Universitat Politècnica de Catalunya (UPC) - BarcelonaTech

Facultat d'Informàtica de Barcelona (FIB)

Speeding up Document Image Classification

Javier Ferrando Monsonís

Master in Innovation and Research in Informatics
Data Science

Advisor: Jordi Torres Viñals
Universitat Politècnica de Catalunya (UPC) - BarcelonaTech
Barcelona Supercomputing Center - Centro Nacional de Supercomputación

June, 2020

Abstract

This work presents a solution by means of light Convolutional Neural Networks (CNNs) in the Document Classification task, essential problem in the digitalization process of institutions. We show in the RVL-CDIP dataset that we can achieve state-of-the-art results with a set of lighter models such as the EfficientNets and present its transfer learning capabilities on a smaller in-domain dataset such as Tobacco3482. Moreover, we present an ensemble pipeline which is able to boost solely image input by combining image model predictions with the ones generated by BERT model on extracted text by OCR. We also show that the batch size can be effectively increased without hindering its accuracy so that the training process can be sped up by parallelizing throughout multiple GPUs, decreasing the computational time needed.

Acknowledgements

I would like to give special thanks to my advisor Jordi Torres for understanding what my interests were and for allowing me to focus on them, for the great advice and for giving me the freedom to learn that much. I would also like to appreciate the help of Juan Luis Domínguez, one of the greatest problem solvers I have ever met, for always giving me sensational inputs and for fixing countless issues I have had.

I would also like to express my gratitude to everyone in the Barcelona Supercomputing Center who has contribute to this work, from my colleagues in the C6-E201 to the Support team.

Lastly, thanks to the collaboration with CaixaBank for creating the opportunity that has led to this work.

Contents

List of Figures

1 Introduction

This thesis emerges from a real problem that appears in the document digitization process, which has become a common practice in a wide variety of industries that deal with vast amounts of archives. This is the case of the financial institution CaixaBank, collaborator of the Barcelona Supercomputing Center - Centro Nacional de Supercomputación[1]. The proposed issue can be categorized as a document classification problem, which is a task to face when trying to automate their document processes. Although initially the target of this work was to give a solution to the aforementioned institute, we present the results using publicly available data. In Figure 1 we show one document for each of the classes from the used dataset. From left to right: Letter, Form, Email, Handwritten, Advertisement, Scientific report, Scientific publication, Specification, File folder, News article, Budget, Invoice, Presentation, Questionnaire, Resume and Memory.

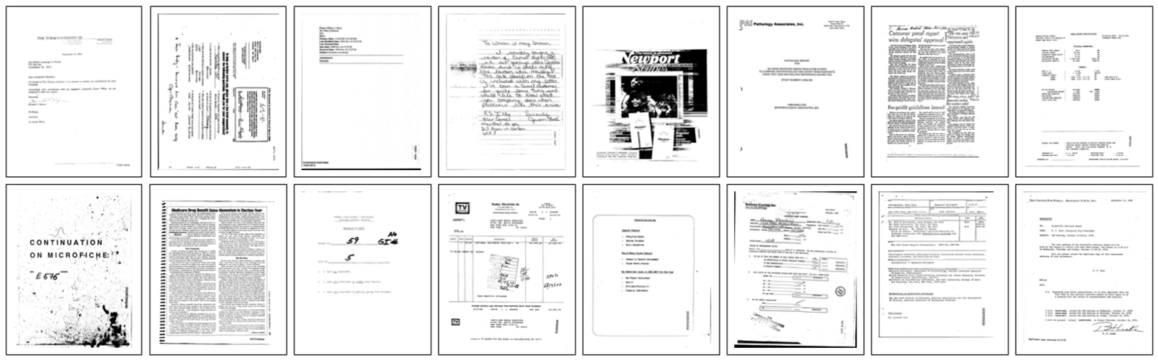

Figure 1: One document example of each of the classes in BigTobacco

The goal is to automatically classify these kinds of documents as accurately as possible. The main problem is the high intra-class and low inter-class variability between documents, which makes this a challenging problem.

Based on the topic literature, first attempts focused on structural similarity between documents [56] and on feature extraction [36, 16, 42] to differentiate characteristics of each class. The combination of both approaches has also been tested [18].

Several classic machine learning techniques have been applied to this problem, i. e. K-Nearest Neighbor approach [9], Hidden Markov Model [24] and Random Forest Classifier [41, 36] while using SURF local descriptors before the Convolutional Neural Networks (CNNs) came into scene.

With the rise of Deep Learning, researchers have tried deep neural networks to improve the accuracy of their classifiers. CNNs have been proposed in past works, initially in 2014 by Le Kang *et al.* [38] who started with a simple 4-layer CNN trained from scratch. Then, transfer learning was demonstrated to work effectively [29, 1] by

[1]https://www.bsc.es

using a pretrained network on ImageNet [22], and later models have become increasingly heavier [1, 63, 21, 2] with the speed and computational resources drawback this entails.

Table 1: Parameters of the CNNs architectures used in BigTobacco.

Model	#Params
AlexNet	60.97M
VGG-16	138.36M
ResNet-50	25.56M
Inception-V3	23.83M
EfficientNet-B2	**9.2M**
EfficientNet-B0	**5.3M**

Recently, textual information has been used by itself or as a combination together with visual features extracted by the previously mentioned models. Although Optical Character Recognition (OCR) is prone to errors, particularly when dealing with handwritten documents, the use of modern Natural Language Processing (NLP) techniques have demonstrated a boost in the classifiers performance [50, 7, 6].

The contributions of this work can be summarized in two main topics:

- Algorithmic performance: we propose a model and a training procedure to deal with images and text that outperforms the state-of-the-art in several settings and is lighter than any previous neural network used to classify the BigTobacco dataset, the most popular benchmark for Document Image Classification (Table 1).

- Training process speed up: we demonstrate the ability of these models to maintain their performance while saving a large amount of time by parallelizing over several GPUs. We also show the performance differences between the two PyTorch implementations to solve this task.

1.1 Document Image Classification

Document Image Classification task tries to predict the class which a document belongs to by means of analyzing its image representation. This challenge can be tackled in two ways, as an image classification problem and as a text classification problem. The former tries to look for patterns in the pixels of the image to find elements such as shapes or textures that can be associated with a certain class. The latter tries to understand the language written in the document and relate this to the different classes.

1.1.1 Datasets

As mentioned earlier, in this work we make use of two publicly available datasets containing samples of images from scanned documents from USA Tobacco companies, published by Legacy Tobacco Industry Documents created by the University of California San Francisco (UCSF). We find these datasets a good representation of what enterprises and institutions may face with, based on the quality and type of classes. Furthermore, they have been go-to datasets in this research field since 2014 with which we can compare results.

RVL-CDIP (Ryerson Vision Lab Complex Document Information Processing) is a 400.000 document sample (BigTobacco from now onwards) presented in [29] for document classification tasks. This dataset contains the first page of each of the documents, which are labeled in 16 different classes with an equal number of elements per class. A smaller sample containing 3482 images was proposed in [36] as Tobacco3482 (SmallTobacco henceforth). This dataset is formed by documents belonging to 10 classes not uniformly distributed.

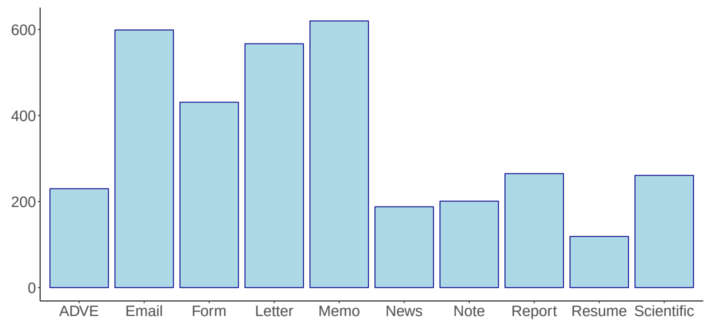

Figure 2: SmallTobacco classes distribution

In order to work with appropriate data formats to both feed correctly into the classification system and be able to store it in our cluster's memory, we need to transform the datasets. In this case, we build HDF5 files for the datasets to work with PyTorch and TFRecords in the TensorFlow case. Both HDF5 and TFRecord formats serialize the dataset into a single file, which makes it much easier to deal with.

To deal with text data in the SmallTobacco case, we extract the characters from the document images by applying OCR Tesseract [58] to them.

We provide scripts to run OCR, save the datasets in HDF5 and TFRecords formats in the accompanying repository https://javiferran.github.io/document-classification/.

1.1.2 Technologies

In this work a bunch of frameworks have been used. The Deep Learning library used to develop all the different experiments have been PyTorch[2], although we also pro-

[2]https://pytorch.org/

vide some of the code for its usage in TensorFlow[3]. PyTorch, developed by Facebook, and TensorFlow, by Google, are the two main Deep Learning frameworks nowadays. PyTorch is the favorite for researchers while TensorFlow is more widely used in production by companies. We selected PyTorch because its compatibility with some of the NLP frameworks like FLAIR[4] (from Zalando research) and Transformers[5] (from HuggingFace), although the latter has been adapted to TensorFlow while the development of this work. Since PyTorch is more used in research and this project deals with some state-of-the-art results, the fact that some of the code of the lately results are just available in PyTorch, especially in the NLP field, has made us choose this option. As mentioned earlier, the main two NLP libraries used in this project are FLAIR and Transformers. The first includes a large number of NLP models, from which one can obtain word embeddings or even NLP task ready output in a user-friendly manner. The second focuses on Transformer architecture, providing pre-trained models in different domains and languages.

[3]https://www.tensorflow.org/

[4]https://github.com/flairNLP/flair

[5]https://huggingface.co/transformers/

2 Theoretical Framework

2.1 Deep Learning

The proposed methods in this work are based on supervised Deep Learning, where each document is associated with a class (label) so that the algorithms are trained by minimizing the error between the predictions and the truth. Deep Learning is a branch of machine learning that deals with deep neural networks, where each of the layers is trained to extract higher-level representations of the previous ones. These models are trained by solving iteratively an unconstrained optimization problem. In each iteration, a random batch of the training data is fed into the model to compute the loss function value. Then, the gradient of the loss function with respect to the weights of the network is computed (backpropagation) and an update of the weights in the negative direction of the gradient is done. These networks are trained until they converge into a loss function minimum.

2.2 Deep Feedforward Networks

Deep feedforward networks, also known as multilayer perceptrons (MLPs) try to approximate a function $y = f^*(x, \theta)$ by adjusting the θ parameters that better fits to the 'label' data y. Usually, this function describing the network is composed by multiple functions, each one representing one layer. At the same time, each layer is formed by several neurons, which fire following an activation function output. Every of these networks has an input layer, an output layer and can have one or several hidden layers. The number of layers is called the depth of the network, usually the term 'deep' network are used to those with more than one hidden layer.

These networks are named feedforward because information flows in a one-way direction, with no recursive steps such as in Recurrent Neural Networks.

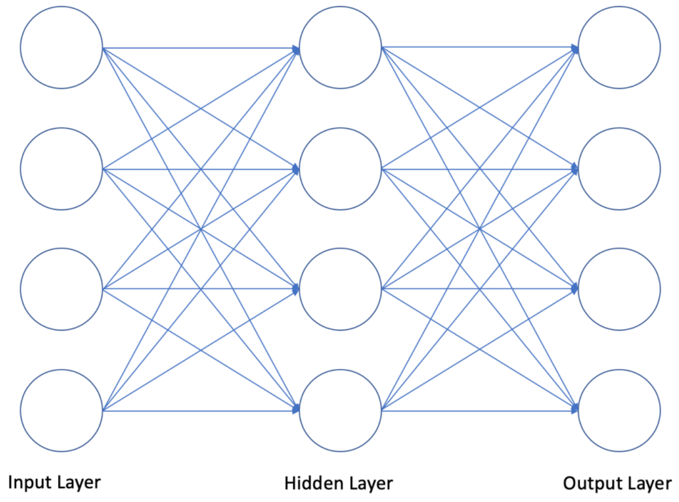

Figure 3: Fully-connected feedforward network

In the above figure it is depicted a fully-connected feedforward network, where each neuron is connected to every neuron of the previous layer. The mathematical expression of the computation done in a neuron j of the layer l is:

$$n_{jl} = \sum_{i=0}^{Dim(l^{-1})} w_{ij}^{l} o_i^{l-1}$$

Where o^{l-1} is the previous layer output and w_{ij} represents the weight associated with the neuron j and the previous layer output o_i. The activation function applies a non-linear transformation to n_{jl}, which could be a differentiable function mapping values to [0,1] range such as the sigmoid function, giving $o_i^l = g(n_{jl})$. A bias term b can be added to the calculation.

One way to represent the previous calculations for a complete layer transformation is by means of a weight matrix W, where each row stores the weights of one neuron, so the output of a hidden layer (l) can be computed by $o^l = g(o^{l-1}W^l + b)$.

2.3 Convolutional Neural Networks

The most used architecture in Computer Vision have been CNN-based networks. Their main operation is the convolution one, which consists on a succession of dot products between the vector representations of both the input space[6] $(L_q \times B_q \times d_q)$ and the filters $(F_q \times F_q \times d_q)$, which are learnable parameters. We slide (or convolve) each filter around the input volume getting an *activation map* of dimension $L_{q+1} = (L_q - F_q + 1)$ and $B_{q+1} = (B_q - F_q + 1)$. The output volume then has a dimension of $L_{q+1} \times B_{q+1} \times d_{q+1}$, where d_{q+1} refers to the number of filters used.

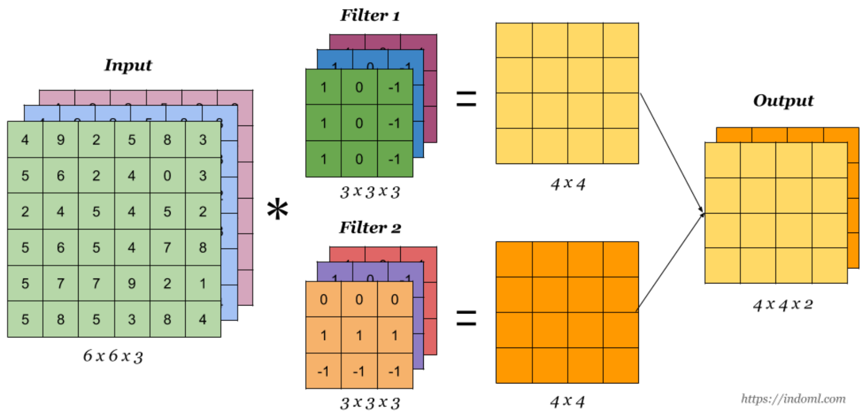

Figure 4: Applying two $3 \times 3 \times 3$ filters over a $6 \times 6 \times 3$ input space. Source [35]

Usually, each convolution layer is associated with an activation layer, where an activation function is applied elementwise to the whole output volume, without changing

[6]Same notation as in [3]

its dimension. To reduce the number of parameters of the network and its computational costs, a pooling layer is typically located between convolution operations. The pooling layer takes a region $P_q \times P_q$ in each of the d_q activation maps and performs an arithmetic operation as shown in Figure 5. The most used pooling layer is the max-pool, which returns the maximum value of the aforementioned region.

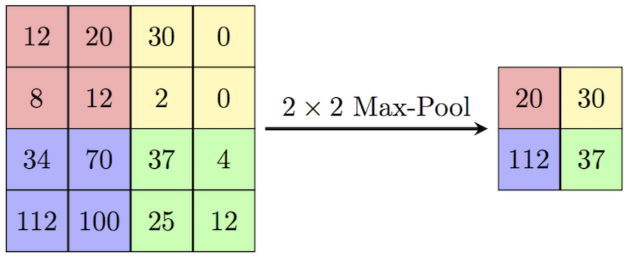

Figure 5: Max-pooling operation. Source [19]

Some important hyperparameters to take into account when building CNNs are the stride, the zero-padding, the number of filters and their size F_q. The stride defines the number of elements in the input matrix that are jumped in each slide. The higher the stride the smaller the output volume. Zero-padding expands the input volume by including zero elements around it as shown in Figure 6.

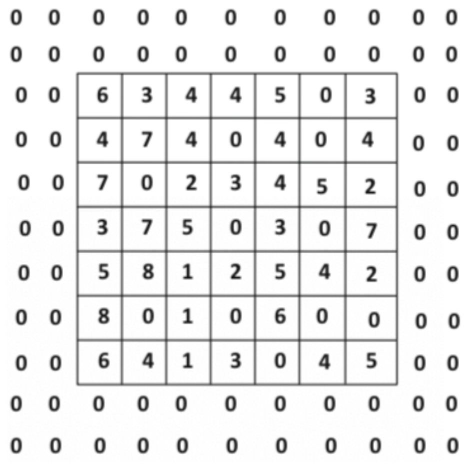

Figure 6: Zero-Padding over a a 2D input. Source [3]

2.4 Recurrent Neural Networks

Recurrent Neural Networks are a type of network which are especially good at dealing with sequences. This has helped to improve the performance in tasks like speech recognition, machine translation, time series forecasting, etc. A simple layer of a RNN has a weight matrix A which takes an input vector x_t and outputs h_t, often named as hidden state. In this case, the hidden state enters again by multiplying A.

So the output h_{t+1} depends on both the input x_{t+1} and the previous hidden state h_t. In the next figure we can see the folded (left) and the unfolded (right) representation of a RNN.

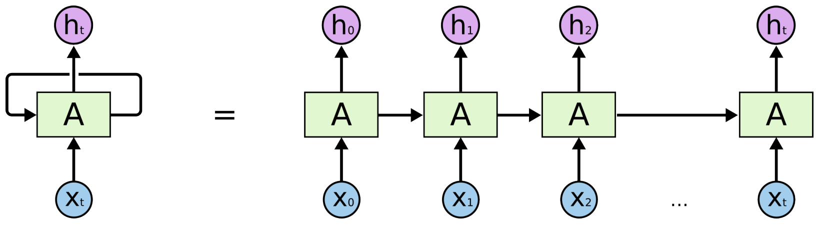

Figure 7: RNN. Source [51]

The problem of vanilla RNNs are the difficulties it has to deal with long-term dependencies. So, for a long sequence such as a long sentence, the hidden state at the last step time step will carry little information about the beggining of the sentence, which in some cases can hurt the performance of the model. To solve this issue, a new variant named the Long Short Term Memory (LSTM) network was proposed by Hochreiter, S. and Schmidhuber, J in [33] and has been highly adopted in NLP.

2.4.1 LSTM Networks

LSTM networks are a type of RNN that are constructed to deal better with long-term dependencies by providing a more complex architecture that allows information from previous time steps flows through the network without losses. In the Figure 8 it is shown the elements of a LSTM cell.

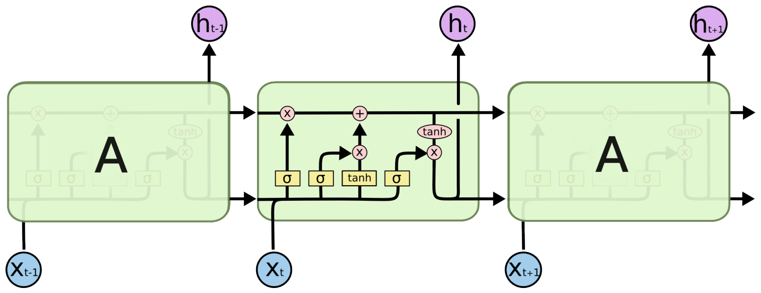

Figure 8: LSTM Network. Source [51]

Each LSTM cell is composed of four layers, the yellow squares represent the weights matrices with an activation function, which are sigmoid σ and tanh functions, and the pink circles represent pointwise operations. Starting from the left, the *forget gate* (1) rules how much information comes into the cell, f_t values close to 0 means that no information comes from the previous cell output h_{t-1} and values close to 1 means that almost all of the information is let pass.

Next, through the *input gate* (2) we get the information h_{t-1} and x_t, which are concatenated. This i_t values are compared with the candidates $\tilde{C}_t$ (3), which are calculated through the *tanh* layer, to choose which values are added to the new cell state C_t. Then the cell state C_t (4) is updated by multiplying the previous cell state by f_t and adding $i_t * \tilde{C}_t$. Finally, the hidden state h_t (6) is obtained from the multiplication of the *tanh* transformation of C_t and the output of the *sigmoid gate* o_t (5). Here is the mathematical formulation of the LSTM:

$$\begin{aligned} f_t &= \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) && (1)\\ i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) && (2)\\ \tilde{C}_t &= tanh(W_C \cdot [h_{t-1}, x_t] + b_C) && (3)\\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t && (4)\\ o_t &= \sigma(W_o[h_{t-1}, x_t] + b_o) && (5)\\ h_t &= o_t * tanh(C_t) && (6) \end{aligned}$$

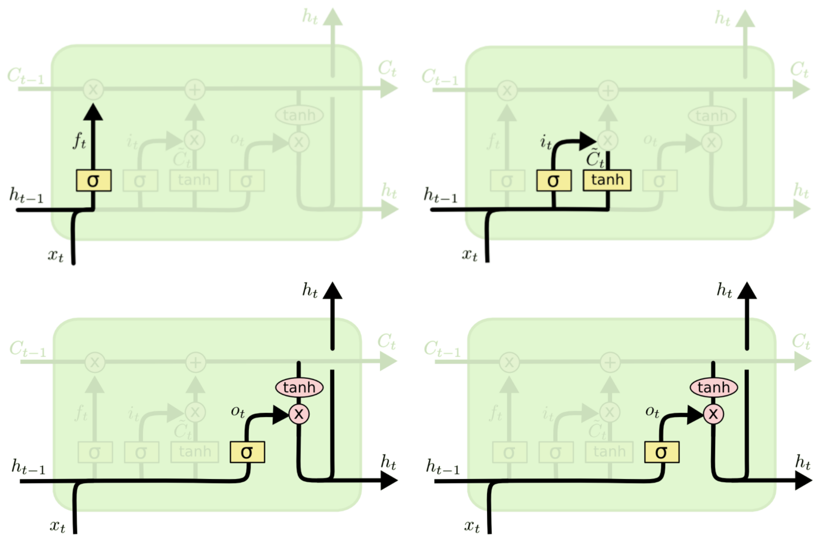

Figure 9: Steps of a LSTM. Source [51]

As we will see in Section 2.6 LSTMs have been used to get word representations based on the context, which has helped the improvement in Natural Language Processing tasks.

2.5 Computer Vision

The field where machines try to get an understanding of visual data is known as Computer Vision (CV). One of the most well-known tasks in CV is image classification. In 2010 The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) was introduced, a competition that dealt with a 1.2 million images dataset belonging to 1000 classes. In 2012 appeared the first CNN-based model that significantly reduced the error rate (AlexNet [40]), setting the beginning of the explosion of deep neural networks. From then onwards, deeper networks have become the norm.

2.5.1 CNNs for Image Classification

To illustrate how convolution layers are stacked to form an image classification system we show the VGG-16 architecture (Figure 10), which is a relatively simple neural network that won the first place in the ImageNet Challenge 2014. It is made of a succession of convolution layers with ReLU activation and max-pooling operations on top of three fully-connected layers and a final softmax layer. The key point when introducing this model was the use of filters with a relatively small receptive field (3×3). Instead, they played with the depth of the architecture, increasing it up to 19 layers. With this, they demonstrated that increasing the depth of the model is beneficial for the classification accuracy.

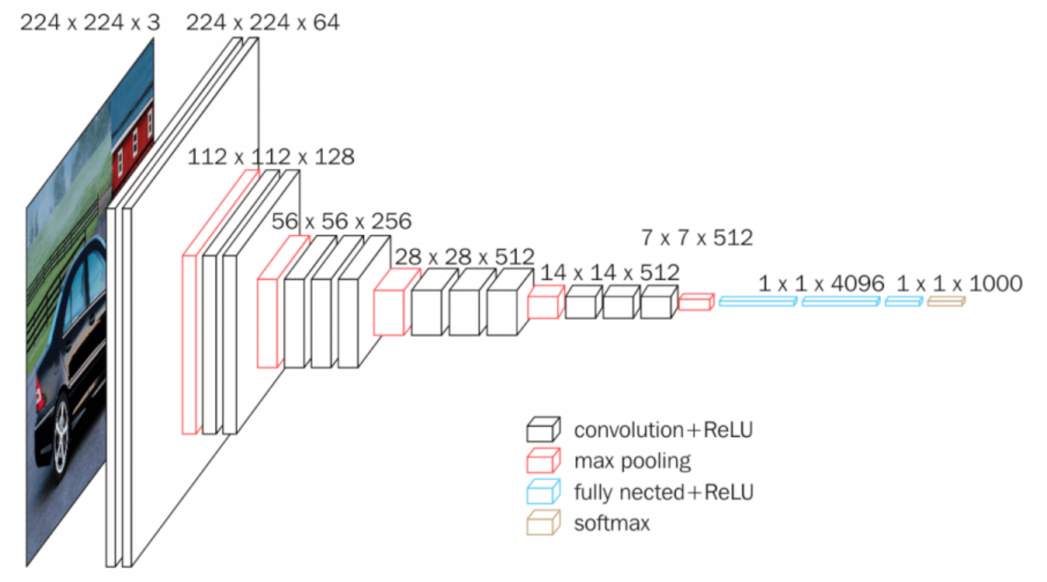

Figure 10: VGG-16 architecture. Source [30]

The use of deeper neural networks led to training difficulties such as the *degradation* problem, where accuracy gets saturated and then degrades quickly when adding more layers to the network. To help mitigate this issue in 2015 were introduced the ResNet family [31], which was considered a big leap forward. ResNet family of models make use of shortcut connection (Figure 11) as a solution to the training difficulties imposed by deep networks.

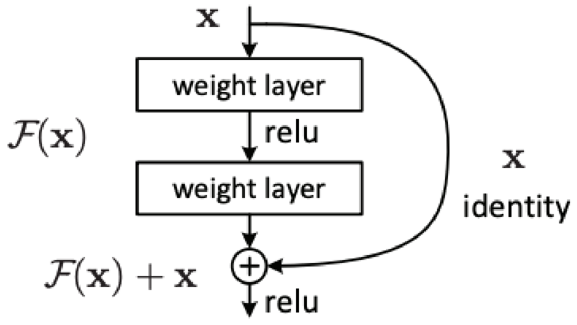
x
weight layer
relu
weight layer
$\mathcal{F}(\mathbf{x})$
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 11: Residual connection. Source [31]

The residual learning framework performs an addition between an identity mapping and the output of more than one layer. The authors hypothesize that it is easier to optimize the residual mapping $\mathcal{F}(x) + x$.

Further improvements have been done throughout the past years. These improvements have been focused on both increasing the accuracy of the image classifiers and trying to keep a good performance while reducing the size of the models. One example of an architecture developed with this latter direction in mind has been the family of MobileNets [54], which can be considered the base of the network we use in this work. Their main objective is the reduction of the network's number of parameters. One of the improvements in the second version of this architecture (MobileNet-V2) is the substitution of the classical 2D convolution by a depthwise and a pointwise operation allowing a lighter and faster model.

2.5.2 Transfer Learning in Computer Vision

Transfer learning can be defined as the improvement of learning in a new task through the transfer of knowledge from a related task that has already been learned. In Computer Vision, ImageNet database has served not only as a way of measuring model's performance but also as a manner of giving models fundamental knowledge about image recognition that has helped improve the results in other datasets.

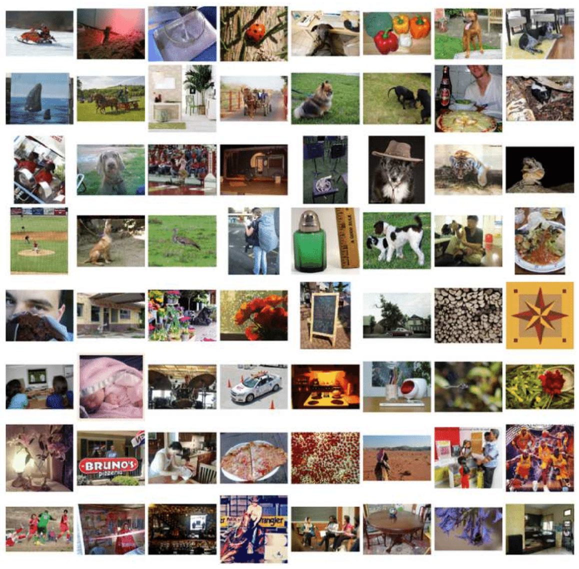

Figure 12: ImageNet sample. Source [22]

Pretrained convolutional neural network models can be easily downloaded. When dealing with additional data from the same application the level of transfer learning can be set. One can fine-tune a subset of the layers in the pretrained neural network with this additional data. If the amount of application-specific data available is low, the parameters of the early layers can be frozen to their pretrained values and the fine-tune can be applied only to the last layers of the model. These first layers usually contain primitive features, which are common to similar applications. For example, in a convolutional neural network, the first layers learn features like curves or edges, which can be useful for detecting any kind of object. However, the later layers contain more complex features which are more specific to the dataset properties. If the data available in the application-specific task is large it can be useful to fine-tune a large number of layers. So, deep neural networks are flexible in terms of how much transfer learning is applied from one task to another.

2.6 Natural Language Processing

The features learned from the OCR output are achieved by means of Natural Language Processing techniques. NLP is the field that deals with the understanding of human language by computers, which captures underlying meanings and relationships between words.

The way machines deal with words is by means of a real value vector representation. Word2Vec [48] showed that a vector could represent semantic and syntactic relationships between words. CoVe [45] introduced the concept of context-based embeddings, where the same word can have a different vector representation depending on the surrounding text. ELMo [52] followed CoVe but with a different training approach, by predicting the next word in a text sequence (Language Modeling), which made it possible to train on a large available text corpus. Depending on the task (such as text classification, named entity recognition...) the output of the model can be treated in different ways. Moreover, custom layers can be added to the features extracted by these NLP models. For instance, ULM-Fit [34] introduced a language model and a fine-tuning strategy to effectively adapt the model to various downstream tasks, which pushed transfer learning in the NLP field. Lately, the Transformer architecture [64] has dominated the scene, being the bidirectional Transformer encoder (BERT) [23] the one who established recently state-of-the-art results over several downstream tasks.

2.6.1 Word Embeddings

2.6.1.1 Neural Model

Given a text sequence of n words $w_{t-n+1}, \cdots, w_{t-1}$ which can be expressed as w_{t-n+1}^{t-1} where $w \in$ vocabulary V, a language model tries to learn a function $f(w_t, \cdots, w_{t-n+1})$ that maximizes the probability of the word w_t given the sequence w_{t-n+1}^{t-1}, $\hat{P}(w_t | w_{t-n+1}^{t-1})$, with the constraint that
$\sum_{i=1}^{|V|} f(i, w_{t-1}, \cdots, w_{t-n+1}) = 1$ for any w_{t-n+1}^{t-1}. The model can be divided in two parts: a mapping for every element i in the vocabulary V to a real vector $C(i) \in \mathbb{R}^m$ and a function g that outputs the conditional probability over words in V given the concatenation of real-valued vectors $C(w_{t-n+1}), \cdots, C(w_{t-1})$. The whole model can be described as:

$$f(i, w_{t-1}, \cdots, w_{t-n+1}) = g(i, C(w_{t-1}), \cdots, C(w_{t-n+1})))$$

The function g can be implemented by any parametrized function, such a feedforward neural network (feedforward NNLM). The mapping C is represented as a matrix $m \times \mid V \mid$ of weights where each column corresponds to the feature vector of a word in the vocabulary. The parameters of the C matrix are shared across every input word.

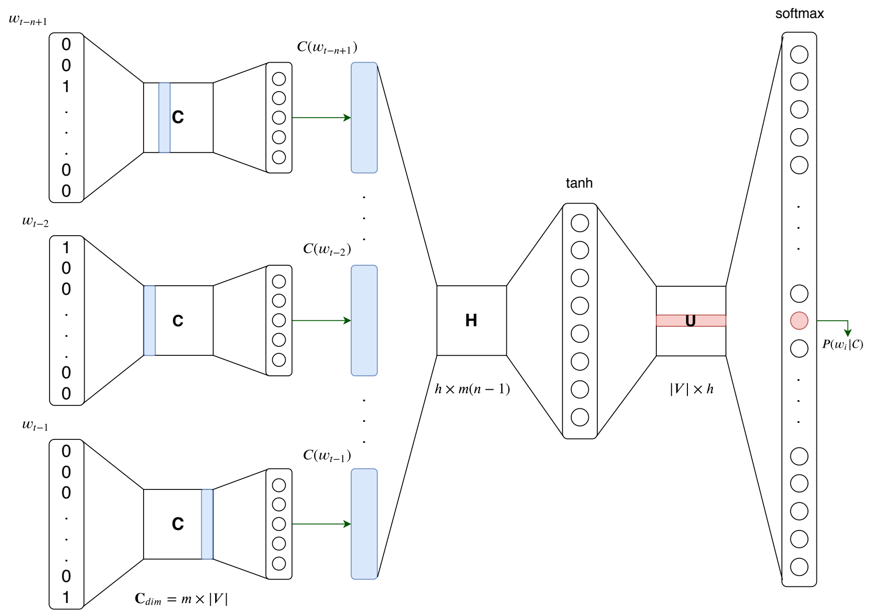

Figure 13: Feedforward Neural Network Language Model

2.6.1.2 Continuous Bag-of-Words (CBOW)

CBOW introduces a new method where past and future words are used. For window size m, the set $\mathcal{C}$ of context words of w_t is made of $w_{t-m}, \cdots, w_{t+m}$, excluding w_t. The words in the context are used to correctly classify the current (middle) word w_t. CBOW is similar to the feedforward NNLM but it gets rid of the non-linear hidden layer. Moreover, the projection layer C is now calculated as an average of the embedded word vectors of the context $\bar{C} = C(w_{t-m}), \cdots, C(w_{t+m})$, so the order of appearance in the sequence is not taken into account, that is why it received the name of bag of words. In this case, parameter matrices C and U are the ones to be learned during the training process.

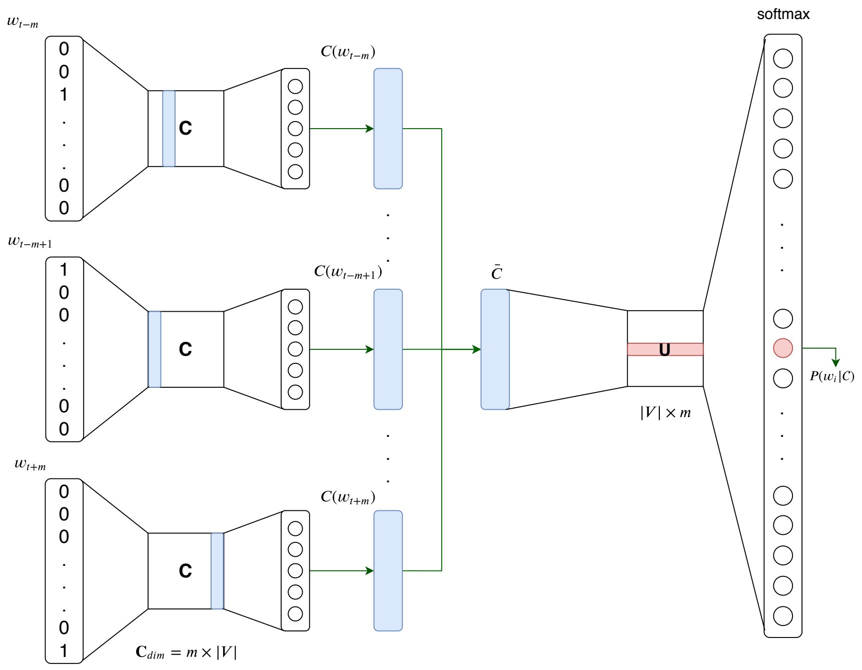

Figure 14: Continuous Bag-of-Words (CBOW) model

2.6.1.3 Continuous Skip-Gram Model

The skip-gram model tries to predict each context word separately, given the center word. The one-hot encoding of the center word is used as input and the one-hot encodings of the context words are used as correct labels. So, given a text sequence and a window size, the skip-gram model is trained by taking all possible center word w_t and 'context' words pairs inside that window-size.

A softmax activation function is applied to the output layer to give a probability distribution. Each of the output layer neurons gives the probability of each of the words in the vocabulary to be the one in the same context as the input word. So, a total number of $| V |$ neurons are used in the output layer.

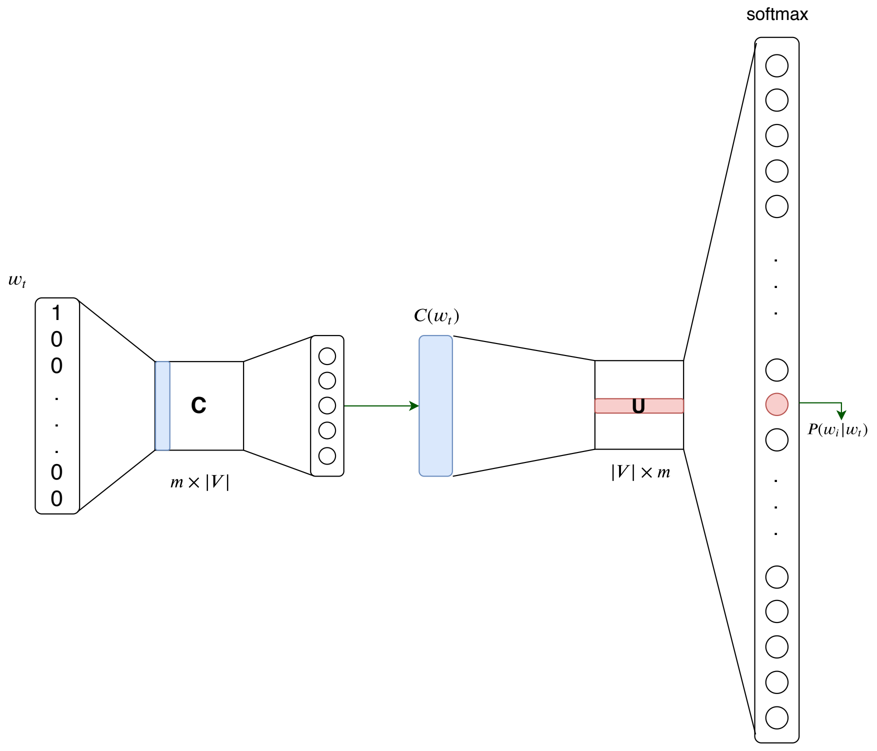

Figure 15: Continuous Skip-Gram model

A common loss function used to measure the similarity between two probability distributions, the predicted probabilities $\hat{y}$ and the ground truth y is the cross-entropy function, that in the discrete setting looks like $H(\hat{y}, y) = -\sum_{v \in \mathcal{V}} p(y) \log p(\hat{y})$. In our case the probability y is zero for every word in $\mathcal{V}$ except for the center word w_t.

$$\begin{aligned}
\mathcal{L}_\theta &= -\sum_{i=1}^{|V|} P(y_i) \log P(\hat{y}_i) \\
&= -\log P(\hat{y}_t) \\
&= -\log P(w_t|\mathcal{C}) \\
&= -\log P(u_t|\bar{C})) \\
&= -\log \frac{exp(u_t \bar{C})}{\sum_{j=1}^{|V|} exp(u_j^T \bar{C})} \\
&= -u_c^T \bar{C} + \log \sum_{j=1}^{|V|} exp(u_j^T \bar{C})
\end{aligned}$$

The weights of the hidden layer represent the embedding for each of the words in the vocabulary. Once trained, C matrix can be used as a look up table for other NLP tasks.

2.6.2 Sequence to sequence learning

Ilya Sutskever et. al [60] proposed a model that learned to map sequences to sequences by means of RNNs. This paper started a new way of training models by making use of the translation task. Machine translation is a subfield of NLP where models try to translate from a source language to a target language. Initially, statistical machine translation models where used, but lately it has been neural machine translation (NMT) the predominant type of architectures. A neural machine translation system is a neural network that directly models the conditional probability $p(y|x)$ of translating a source sentence, $x_1, ..., x_n$, to a target sequence, $y_1, ..., y_m$. NMT basic models are made of an encoder and a decoder. The encoder calculates a representation of the input sequence (s) and the decoder generates a sequence based on the encoder representations and the previous target words. Typically, the encoder and decoder are RNNs.

The decoder defines the probability over the target sentence y as:

$$p(y) = \prod_{j=1}^{m} p(y_j|y_{<j}, s)$$

The conditional probability of each target word can be calculated from a RNN decoder as:

$$p(y_t|y_{<t}, s) = \text{softmax}(g(h_j))$$

where g is a transformation function that outputs a vector of the size of the target language vocabulary, and h_j is the j_{th} hidden state of the decoder RNN, which is calculated by taking as input h_{j-1} and the encoder representation s, $h_j = f(h_{j-1}, s)$. Bahdanau *et al.* [8] proposed a model where the encoder representation is checked in every time step but other implementations have used it only to initialize the decoder RNN. The training objective for these models is:

$$J = \sum_{x,y \in \mathcal{D}} -\log p(y|x))$$

being $\mathcal{D}$ the parallel corpus.

Some lately models have used the attention mechanism between the encoder and decoder. These models compute a context vector for every time step c_t based on the hidden states of the encoder, which is concatenated with the decoder hidden state h_t to give the attentional vector $\tilde{h}_t$, calculated as:

$$\tilde{h}_t = \tanh(W_c[c_t; h_t])$$

The context vector c_t is calculated as a weighted average of the hidden states of the encoder $c_t = \sum_{s=1}^{n} \alpha_t \overline{h}_s$ This weights $\alpha_t(s)$ are computed by comparing the target hidden state h_t and the source hidden state h_s

$$\alpha_t(s) = \text{align}(h_t, \overline{h}_s)$$
$$= \frac{\exp(\text{score}(h_t, \overline{h}_s))}{\sum_s \exp(\text{score}(h_t, \overline{h}_s))}$$

where the score function can be for instance the dot product between h_t and $\overline{h}_s$.

Finally the attentional vector $\tilde{h}_t$ is passed through a linear transformation g together with softmax layer

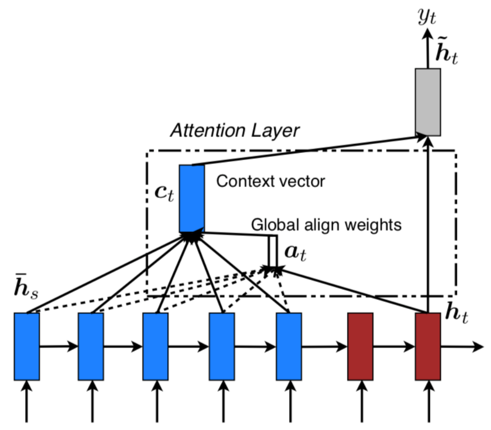

Figure 16: Attention mechanism from [44].

2.6.2.1 CoVe

Machine translation was used by McCan *et al.* to give context information to word vector representations [45]. These word embeddings are learned by means of a two-layer bidirectional LSTM, which the authors called MT-LSTM. MT-LSTM is used as the encoder that will later be used to transfer the knowledge learned into other downstream tasks.

Firstly, GloVe vectors of the source language words $w^x = [w_1^x, ..., w_n^x]$ are used as input to MT-LSTM, which computes a sequence of hidden states, one for each input word

$$h = \text{MT-LSTM}(\text{GloVe}(w^x))$$

These hidden states are the new learned words vectors CoVe(w^x).

To be more precise, the MT-LSTM produces $h_t = [\overrightarrow{h}_t; \overleftarrow{h}_t]$, $\overrightarrow{h}_t = \text{LSTM}(x_t, \overrightarrow{h}_{t-1})$ and $\overleftarrow{h}_t = \text{LSTM}(x_t, \overleftarrow{h}_{t-1})$. x_t represents the GloVe(w^x) vector.

During the training phase, an attentional decoder is placed on top of the encoder producing a conditional distribution over the target language words $p(w_t^z \mid H, w_1^z, ..., w_{t-1}^z]$, where H represents the a stack of hidden states for each input word. This decoder is made by a two-layer uni-directional LSTM which gives a hidden state h_t^{dec} for each time step calculated as

$$h_t^{dec} = \text{LSTM}([z_{t-1}, \tilde{h}_{t-1}], h_{t-1}^{dec})$$

Finally, the decoder computes a vector of weights $\alpha = \text{softmax}(H(W_1 h_t^{dec}) + b_1)$, which are used in an attentional sum concatenated with the decoder state and passed through a tanh activation function giving the context-adjusted hidden state $\tilde{h}$

$$\tilde{h}_t = \tanh(H(W_2 H^\top \alpha_t) + b_2; h_t^{dec})$$

The distribution over the target language words $p(w_t^z \mid H, w_1^z, ..., w_{t-1}^x]$ is then generated by applying a softmax function to the transformation $W_{out}\tilde{h}_t + b_{out}$.

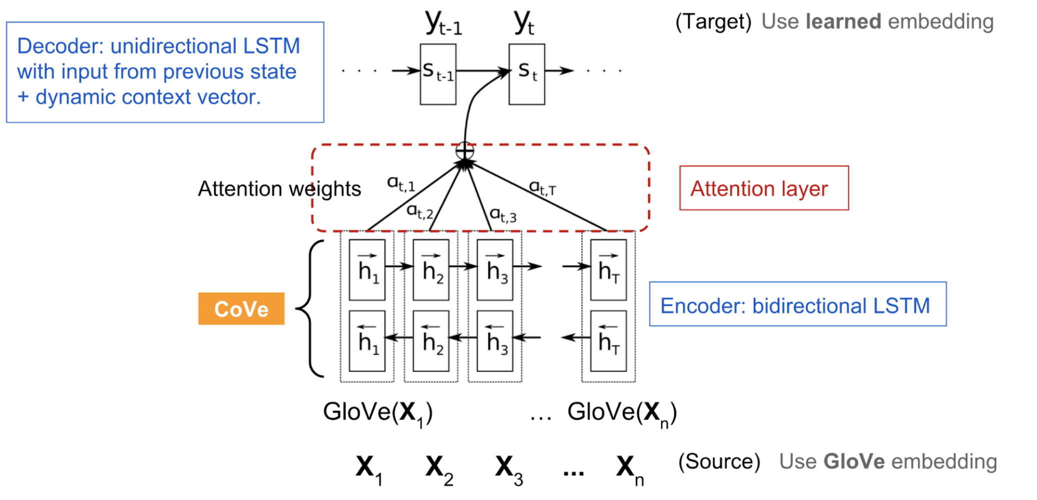
Decoder: unidirectional LSTM with input from previous state + dynamic context vector.
y_{t-1}
y_t
s_{t-1}
s_t
(Target) Use learned embedding
Attention weights
$\alpha_{t,1}$
$\alpha_{t,2}$
$\alpha_{t,3}$
$\alpha_{t,T}$
Attention layer
CoVe
Encoder: bidirectional LSTM
GloVe(x_1)
... GloVe(x_n)
x_1 x_2 x_3 ... x_n
(Source) Use GloVe embedding

Figure 17: CoVe architecture. Source [66]

2.6.3 Language Models

Language modeling is a task where models try to predict the next word in a sequence. This task allows training models without the need of having labels, since the next word in the context acts as a label.

2.6.3.1 ELMo

Embeddings from Language Model (ELMo) [52] continued the concept of context-based embeddings where the same word can have different vector representations depending on the surrounding text. ELMo model is trained by predicting a word in a text sequence, which makes it possible to be trained on large available text corpus in a self-supervised way. In this case, it uses words before and after the target word. Word representations are obtained by means of a linear combination of the internal states of the network. It has a parameter that changes the result of this linear combination depending on the task it performs (text classification, named entity recognition...).

ELMo uses a two-layer bidirectonal LSTM (bi-LM), each bi-LSTM layer j ouputs hidden states $h_{k,j}^{LM} = [\overrightarrow{h}_{k,j}^{LM}; \overleftarrow{h}_{k,j}^{LM}]$ for the k_{th} token, being $j = 0$ the token embedding layer and $j = L$ the top layer. ELMo k_{th} token vector representation for a specific $task$ is computed as follows:

$$\text{ELMo}_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_j^{task} h_{k,j}^{LM}$$

where s_j^{task} is a normalized-softmax weight vector and γ^{task} allows scaling the ELMo vector.

2.6.3.2 ULM-Fit

ULM-Fit (Universal Language Model Fine-tuning) [34] a fine-tuning procedure to achieve transfer learning in NLP task. Specifically, Howard *et al.* propose discriminative fine-tuning, slanted triangular learning rates, and gradual unfreezing techniques. In their experiments the use the AWD-LSTM model [47], a regular LSTM which together with the proposed pre-training methods is sufficient to beat the previous state-of-the-art in several text classification datasets. We use slanted triangular learning rates in the training process of the image model, which we talk about in Section 3.3.2, and discriminative fine-tuning when training the text model in Section 3.3.4.

2.6.4 Transformer-based models

The Transformer architecture was introduced in [64]. It is based on a stack of encoders and another stack of decoders, as in a seq2seq model. However, each encoder layer is made up of a self-attention layer and a feed-forward network and each decoder layer has this same two blocks but a masked attention layer at the bottom as well.

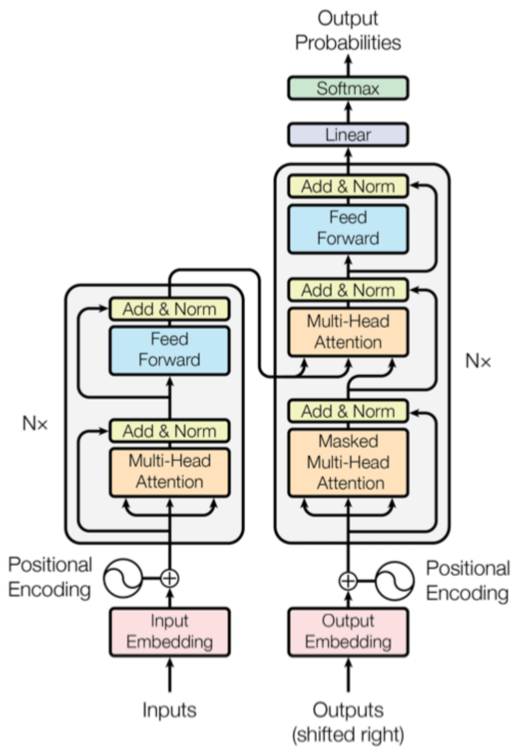

Figure 18: The Transformer architecture. Source [64]

The Transformer model is trained in the machine translation task, where inputs are sequences of words of the source language and the outputs are the translation into a target language.

The encoder receives as input either a word embedding plus a positional embedding, in the case of the first layer (gives information about the order of the tokens in the sequence), or the output of the previous encoder. In every case, this input is passed firstly through a self-attention layer, which does the following operation:

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

Each matrix Query (Q), a Key (K), and a Value (V) are created by multiplying the embedding matrix (stack of token embeddings) or previous encoder output matrix (stack of previous encoder output vectors) by three matrices that are trained during the training process, W^Q,W^K and W^V. These Q, K and V can be seen as matrices were each row represent the q, k and v vector of each token.

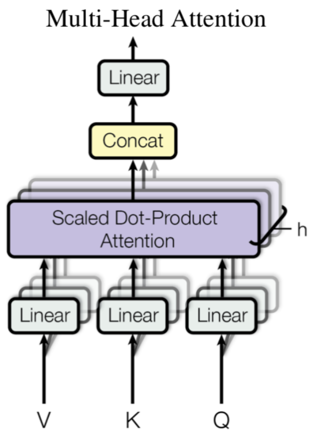

Figure 19: Multi-head attention layer. Source [64]

For better understanding we can simplify this by calculating the attention of the i-th token Attention_i by using its vector q_i. The self-attention calculation in this case is:

$$\text{Attention}_i(q_i, K, V) = \text{softmax}\frac{q_i K^T}{d_k} V$$

A score between the input token of the sequence and every other is calculated. This score represents how much focus to put on other parts of the sequence when encoding a specific token (i in this case). So, a dot product is calculated between the q_i vector and the Key vector of every other object in the sequence (the intuition is that this is a similarity operation, we try to find the key similar to the queries and a high value to them), which is equivalent to multiplying q_i by the matrix K^T, giving as many scores as elements in the sequence. Then this output of scalars are divided by the square root of the dimension of the key vectors d_k (for getting more stable gradients) and finally it is passed through a softmax operation. Once we have a normalized score for every pair of tokens, we multiply this score by the value vector of each of the tokens (each row in matrix V) and finally sum up those vectors, generating one single vector z_i. So for each query we get one vector z_i.

Turning back to the matrix notation, the previous process can be done for the entire input sequence by using matrix Q (each row is a token q vector). So, the benefit of these matrices operations is that they can be parallelized. In this case we end up with a matrix Z with each row representing the self-attention vector result of each token.

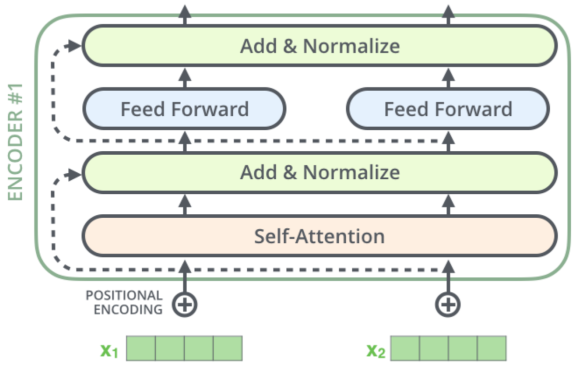

Figure 20: The Transformer encoder. Source [5]

This process is done in each of the self-attention heads, in the original transformer the model has 8 heads. So, in total 8 matrices are output. Each head has its own W^Q, W^K and W^V weight matrices which are randomly initialized, so the result leads to different representation subspaces in each of the self-attention heads.

The output matrix Z of every self-attention head are concatenated into a single matrix to which a linear transformation is applied,
$MultiHead = Concat(Z_{head_i}, \cdots, Z_{head_h})W^O$ as show in Figure 19. This linear projection is added to the initial embeddings entering the encoder and then a LayerNorm operation is performed.

On top of each encoder and decoder layer the transformer has a feed-forward network (FFN) which consists of two linear transformations with a ReLu activation function after the first transformation.

The decoder takes as input a double copy of the encoder output as K, V matrices. The first self-attention layer, also composed of 8 heads gets K, V and Q inputs from the output sequence but only from previously seen positions, the posterior ones are masked.

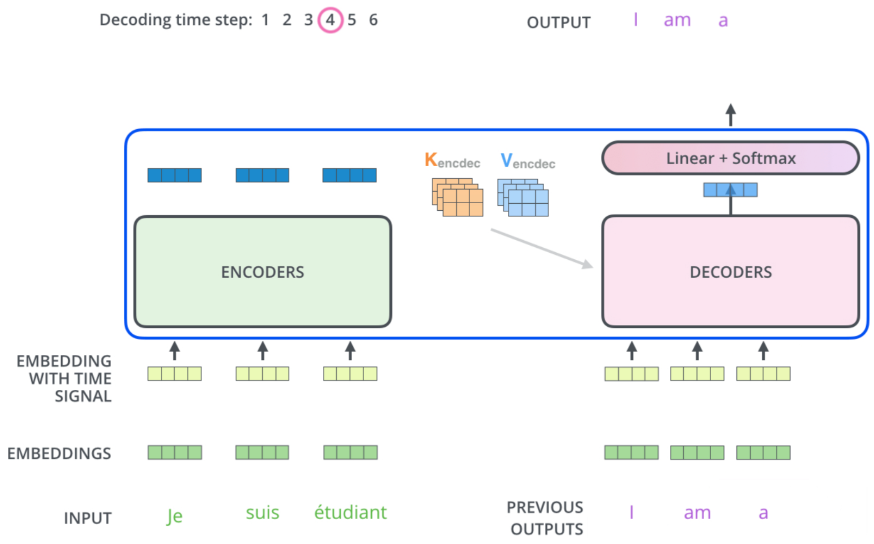

Figure 21: The Transformer decoder in action. Source [5]

The decoder outputs a vector of floats that is turned into a target language probability distribution by making a linear transformation and adding a softmax operation. The cross-entropy loss function is used compare the output probability distribution for each word and the ground-truth probability distribution.

2.6.5 BERT

BERT model [23] was presented in 2018 and it set a major breakthrough in NLP. This model is based on the encoder layer of the Transformer model, so to understand how BERT works we first needed to dive into the Transformer model.

BERT model gets rid of the decoder part of the Transformer, which leaves the architecture showed Figure 22, note that this is top-down view. Two model sizes were presented, BERT_BASE with 12 layers, 12 attention heads and hidden vectors $\in \mathbb{R}^{768}$ and BERT_LARGE with 24 layers, 16 attention heads and hidden vectors size $\in \mathbb{R}^{1024}$.

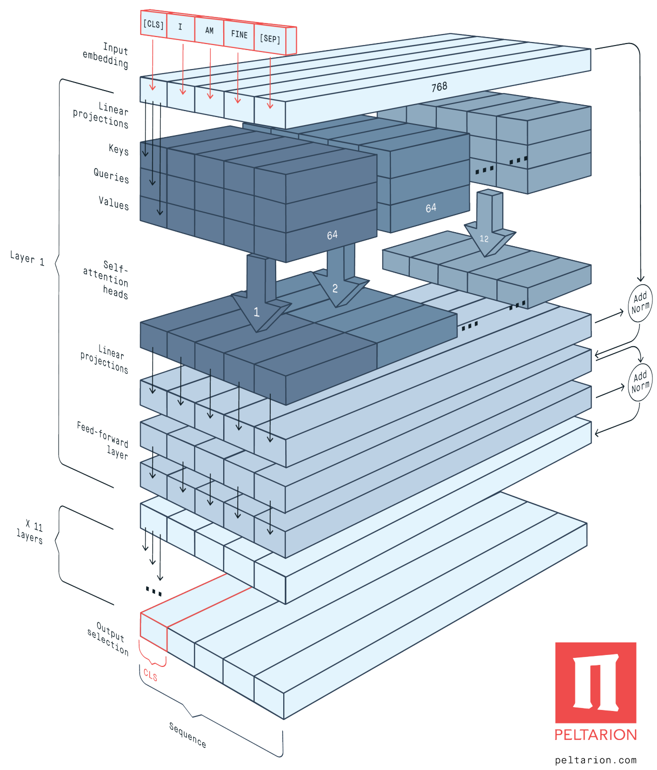

Figure 22: BERT architecture. Source [12]

BERT introduces a language model that is able to learn from both sides of the word it tries to predict, instead of a left-to-right model or a separate right-to-left and left-to-right training such in [52]. BERT objective is to introduce a model that can be fine-tuned into downstream tasks with minor changes to the architecture and starting with the pre-trained parameters.

BERT is pre-trained using two tasks: Masked LM and Next sentence prediction (NSP). In order to solve the problem in LM of the attention mechanism seeing the token it is trying to predict in a multilayered model BERT uses masks. The way it does it is by selecting 15% of the random tokens and masking with a 0.8 probability with the token [MASK]. NSP task is about predicting if a pair of sentences are consecutive. By choosing sentences A and B, 50% of the time B follows A and 50% of the time it does not. In this case a token [SEP] is added between the two tokenized sentences and a segment embedding is summed to the input tokens describing whether it is the first or the second sentence.

BookCorpus [68] and English Wikipedia are used to pre-train the model.

2.7 Related Work in Document Image Classification

Several ways of measuring models have been shown in the past years regarding document classification on the Legacy Tobacco Industry Documents [43]. Some authors have tested their models on a large-scale sample BigTobacco. Others tried on a smaller version named SmallTobacco, which could be seen as a more realistic scale of annotated data that users might be able to find. Lastly, transfer learning from in-domain datasets has been tested by using BigTobacco to pre-train the models to finally fine-tune on SmallTobacco. Table 2 summarizes the results of previous works in the different categories over time.

First results in the Deep Learning era have been mainly based on CNNs using transfer learning techniques. Multiple networks were trained on specific sections of the documents [29] to learn region-based high dimensional features later compressed via Principal Component Analysis (PCA). The use of multiple Deep Learning models was also exploited by Arindam Das *et al.* by using an ensemble as a meta-classifier [21]. A VGG-16 stack of networks using 5 different classifiers has been proposed [57], one of them trained on the full document and the others specifically over the header, footer, left body and right body. The Multi Layer Perceptron (MLP) was the ensemble that performed the better. A committee of models but with a SVM as the ensemble was also proposed [53].

Table 2: Previous results comparison (accuracy in %).

	BigTobacco	**SmallTobacco**			
		BigTobacco Pre-training		No Pre-training	
Author	Image	Image	Image + Text	Image	Image + Text
Kumar et al. (2014)[36]				43.8	
Kang et al. (2014)[38]				65.37	
Afzal et al. (2015)[1]				77.6	
Harley et al. (2015)[29]	89.8			79.9	
Csurka et al. (2016)[20]	90.7				
Noce et al. (2016)[50]					79.8
Afzal et al. (2017)[2]	90.97	91.13			
Tensmeyer et al. (2018)[63]	90.8				
Das et al. (2018)[21]	92.21				
Audebert et al. (2019)[7]				84.5	87.8
Asim et al. (2019)[6]		93.2[7]	95.8[8]		
Proposed work (2020)	**92.31**	**94.04**	94.9	**85.99**	**89.47**

The addition of content-based information has been investigated on SmallTobacco by extracting text through OCR and embedding the obtained features into the original document images as a previous phase to the training process [50]. Lately, a Mobilenetv2 architecture [54] together with a CNN 2D [39, 67] taking as input FastText embeddings [11, 37] have achieved the best results in SmallTobacco [7].

A study of several CNNs was carried out [2], where VGG-16 architecture [57] was found optimal. Afzal *et al.* also demonstrated that transfer learning from in-domain dataset like BigTobacco [2] increases by a large margin the results in SmallTobacco.

[7]Accuracy obtained in 9 classes that overlap in BigTobacco

[8]Evaluation method not specified

This was further investigated by adding content-based information with CNN 2D with ranking textual features (ACC2) to the OCR extracted.

As far as we are concerned, there is no study about the use of multiple GPUs in the training process for the task of Document Image Classification. However, parallelizing a computer vision task has been shown to work properly using ResNet-50, which is a widely used network that usually gives good results despite its low complexity architecture. Several training procedures are demonstrated to work effectively with this model [4, 27]. A learning rate value proportional to the batch size, warmup learning rate behavior, batch normalization, SGD to RMSProp optimizer transition are some of the techniques exposed in these works. A study of the distributed training methods using ResNet-50 architecture on an HPC cluster is shown in [14, 15]. To know more about the algorithms used in this field, look at [10].

3 Results

In this section we compare the performance of the different EfficientNets in SmallTobacco and BigTobacco as showed in Table 2 and demonstrate the benefits of the multiple GPU training. Experiments have been carried out using GPUs clusters Power-CTE[9] of the Barcelona Supercomputing Center - Centro Nacional de Supercomputación[10], each one composed by: 2 IBM Power9 8335-GTGH at 2.40GHz (20 cores and 4 threads/core), 512GB of main memory distributed in 16 dimms $\times$ 32GB at 2666MHz and 4 GPU NVIDIA V100 (Volta) with 16GB HBM2.

The operating system is RedHat Linux 7.4. The models and their training are implemented with PyTorch version 1.0 running on CUDA 10.1 and using cuDNN 7.6.4.

The only modification done to the images is a resize to 384 $\times$ 384 as explained in Section 3.3.1 and, in order to avoid overfitting, a shear transformation of an angle $\theta \in [-5°, 5°]$ [63] which is randomly applied in the training phase. No other modifications are used in our experiments. Source code is at https://github.com/javiferran/document-classification.

3.1 Evaluation

In order to compare with previous results in SmallTobacco dataset, we divide the dataset following the procedure in [36]. Documents are split into training, test and validation sets, containing 800, 2482 and 200 samples each one. 10 different splits of the dataset are created by randomly sampling from the 3482 documents so that 100 samples per class are guaranteed between train and validation sets. In the Figure 30 we give the accuracy on SmallTobacco as the median over the 10 dataset splits to compare with previous results. Accuracy on BigTobacco is shown as the one achieved on the test set. BigTobacco dataset used in Section 3.5 is slightly modified, where overlapping documents with SmallTobacco are extracted. Top performing model's accuracies are written down in Table 2.

3.2 Early Results

In order to iterate fast and get an intuition about how to solve the proposed problem we first made use of the SmallTobacco dataset. A few different image and text models were tried. In the text side, we started with FLAIR library embeddings. As explained in Section 2.6.1, embeddings are the way computers deal with words. FLAIR offers a wide variety of word embeddings:

Moreover, FLAIR library allows combining several embeddings with its *StackedEmbedding* functionality. Basically, it concatenates the values of the embeddings for each

[9]https://www.bsc.es/support/POWER_CTE-ug.pdf
[10]https://www.bsc.es

Class	**Type**	**Paper**
BertEmbeddings	Embeddings from pretrained BERT	[23]
BytePairEmbeddings	Subword-level word embeddings	Heinzerling and Strube (2018)
CharacterEmbeddings	Task-trained character-level embeddings	Lample et al. (2016)
ELMoEmbeddings	Contextualized word-level embeddings	[52]
FastTextEmbeddings	Word embeddings with subword features	Bojanowski et al. (2017)
FlairEmbeddings	Contextualized character-level embeddings	Akbik et al. (2018)
PooledFlairEmbeddings	Pooled variant of FlairEmbeddings	Akbik et al. (2019)
OneHotEmbeddings	Standard one-hot embeddings of text or tags	-
OpenAIGPTEmbeddings	Embeddings from pretrained OpenAIGPT	Radford et al. (2018)
RoBERTaEmbeddings	Embeddings from RoBERTa	Liu et al. (2019)
TransformerXLEmbeddings	Embeddings from pretrained transformer-XL	Dai et al. (2019)
WordEmbeddings	Classic word embeddings	
XLNetEmbeddings	Embeddings from pretrained XLNet	Yang et al. (2019)
XLMEmbeddings	Embeddings from pretrained XLM	Lample and Conneau (2019)

of the words into a larger vector. With these options, the possibilities are almost endless.

Given the embeddings of a sequence (document in our case) one can concatenate them into a single vector and connect it to an MLP or stack the embeddings as a matrix and perform more complex operations like in the case of convolutions over the matrix as in Figure 23. This method was successfully proved in [39] and works by applying a CNN 2D operation with different filter sizes. Each filter slides in a one-way direction, only through the rows of the embedding matrix. The width of the filter is determined by the word embedding vector length and the height is left as a hyperparameter. To the feature maps obtained from the convolution operation we apply a max-pooling operation. Then, these values are concatenated into a single vector and passed to a fully-connected layer with a softmax function.

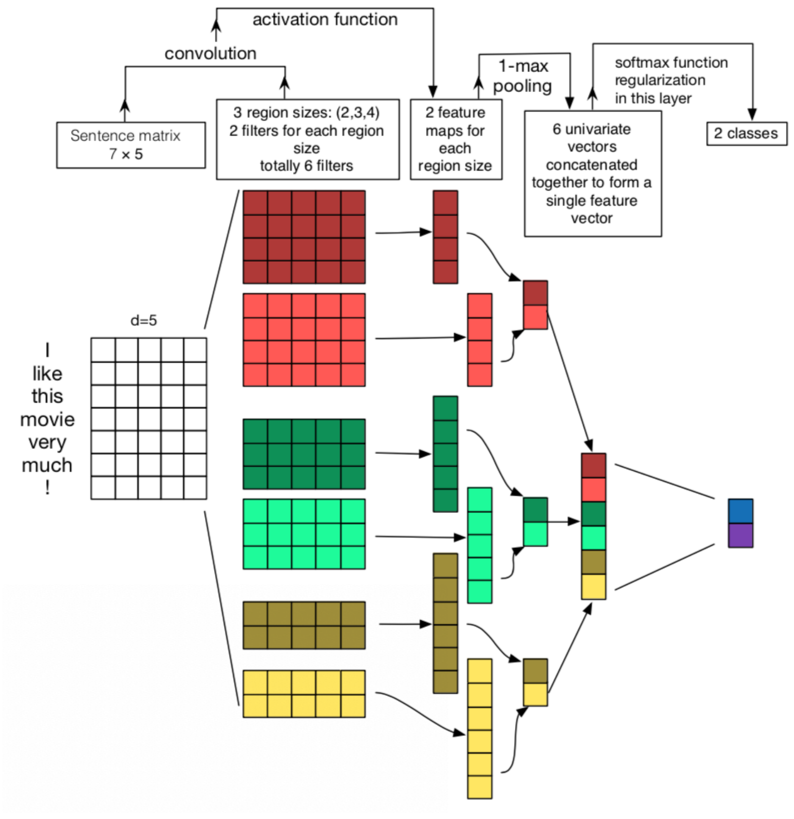
activation function
convolution
Sentence matrix
7 × 5
3 region sizes: (2,3,4)
2 filters for each region size
totally 6 filters
2 feature maps for each region size
1-max pooling
6 univariate vectors concatenated together to form a single feature vector
softmax function regularization in this layer
2 classes
d=5
I
like
this
movie
very
much
!

Figure 23: CNN operation on the sequence embedding matrix [67]

In this work we carry out multiple experiments with this embedding + CNN 2D configuration, some of the results obtained are shown in Figure 3. Moreover, FLAIR allows obtaining a single embedding per document (*document embedding*). It is calculated based on a word embedding type. It applies a pooling operation (max, min or mean) over the whole document word embeddings. Note that in the table we show the document embeddings results using Flair forward embeddings.

Embedding	**Accuracy (%)**
Fast	77,07%
Byte	70,78%
FastText + Byte	64.7%
Flair forward	73,53%
Flair forward + backward	75,66%
Forward + Fast	78.2%
Forward + Backward + Fast	77.7%
Glove	67,10%
BERT	**80.3%**
Document embedding (max)	62,10%
Document embedding (min)	61,20%

Document embedding (mean)	69,60%

After testing some of the FLAIR embeddings, BERT outperforms any other embedding. To further investigate how BERT embedding can help improve the performance on this task we continue with the Huggingface's Transformers library and directly tune the model. The details of the implementation are explained in Section 3.3.4.

3.3 Proposed Approach

In this section we present the models used and a brief explanation of them. We also show the training procedure used in both BigTobacco and SmallTobacco and the pipeline of our approach to the problem.

3.3.1 Image model

EfficientNets [62] are a set of light CNNs designed to scale up in a structured manner. The network's width (w), depth (d) and resolution (r) are defined as: $w = \alpha^{\phi}$, $d = \beta^{\phi}$ and $r = \gamma^{\phi}$, where ϕ is the scaling compound coefficient. The optimization problem is set by constraining $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$ and $\alpha \geq 1, \beta \geq 1, \gamma \geq 1$.

By means of a grid search of α, β, γ with AutoML MNAS framework [61] and fixing $\phi = 1$, a baseline model (B0) is generated optimizing FLOPs and accuracy. Then, the baseline network is scaled up uniformly fixing α, β, γ and increasing ϕ. We find that scaling the resolution parameter as proposed in [62] does not improve the accuracy obtained. In our experiments in section 3 we proceed with an input image size of 384×384, which corresponds to a resolution $r = 1.71$, as proposed by Tensmeyer *et al.* in [63] with AlexNet architecture [40].

The main block of the EfficientNets is the mobile inverted bottleneck convolution [54, 61]. This block is formed by two linear bottlenecks connected through both a shortcut connection and an intermediate expansion layer with a depthwise separable convolution (3×3) [17]. Probabilities $P(class|FC)$ are obtained by applying the softmax function on top of the fully connected layer FC of the EfficientNet model.

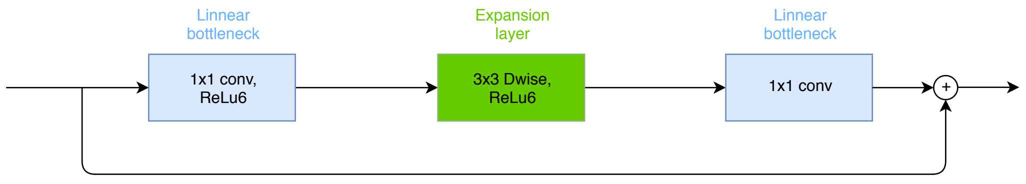

Figure 24: Mobile inverted bottleneck block.

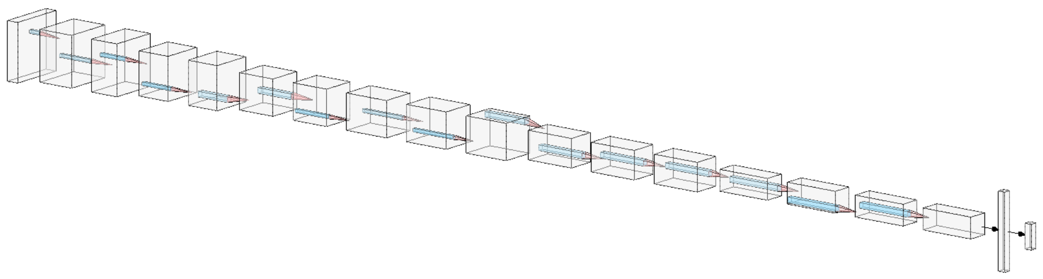

Figure 25: EfficientNet B0 architecture

3.3.2 Pre-training on BigTobacco

We train EfficientNets (pretrained previously on ImageNet) on BigTobacco using Stochastic Gradient Descent for 20 epochs with *Learning Rate Warmup* strategy [32], specifically we follow *STLR* (Slanted Triangular Learning Rate) [34] which linearly increases the learning rate at the beginning of the training process and linearly decreases it after a certain number of iterations. We chose the *reference learning rate* η following the formula proposed in [27] and used in [4] and [32]. Specifically, we set $\eta = 0.2 \cdot \frac{nk}{256}$, where k denotes the number of workers (GPUs) and n the number of samples per worker (4). Figure 27 shows the multi-GPU training procedure to get $\text{EfficientNet}_{\text{BigTobacco}}$, which represents EfficientNet model pretrained on BigTobacco. EfficientNet is loaded with ImageNet weights ($\text{EfficientNet}_{\text{ImageNet}}$) and then located in different GPUs within the same node.

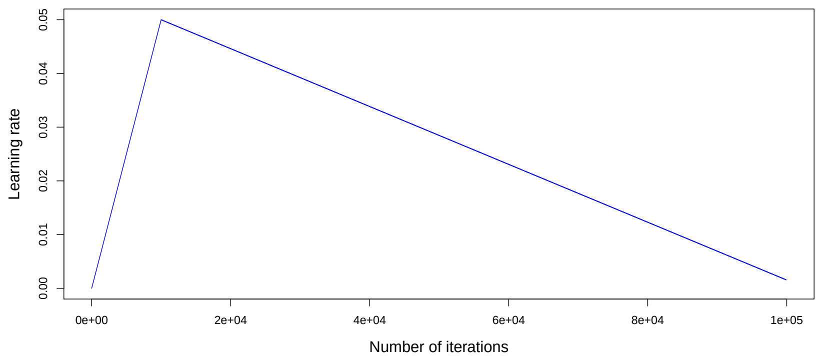

Figure 26: Slanted Triangular learning rate

3.3.3 Fine-tuning on SmallTobacco

We fine-tune on SmallTobacco the pre-trained models by freezing the entire network but the last softmax layer. Just 5 epochs are enough to get the peak of accuracy. *STLR* is used this time with $\eta = 0.8 \cdot \frac{nk}{256}$. Since only the last layer is trained, there's no risk of catastrophic forgetting [46]. Final fine-tuned model is represented as $\text{EfficientNet}_{\text{BigTobacco}}$ in Figure 27.

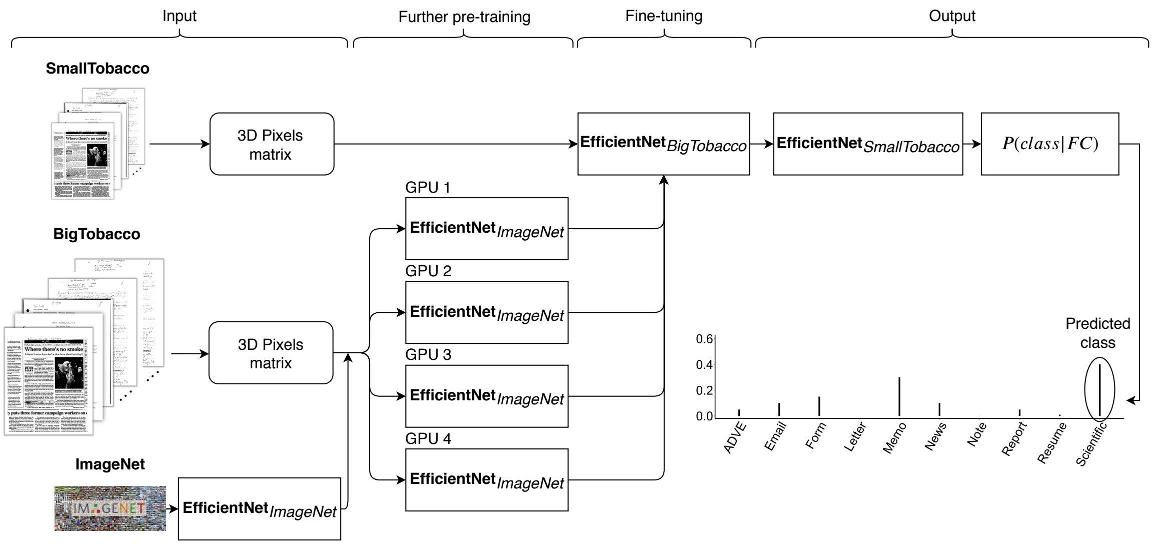

Figure 27: Pipeline of the different stages of the pre-training of EfficientNet over multiple GPUs.

3.3.4 Text model

Predictions from OCR Tesseract are obtained by means of the BERT model [23]. BERT is a multi-layer bidirectional Transformer encoder model pre-trained on a large corpus. In this work we use a modification of the original pre-trained BERT$_{\text{BASE}}$ version. In our case, we reduce to 6 the number of BERT layers since we find less variance in the final results and faster training/inference times. The output vector size is kept to 768. The maximum length of the input sequence is set to 512 tokens. The first token of the sequence is defined as $[CLS]$, while $[SEP]$ is the token used at the end of each sequence. Since some documents are larger than the 512 token's limit, we have to leave out of the model some of the text in the documents. In this case we select the first 128 tokens and the last 382, as proposed in [59].

A fully connected layer is added to the final hidden state of the $[CLS]$ token $h_{[CLS]}$ of the BERT model, which is a representation of the whole sequence. Then, a softmax operation is performed giving $P(class|h_{[CLS]})$ the probabilities of the output vector $h_{[CLS]}$, i.e the whole input sequence, pertaining to a certain *class*.

The training strategies used in this paper are similar to the ones proposed in [65, 59]. We use a learning rate $\eta_B = 3e^{-5}$ for the embedding, pooling and encoder layers while a custom learning rate $\eta_C = 1e^{-6}$ for the layers on top of the BERT model. A decay factor $\xi = 1e^{-8}$ is used to reduce gradually the learning rate along the layers, $\eta^l = \xi \cdot \eta^{l-1}$. ADAM optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and L_2-weight decay factor of 0.01 is used. The dropout probability is set at 0.2. Just 5 epochs are enough to find the peak of accuracy with a batch size of 6, the maximum we could use due to memory constraints.

3.3.5 Image and Text ensemble

In order to get the final enhanced prediction of the combination of both text and image model we use a simple ensemble as in [6].

$$P(class|out_{image}, out_{text}) = w_1 \cdot P(class|h_{[CLS]}) + w_2 \cdot P(class|FC)$$

$$Predicted\ Class = \arg\max_{class}(P(class|out_{image}, out_{text}))$$

In this work $w_1, w_2 = 0.5$ are found optimal. These parameters could be found by a grid search where $\sum_{i=1}^{N} w_i = 1$, being N the number of models. This procedure shows to be an effective solution when both models have similar accuracy and it allows us to avoid another training phase [7]. In Figure 28 this whole process is depicted.

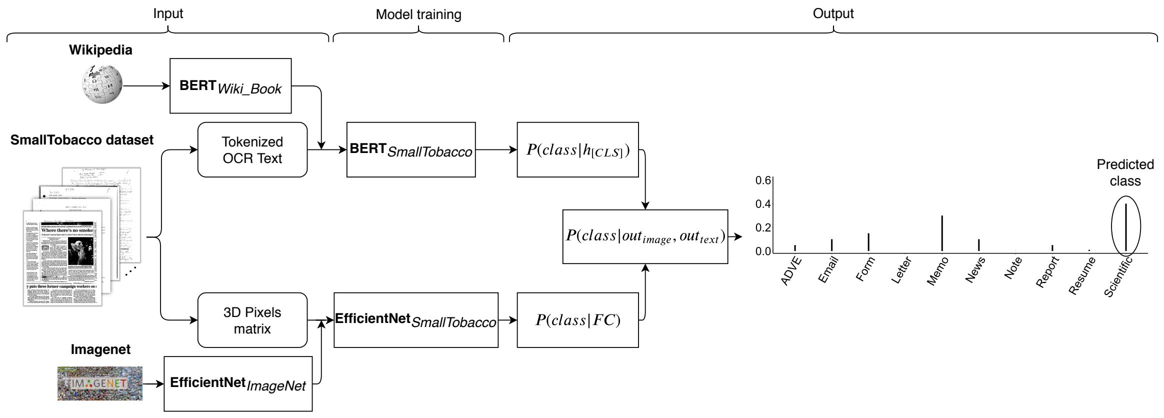

Figure 28: Pipeline of the proposed multimodal approach.

3.4 Results on BigTobacco

The best performing model in BigTobacco dataset is EfficienNet-B4 with 92.31% accuracy in the test set. Although it beats the previous state-of-the-art just for a bit, the main difference is that we use one single model to do so while Das et al. [21] used an ensemble of 5 different networks.

Table 4: Previous results on BigTobacco (accuracy in %).

	BigTobacco
Author	Image
Harley et al. (2015)[29]	89.8
Csurka et al. (2016)[20]	90.7
Afzal et al. (2017)[2]	90.97
Tensmeyer et al. (2018)[63]	90.8
Das et al. (2018)[21]	92.21
Proposed work (2020)	**92.31**

Table 5: Time (hours) needed to train the EfficientNet models.

Model \ GPUs	1	2	3	4
B0	13.33	6.81	4.58	3.4
B1	19.44	9.81	6.81	4.94
B2	20.55	10.22	6.92	5.16
B3	25.64	12.94	8.78	6.55
B4	34.28	17.36	11.69	8.75

We show in Table 5 the time it takes to train the different networks while using 1, 2, 3 or 4 GPUs in a single node. In order to take advantage of the multiple GPUs we use data parallelism, which consists of placing a copy of the model in each of them. Since every GPU share parameters, it is equivalent to having a single GPU with larger batch size.

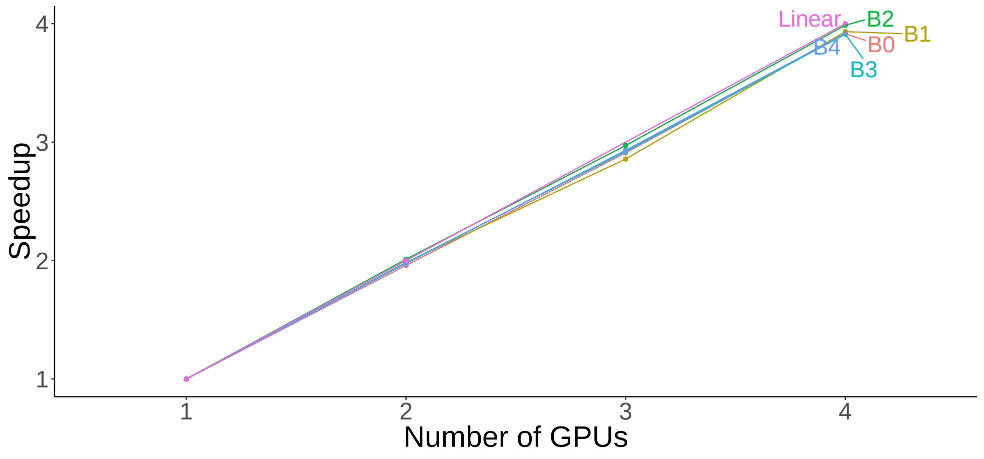

Figure 29: Speedup of the training process when parallelizing.

The time reduction to complete the entire training process with B0 variant is ≈ 61.14% lower when compared with B4 (4 GPUs). Time reduction by using multiple GPUs is clearly shown in Figure 29. For instance, EfficientNet-B0 benefits from a ≈ 75.4% time reduction after parallelizing over 4 GPUs.

Table 6: Previous results on SmallTobacco (accuracy in %).

	SmallTobacco		
	BigTobacco Pre-training	No Pre-training	
Author	Image	Image	Image + Text
Kumar et al. (2014)[36]		43.8	
Kang et al. (2014)[38]		65.37	
Afzal et al. (2015)[1]		77.6	
Harley et al. (2015)[29]		79.9	
Noce et al. (2016)[50]			79.8
Afzal et al. (2017)[2]	91.13		
Das et al. (2018)[21]		84.5	87.8
Audebert et al. (2019)[7]	93.2		
Proposed work (2020)	**94.04**	**85.99**	**89.47**

3.5 Results on SmallTobacco

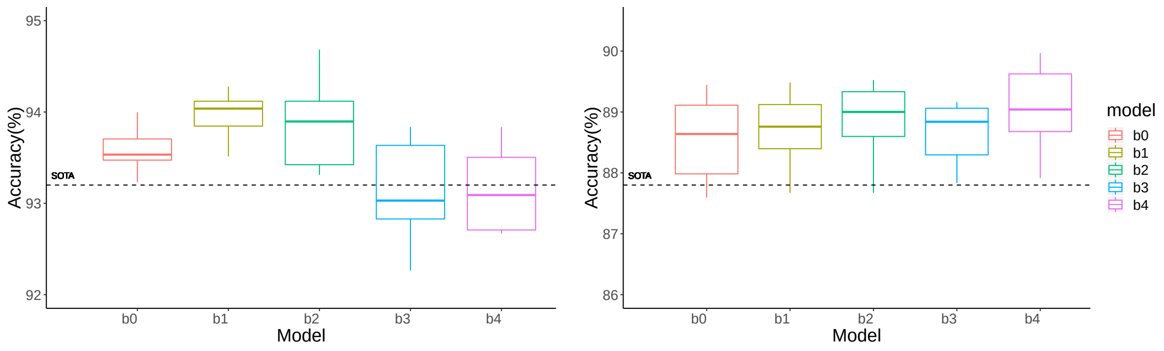

Figure 30: Accuracy obtained in SmallTobacco by models pre-trainined on BigTobacco (Left) and without BigTobacco pre-training (Right).

Accuracies of the EfficientNets pre-trained on BigTobacco and finally fine-tuned on SmallTobacco are depicted in the left plot of Figure 30. Simpler models perform with less variability between the 10 random splits than the heavier ones. The best performing model is the EfficientNet-B1, achieving a new state-of-art accuracy of 94.04% median over 10 splits.

In this work, we also wanted to test the potential of light EfficientNet models on a small dataset such as SmallTobacco without the use of transfer learning from in-domain dataset, and compared it with the previous state-of-the-art. Results given by our proposed method described in section 3.3.5 are shown in the right plot of Figure 30. Although we perform the tests over 10 different random splits to give a wider view of how these models work, in order to compare with *Audebert et al.* [7] we calculate the average over 3 random splits, which gives us an 89.47% accuracy.

Every ensemble model achieves better accuracy than previous results, and there is almost no difference between different EfficientNets results.

4 Parallel and Distributed Deep Learning

Single GPU training requires a huge amount of time, especially when dealing with heavy architectures. For this reason, experimenting with several workers is crucial to minimize the amount of time spent on these tasks. In order to try to minimize the time spent in training these models and taking advantage of the computational resources of the Barcelona Supercomputing Center - Centro Nacional de Supercomputación, we exploit the available libraries to accomplish this goal. We focus on PyTorch framework and its own APIs for making a parallel training in several GPUs by means of data parallelism PyTorch's Dataparallel and DistributedDataparallel.

4.1 Data Loading

Data, initially stored in the disk, is sent to the host, where changes from pageable to pinned memory. From pinned memory can then be transferred to the GPU. When trying to optimize the whole training procedure, an important step is to parallelize the data loading process so that it does not become a bottleneck. To do so, PyTorch allows set the number of processes in the CPU that participates in the data transfer from disk and also allows to using multi-threading in the pageable to pinned memory conversion. The key point here is that the whole data loading process takes less time than the operations done by the GPU with the previous batch of data. However, an excess number of workers will end up not being useless and wasting unnecessarily computational resources.

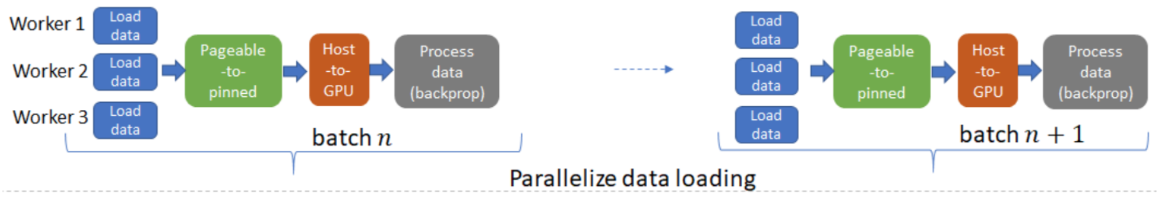

Figure 31: Data loading process. Source [49]

4.2 Data Parallelism

Data Parallelism consists of dividing the data samples that form each training batch into different chunks, each one placed in a different computational resource (GPU in this case). A copy of the model is placed on every device and in each training step a forward-pass is done independently in each of them computing the outputs. Then, the loss with respect to the network parameters is computed, the gradients are averaged and the network's weights are updated. As networks get bigger, the information flowing through the communication channels between GPUs gets bigger since each parameter has its own gradient, reaching even gigabytes per second.

4.2.1 PyTorch's DataParallel library

The simplest way to perform data parallelism is to set a GPU as the master. The master is in charge of getting the outputs computed by each of the network's devices, calculating the gradients and averaging them, then sending back to each GPU as shown in Figure 32.

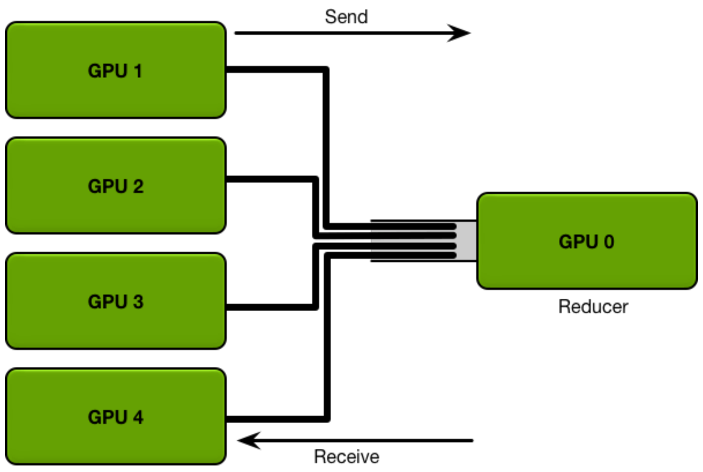

Figure 32: Data Parallelism. Source [26]

In PyTorch this is implemented by means of DataParallel library, which performs the aforementioned operations:

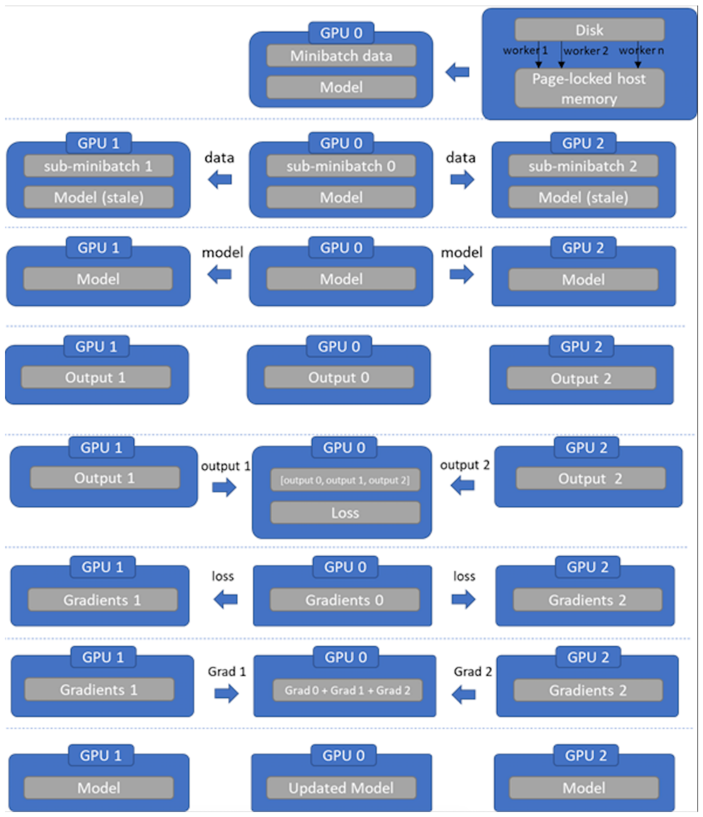

Figure 33: PyTorch's Dataparallel. Source [49]

We can see that data is read from the disk to the host memory, then passed to the master GPU and finally scattered to every other GPU. This is an inefficient procedure. Moreover, before each forward-pass the master must send a new copy of the model with the newly updated parameters so that they are synchronized. Finally, there is a uneven usage of the GPUs since the master GPU is the one performing the loss function and the average of the gradients thus computing more than the rest.

To help mitigate these inefficiencies some solutions in terms of more optimal communication between GPUs have been done. A big step forward has been the introduction of an HPC technique called Ring All-Reduce.

4.2.2 PyTorch's DistributedDataParallel library

DistributedDataParallel applies data parallelism but makes use of a novel technique to solve the previous method communication issues named Ring All-reduced. Ring All-Reduce applied to Deep Learning was introduced by Baidu Research [26]. In this configuration, each GPU is connected to two more following a ring layout where each GPU receives data from its left neighbor and sends data to its right neighbor (Figure 34). As opposed to the previous section method, the cost of the ring all-reduce does not grow linearly as the number of GPUs increases, it remains constant regardless of the number of GPUs and it is only determined by the speed of the communication channel between GPUs.

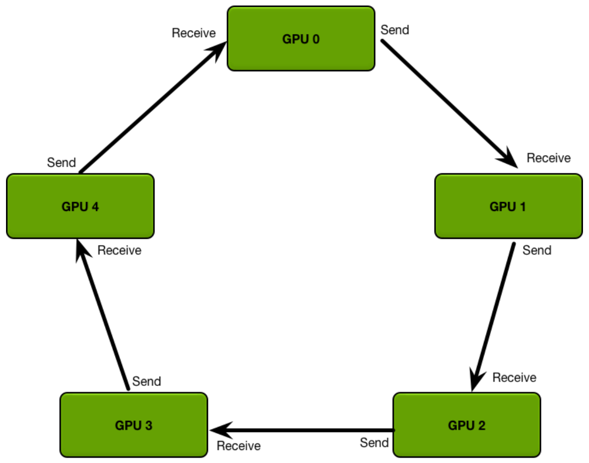

Figure 34: Ring All-reduce method. Source [26]

The algorithm can be divided into two main steps: the scatter-reduce and the all-gather. The scatter-reduce phase consists of dividing data in each GPU into chunks of data, each of those is then shared to the next GPU and a reduced operation is performed, the average for instance. In the all-gather phase every reduced result is communicated until every GPU has the same information. This is very useful in the case of neural networks training. During the backward-pass gradients are calculated starting from the last layer all the way to the first layer. As gradients are being computed in each GPU, the already computed gradients can be scatter-reduced to the rest of the GPUs, while the backward-pass is still being executed.

PyTorch's DistributedDataParallel functionality follows the all-reduce procedure as shown in Figure 35.

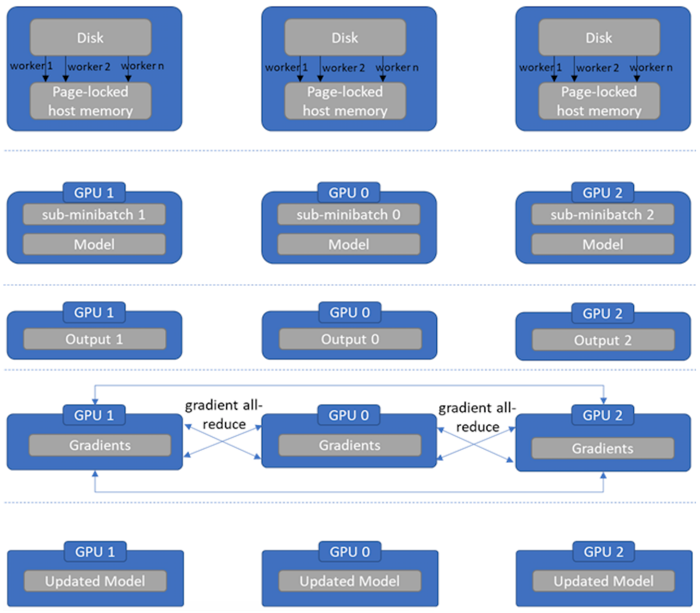

Figure 35: PyTorch's DistributedDataparallel. Source [49]

4.2.3 Parallel platforms results

As mentioned in Section 4 single GPU training requires a huge amount of time, especially when dealing with architectures like in the case of the EfficientNet-B4, which takes almost two days to complete the whole training phase. For this experiment we use the B0, B2 and B4 EfficientNets models. In Figure 36 we show the speedup values for these models with PyTorch's Dataparallel (PyTorch/DP) and PyTorch's DistributedDataparallel (PyTorch/DDP). Clearly the strategy using all-reduce algorithm outperforms the naive data parallel approach.

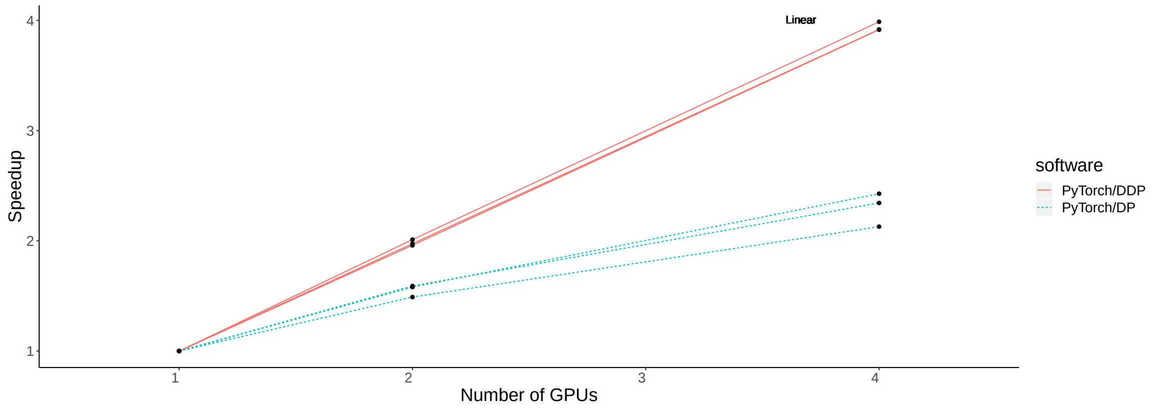

Figure 36: PyTorch DDP vs DP speedup

5 Conclusion

In this work we have presented the use of EfficientNets for the Document Image Classification task and their scaling capabilities through several GPUs. By means of two versions of the Legacy Tobacco Industry Documents, a huge and a small dataset, we demonstrated the training process to obtain high accuracy in both of them. We have compared the different versions of the EfficientNets and raised the state-of-the-art classification accuracy to 92.31% in BigTobacco and 94.04% when fine-tuned in SmallTobacco. We can consider the B0 the best choice when considering limited computational resources. We have also presented an ensemble method by adding the content extracted by OCR. A reduced version of the BERT model is trained and both models predictions are combined to achieve a new state-of-the-art accuracy of 89.47%.

Moreover, we have demonstrated that it is possible to distribute the training process in several GPUs by means of data parallelism and to achieve a linear speedup performance.

With this work we also provide researchers a benchmark in the Document Image Classification task, which can serve as a reference point to effortlessly test parallel systems in both PyTorch and TensorFlow.

Lastly, we release as open source in GitHub the code needed to run the experiments shown in this work and a webpage that shows a brief explanation of the whole work and a usage guide https://javiferran.github.io/document-classification/. As an outcome of this thesis a research paper has been presented in ICCS 2020 [25].

5.1 Acquired knowledge

Throughout the development of this thesis we have been exposed to multiple technologies, which we have been forced to understand both in the theoretical and practical sides. This knowledge includes a deeper understanding of the NLP and Computer Vision techniques that are being developed lately. Since both fields are closely related to Deep Learning, we have been exposed to the recent model architectures and their implementations. Having used PyTorch as the main framework, we have developed fluency not just building Deep Neural Network models but also in many of the packages that form PyTorch. While working on the optimization of the training process of the models, we have acquired a deeper understanding of the current techniques for distributed training such as the HPC technique Ring All-Reduce.

While working at the Barcelona Supercomputing Center - Centro Nacional de Supercomputación we have acquired a bigger understanding of how it is to work on a research project, to meet the deadlines and to work as a group. We are sure this all this knowledge will be valuable in our careers.

5.2 Future work

Future work may evaluate the use of different OCR engines, as we suspect this could have a great impact on the quality of the text model predictions as well as more advanced preprocessing techniques over the extracted text. Another area to focus on is the use of the latest language models developed during the realization of this thesis such as the GPT-3 [13] or language models which are pretrained in related domain texts [28]. In the scalability side, the use of distributed libraries such as Horovod [55] for multiple nodes training, which we have already done some experiments in the Barcelona Supercomputing Center, can be useful to further reduce training times and facilitate the prototyping speed.

References

[1] Afzal, M.Z., Capobianco, S., Malik, M.I., Marinai, S., Breuel, T.M., Dengel, A., Liwicki, M.: Deepdocclassifier: Document classification with deep convolutional neural network. In: ICDAR. p. 1273–1278 (2015)

[2] Afzal, M.Z., Kölsch, A., Liwicki, S.A.M.: Cutting the error by half: Investigation of very deep cnn and advanced training strategies for document image classification. In: ICDAR (2017)

[3] Aggarwal, C.C.: Neural Networks and Deep Learning: A Textbook. Springer (2018)

[4] Akiba, T., Suzuki, S., Fukuda, K.: Extremely large minibatch sgd: Training resnet-50 on imagenet in 15 minutes. arXiv preprint arXiv:1711.04325 (2017)

[5] Alammar, J.: The illustrated transformer. https://jalammar.github.io (2018), https://jalammar.github.io/illustrated-transformer/

[6] Asim, M.N., Khan, M.U.G., Malik, M.I., Razzaque, K., Dengel, A., Ahmed, S.: Two stream deep network for document image classification. In: ICDAR (2019)

[7] Audebert, N., Herold, C., Slimani, K., Vidal, C.: Multimodal deep networks for text and image-based document classification. In: APIA (2019)

[8] Bahdanau, D., Cho, K., Bengio, Y.: Neural machine translation by jointly learning to align and translate. In: ICLR Workshop Papers (2015)

[9] Baldi, S., Marinai, S., , Soda, G.: Using tree-grammars for training set expansion in page classification. In: ICDAR (2003)

[10] Ben-Nun, Hoefler, T.: Demystifying parallel and distributed deep learning: An in-depth concurrency analysis. In: ACM Computing Surveys. vol. 12 (2019)

[11] Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. In: Trans. Assoc. Comput. Linguist (TACL) (2017)

[12] Brinne, B.: The illustrated transformer. https://peltarion.com/ (2019), https://peltarion.com/blog/data-science/illustration-3d-bert

[13] Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., an et al., A.A.: Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020)

[14] Campos, V., Sastre, F., Yagues, M., Torres, J., i Nieto, X.G.: Scaling a convolutional neural network for classification of adjective noun pairs with tensorflow on gpu clusters. In: CCGRID. pp. 677–682 (2017)

[15] Campos, V., Sastre, F., Yagues, M., Torres, M.B.J., i Nieto, X.G.: Distributed training strategies for a computer vision deep learning training algorithm on a distributed gpu cluster. In: ICCS. pp. 315–324 (2017)

[16] Chen, S., He, Y., Sun, J., Naoi, S.: Structured document classification by matching local salient features. In: ICPR. pp. 1558–1561 (2012)

[17] Chollet, F.: Xception: Deep learning with depthwise separable convolutions. In: CVPR (2017)

[18] Collins-thompson, K., Nickolov, R.: A clustering-based algorithm for automatic document separation. In: SIGIR. p. 1–8 (2002)

[19] computersciencewiki.org: Max-pooling / pooling. computersciencewiki.org (2018), https://computersciencewiki.org/index.php/Max-pooling_/_Pooling

[20] Csurka, G., Larlus, D., Gordo, A., , Almazan, J.: What is the right way to represent document images? arXiv preprint arXiv:1603.01076 (2016)

[21] Das, A., Roy, S., Bhattacharya, U., Parui, S.K.: Document image classification with intra-domain transfer learning and stacked generalization of deep convolutional neural networks. In: ICDAR (2018)

[22] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Li, F.F.: Imagenet: a large-scale hierarchical image database. In: CVPR. pp. 248–255 (06 2009)

[23] Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. In: NAACL (2019)

[24] Diligenti, M., Frasconi, P., , Gori., M.: Hidden tree markov models for document image classification. In: Transactions on Pattern Analysis and Machine Intelligence (TPAMI) (2003)

[25] Ferrando, J., Domínguez, J.L., Torres, J., García, R., García, D., Garrido, D., Cortada, J., Valero, M.: Improving accuracy and speeding up document image classification through parallel systems. In: Krzhizhanovskaya, V.V., Závodszky, G., Lees, M.H., Dongarra, J.J., Sloot, P.M.A., Brissos, S., Teixeira, J. (eds.) Computational Science – ICCS 2020. pp. 387–400. Springer International Publishing, Cham (2020), https://doi.org/10.1007/978-3-030-50417-5_29

[26] Gibiansky, A.: Bringing hpc techniques to deep learning. https://andrew.gibiansky.com/ (2017), https://andrew.gibiansky.com/blog/machine-learning/baidu-allreduce/

[27] Goyal, P., Dollar, P., Girshick, R., Noordhuis, P., Wesolowski, L., Kyrola, A., Tulloch, A., Jia, Y., He, K.: Accurate, large minibatch sgd: Training imagenet in 1 hour. CoRR, vol. abs/1706.02677 (2017)

[28] Gururangan, S., Marasović, A., Swayamdipta, S., Lo, K., Beltagy, I., Downey, D., Smith, N.A.: Don't stop pretraining: Adapt language models to domains and tasks. In: Proceedings of ACL (2020)

[29] Harley, A.W., Ufkes, A., Derpanis, K.G.: Evaluation of deep convolutional nets for document image classification and retrieval. In: Proc. ICDAR 2015. IEEE. p. 991–995 (2015)

[30] ul Hassan, M.: Vgg16 – convolutional network for classification and detection. https://neurohive.io (2018), https://neurohive.io/en/popular-networks/vgg16/

[31] He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition (2015)

[32] He, T., Zhang, Z., Zhang, H., Zhang, Z., Xie, J., Mu Accurate, L.M.S.: Bag of tricks for image classification with convolutional neural networks. arXiv preprint arXiv:1812.01187 (2018)

[33] Hochreiter, S., Schmidhuber, J.: Long short-term memory. In: Neural computation, 9(8):1735–1780 (1997)

[34] Howard, J., Ruder, S.: Universal language model fine-tuning for text classification. In: Association for Computational Linguistics. vol. 1, p. 328–339 (2018)

[35] indoml: Student notes: Convolutional neural networks (cnn) introduction. indoml (2019), https://indoml.com/2018/03/07/student-notes-convolutional-neural-networks-cnn-introduction/

[36] Jayant, K., Peng, Y., David, D.: Structural similarity for document image classification and retrieval. In: Pattern Recognition Letters. vol. 43, pp. 119–126 (2014)

[37] Joulin, A., Grave, E., Bojanowski, P., Mikolov, T.: Bag of tricks for efficient text classification. arXiv preprint arXiv:1607.01759 (2016)

[38] Kang, L., Kumar, J., Ye, P., Li, Y., Doermann, D.: Convolutional neural networks for document image classification. In: ICPR. p. 3168–3172 (2014)

[39] Kim, Y.: Convolutional neural networks for sentence classification. In: EMNLP (2014)

[40] Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems (2012)

[41] Kumar, J., Doermann, D.S.: Unsupervised classification of structurally similar document images. In: ICDAR. pp. 1225–1229 (2013)

[42] Kumar, J., Ye, P., Doermann, D.S.: Learning document structure for retrieval and classification. In: ICPR. pp. 653–656 (2012)

[43] Lewis, D., Agam, G., Argamon, S., Frieder, O., Grossman, D., Heard., J.: Building a test collection for complex document information processing. In: SIGIR. pp. 665–666 (2006)

[44] Luong, M.T., Pham, H., D.Manning, C.: Effective approaches to attention-based neural machine translation. In: ICLR Workshop Papers (2015)

[45] McCann, B., Bradbury, J., Xiong, C., Socher, R.: Learned in translation: Contextualized word vectors. In: Advances in Neural Information Processing Systems. pp. 6297–6308 (2017)

[46] McCloskey, M., Cohen., N.J.: Catastrophic interference in connectionist networks: The sequential learning problem. In: Psychology of learning and motivation. vol. 24, pp. 109–165 (1989)

[47] Merity, S., Keskar, N.S., Socher, R.: Regularizing and optimizing lstm language models. In: ICLR (2018)

[48] Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. In: ICLR Workshop Papers (2013)

[49] Mohan, A.: Distributed data parallel training using pytorch on aws. http://www.telesens.co (2019), http://www.telesens.co/2019/04/04/distributed-data-parallel-training-using-pytorch-on-aws

[50] Noce, L., Gallo, I., Zamberletti, A., Calefati, A.: Embedded textual content for document image classification with convolutional neural networks. In: Proceedings of the 2016 ACM Symposium on Document Engineering (DocEng '16) (2016)

[51] Olah, C.: Understanding lstm networks. http://colah.github.io/ (2018), http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[52] Peters, M.E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L.: Deep contextualized word representations. In: Proc. of NAACL (2018)

[53] Roy, S., Das, A., Bhattacharya, U.: Generalized stacking of layerwise-trained deep convolutional neural networks for document image classification. In: 23rd International Conference on Pattern Recognition (ICPR). p. 1273–1278 (2016)

[54] Sandler, M., Howard, A., Menglong, Zhu, Zhmoginov, A., Chen, L.C.: Mobilenetv2: Inverted residuals and linear bottlenecks. In: CVPR. pp. 4510–4520 (2018)

[55] Sergeev, A., Balso, M.D.: Horovod: fast and easy distributed deep learning in TensorFlow. arXiv preprint arXiv:1802.05799 (2018)

[56] Shin, C., Doermann, D.S.: Document image retrieval based on layout structural similarity. In: IPCV. pp. 606–612 (2006)

[57] Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR, abs/1409.1556 (2014)

[58] Smith, R.: An overview of the tesseract ocr engine. In: International Conference on Document Analysis and Recognition (ICDAR) (2007)

[59] Sun, C., Qiu, X., Xu, Y., Huang, X.: How to fine-tune bert for text classification? arXiv preprint arXiv:1905.05583 (2019)

[60] Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. In: Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N.D., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 27, pp. 3104–3112. Curran Associates, Inc. (2014), http://papers.nips.cc/paper/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[61] Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., Le, Q.V.: Mnasnet: Platform-aware neural architecture search for mobile. In: CVPR (2019)

[62] Tan, M., Le, Q.V.: Efficientnet: Rethinking model scaling for convolutional neural networks. In: International Conference on Machine Learning (2019)

[63] Tensmeyer, C., Martinez, T.: Analysis of convolutional neural networks for document image classification. In: ICDAR (2017)

[64] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L.u., Polosukhin, I.: Attention is all you need. In: Advances in Neural Information Processing Systems 30. p. 6000–6010 (2017)

[65] Wang, R., Su, H., Wang, C., Ji, K., Ding, J.: To tune or not to tune? how about the best of both worlds? arXiv preprint arXiv:1907.05338 (2019)

[66] Weng, L.: Generalized language models. lilianweng.github.io/lil-log (2019), http://lilianweng.github.io/lil-log/2019/01/31/generalized-language-models.html

[67] Zhang, Y., Wallace, B.C.: A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. arXiv preprint arXiv:1510.03820 (2015)

[68] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., Fidler, S.: Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In: ICCV. pp. 19–27 (2015)