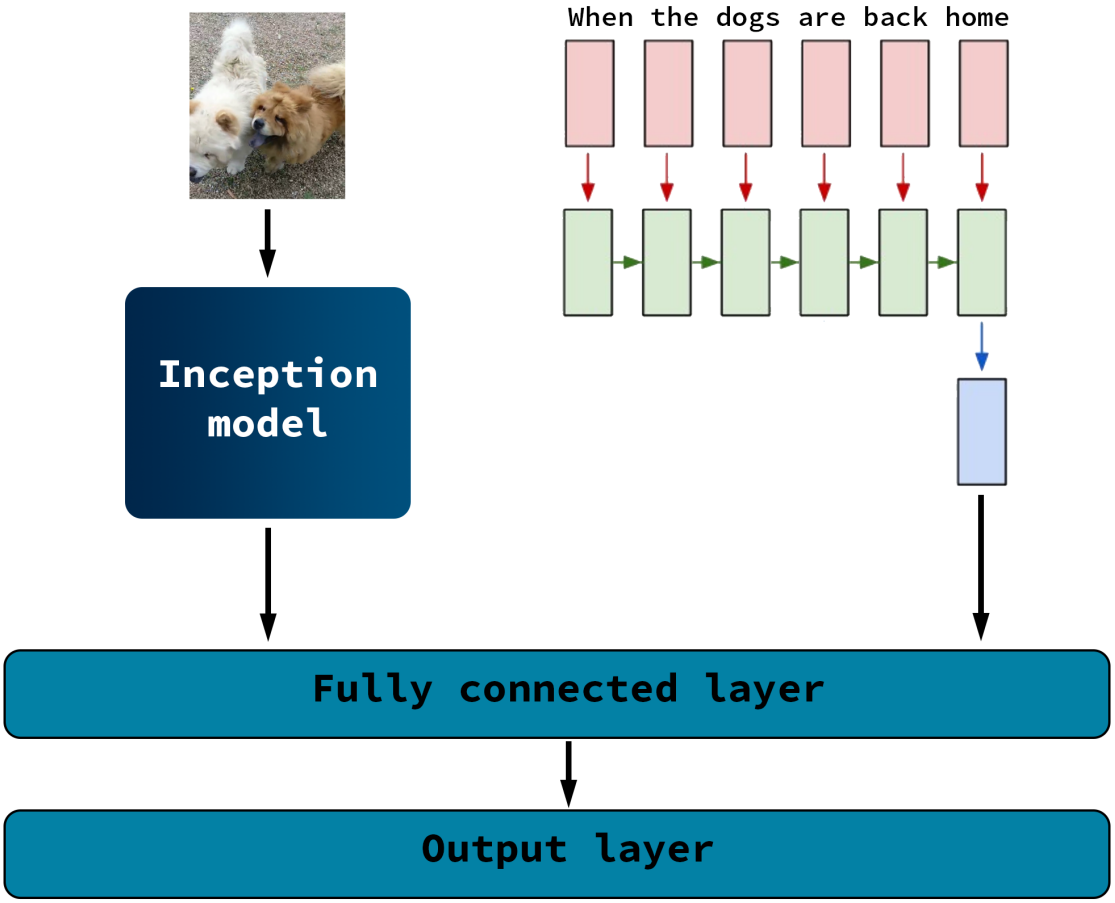


Figure 4.1: Deep Sentiment architecture

1. The image will go through the pre-trained Inception model that will extract features from the images.

2. The text will be embedded in a high-dimensional space with Word2Vec and will be fed to an LSTM.

3. The two outputs will be combined in a fully-connected layer.

4. The final layer will give the probability distribution of the emotion of the post.

Chapter 5

Generation of Tumblr Posts

This chapter will be about image and text generation.

Bibliography

[1] S. Flaxman and K. Kassam, On #agony and #ecstasy: Potential and pitfalls of linguistic sentiment analysis. In preparation, 2016.

[2] J. Bollen, H. Mao, X.-J. Zeng, Twitter mood predicts the stock market. In *Journal of Computational Science*, 2011.

[3] P. Ekman, An Argument for Basic Emotions. In *Cognitive and Emotion*, 1992.

[4] 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

[5] D. H. Huble and T. N. Wiesel, Receptive fields and functional architecture of monkey striate cortex. In *Journal of Physiology (London)*, 1968.

[6] 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.

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

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

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

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

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

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

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

[14] T. Mikolov et al., Efficient Estimation of Word Representations in Vector Space. In *ICLR*, 2013.

[15] TensorFlow, Word2Vec tutorial, https://www.tensorflow.org/tutorials/word2vec.

[16] C. McCormick, Word2Vec Tutorial - The Skip-Gram Model, http://mccormickml.com/2016/04/19/word2vec-tutorial-the-skip-gram-model/.

[17] T. Mikolov et al., Distributed Representations of Words and Phrases and their Compositionality. In *NIPS*, 2013.

[18] A. Mnih and Y. W. Teh, A fast and simple algorithm for training neural probabilistic language models. In *ICML*, 2012.

[19] B. Zoph et al., Simple, Fast Noise-Contrastive Estimation for Large RNN Vocabularies. In NAACL, 2016.

[20] Word2Vec pre-trained model, Google, 2013. https://code.google.com/archive/p/word2vec/

[21] S. Flaxman et al., Who Supported Obama in 2012? Ecological Inference through Distribution Regression. In *KDD*, 2015.

[22] A. Karpathy, The Unreasonable Effectiveness of Recurrent Neural Networks, 2015. http://karpathy.github.io/2015/05/21/rnn-effectiveness/

[23] C. Olah, Understanding LSTM Networks, 2015. http://colah.github.io/posts/2015-08-Understanding-LSTMs/