Advantages of pre-trained models
Deep Learning

Luca Bandelli (s3221253)
l.bandelli@student.rug.nl

Amitoj Battu (s4330617)
a.battu.1@student.rug.nl

Marieke Bouma
m.bouma.10@student.rug.nl

Pedro Rodriguez de Ledesma Jimenez
p.rodriguez.de.ledesma.jimenez@student.rug.nl

June 23, 2023

Abstract

Different data types call for different components in a deep learning pipeline. In this research, the aim is to explore the advantages of a pre-trained model compared to a model trained from scratch. Training both models on the same task for comparison, various approaches are reported for data pre-processing, parameter selection, classification and evaluation. For the image dataset, a maximum accuracy of score of 0.91 is achieved for pre-trained model and a score of 0.75 for the scratch trained model. For the natural language processing dataset, a maximum accuracy score of 0.997 is achieved for pre-trained model and a score of 0.982 for the scratch trained model. Reasoning is given behind the different choices in approaches and the resulting performances as a manner of comparing the model types.

1 Introduction

Deep learning is gaining popularity due to its supremacy in terms of accuracy when trained with huge amounts of data that outperform other approaches and even humans at many problems. Despite its popularity, models are still unable to accurately predict the time it will take to train a network to solve a given problem. This training time can be seen as the product of training time per epoch and the number of epochs which need to be performed to reach a desired level of accuracy. Training a neural network model may take days or even weeks on very large datasets. Other issues that models deal with is the disposal of large amounts of training datasets to train the models that plays a critical role in the model generalization. Therefore, different techniques are used to shortcut the training process and archive generalization with small datasets. Transfer learning re-uses the model weights from pre-trained models trained on a similar task for a new model with a related task. The weights in re-used layers may be used as the starting point for a training process and adapted in response to the new task. This usage treats transfer learning as a type of weight initialisation scheme and these models are called pre-trained models.

In this paper pre-trained models are compared with models trained from scratch in terms of accuracy and time of training. For it, two different tasks have been selected in order to achieve this goal. The first task the selected model identifies fake news, while the second task the model classifies three different thematics in paintings. In the next sections the dataset used are presented, the architectures of the model defined and the insights of the experiments exposed.

2 Methodology

2.1 Computer Vision task

In this part of the research, we perform image classification of art paintings using MobileNet.

2.1.1 MobileNet

MobileNet is a type of convolutional neural network designed for mobile and embedded vision applications. Using depthvise separable convolutions, it significantly reduces the number of parameters when compared to networks with regular convolutions with same depth in the nets. This overall results in lightweight deep neural network. The depthwise separable convolution is made from two operations:

- Depthwise convolution: Implemented by separating a filters' depth and spatial dimensions.
- Pointwise convolution: Implemented by convulating with a kernel of size 1x1 to combine the features created by the depthwise convolution.

The size of an input image is $224 \times 224 \times 3$. This architecture was selected due to the fact that it is readily available as a `keras` application and due to its simplicity (having only 4.3 million parameters), allowing us to run experiments in a short period of time and focus on the goal of the assignment. The pre-trained model used was trained with the dataset ImageNet, a large visual database (over 1 million images, 1000 classes) designed for use in visual object recognition software research.

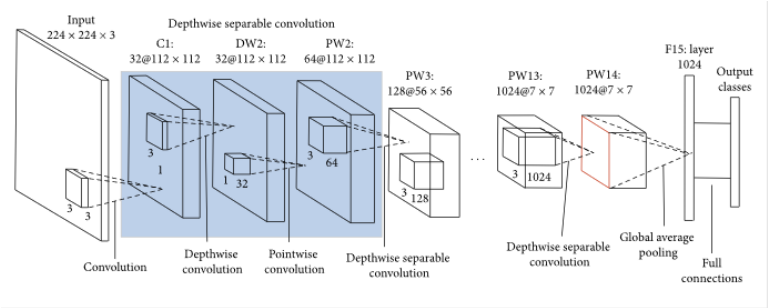

Figure 1: MobileNet architecture

2.1.2 Activation Function - Softmax

Softmax is a mathematical function that converts a vector of numbers into a vector of probabilities, where the probabilities of each value are proportional to the relative scale of each value in the vector. For our research, it is used as an activation function in the final layer where it is configured to output N values, one for each class in our classification task, and the softmax function is used to normalize the outputs, converting them from weighted sum values into probabilities that sum to one. Overall, each output value of the function is equated as the probability of each class.

$$Softmax(\vec{x_i}) = \frac{exp(x_i)}{\sum_{j=1}^{k} exp(x_j)}$$

2.1.3 Data and preprocessing

The image dataset used for our pipeline is a collection of paintings divided into three classes: `marina`, `flower`, and `animal`. There exist around 1800 instances of each class, from which 200 images were split into test sets per class, 600 images split into validation per class and the remaining were used for training. The split was performed manually. The dataset was obtained using the Kaggle database website `https://www.kaggle.com/datasets/ipythonx/wikiart-gangogh-creating-art-gan`. Since each image depicts only one class, the task at hand is multi-class, single-label classification.

After reading in the images and analyzing the distribution and dimensions, pre-processing approaches were considered. Using an import function, all images were reshaped into 224x224x3 dimension using interpolation so as not to lose information. Further, all pixel values were normalized between the values of -1 and 1 for easier convergance of the model to an optimum solution. For this purpose, a `keras` inbuilt function was utilized.

Figure 2: Example images, one of each class.

2.1.4 Data Augmentation

For there being 1800 images per class, we decided to also perform data-augmentation per class to achieve a better result. This practice yielded unexpected results as the scratch model showed above 95 percent accuracy on the training set but a poor 55 percent accuracy on the evaluation test set. We hypothesize this is being caused by the scratch model over-fitting on the training set. Reducing epoch cycles for training yields similar dissatisfying results.

2.1.5 Baseline Classification - KNN

To compare the validity of the choice of a CNN for this task, we compare the results of our scratch model compared to those of a baseline classifier on the same dataset. We use *k*-Nearest Neighbors, one of the most well-known supervised methods for classification. The idea behind KNN is to classify new data points based on their similarity measures with other data points. For this study, we used the Euclidean distance as a similarity measure because it is the most widely used and straightforward method. For KNN's implementation, we used scikit-learn software. The main hyperparameter that can be changed for KNN is the number of nearest neighbours K that is taken into account when classifying a data point.

2.1.6 Procedure

For ease of use, we load the MobileNet architecture as a `keras` application. This way, we can use the same model either with pretrained weights or randomly initialized weights when calling an instance of the model. Either way, we need to alter some of the last layers of the model such that the output is one of the three classes in our dataset, rather than one of the 1000 ImageNet classes. Moreover, when using the model with pretrained weights, we must make all layers except those on top (the ones we modified) untrainable.

We load the images and train the models on them in batches of 32 images at a time. We train the respective models for 20 epochs, which is enough to reach decent training accuracies.

2.1.7 Hyperparameters and tuning

As is standard with classifications where the target label is an integer, we use sparse categorical cross-entropy as a loss function. In our base model, the optimizer used is Adam. We investigate the optimizer that Adam is based on, SGD, as well. In the original MobileNet model, the default value for the dropout rate is 0.001. However, this is a parameter we can easily tune. In the interest of saving time, we perform our hyperparameter sweeps on the pretrained model only.

Lastly, the MobileNet architecture allows for two hyperparameters to be tuned: the depth multiplier and the width multiplier. In the interest of saving time, we leave this tuning to future research.

2.2 NLP task

2.2.1 Task and dataset

The NLP task selected is related to the ever-so-current topic of fake news detection. The phenomenon of fake news takes on several forms on different media platform, ranging from Twitter and other social media, to (sometimes alleged) news outlets. In the present work, we focus on news articles rather

than tweets or other forms of posts. The task involves working with raw text, which may need to be preprocessed, before using some neural model for automatic feature extraction feeding representation to a classifier network.

The simpler approach is to look at the problem as a binary classification task discerning "True" news from "Fake" news. The domain is actually a bit more sophisticated and other related datasets treat the problem as a multi-class one, where the class of fake news is divided into categories like "hoax", "propaganda", "satire".

The dataset used for this task, taken from `https://www.kaggle.com/datasets/clmentbisaillon/fake-and-real-news-dataset`, consists of 44898 news articles roughly equally distributed between reliable news and fake news (23481 fake 52%, 21417 true 48%).

The original dataset contains the title and the text of each news article, which were combined to get a full text column.

2.2.2 Text preprocessing

The text of the news articles was preprocessed according to 2 different pipelines. The input for the scratch-trained model receives some more heavy preprocessing involving the removal of stop words, and Lemmatization. The input for the transformer based model was processed with a lighter pipeline as suggested by documentation on transformer models, which does not involve lemmatization nor stop word removal.

Often, the fake news text contains particular pieces of text that are typical only of fake news, typically at the end of the text:

- 'via: ... ' or 'source: ... ' reporting the alleged news source;
- 'Photo by ... ' giving credits for a picture;
- 'Featured ... ' signatures

Preprocessing involves the 'fnws1.dataprep' module.

Similarly, genuine articles often contain the news source and geographic location of the writer at the beginning of the text separated by a dash.

We think these 'artifacts' could be used as discriminative features rather focusing on the broader text, so we decided to remove them. Both preprocessing pipelines include the removal of 12 of these artifacts. Finally urls and '@' mentions are also removed from the text.

2.2.3 Models

The **scratch trained** model is implemented in Pytorch, using the `torchtext` module to build the a vocabulary index for the tokens (after preprocessing). The tokenizer turns the news article text into a vector of integer indices suitable for neural networks. The input is fed to an EmbeddingBag layer, which computes the embedding mean for the text without computing the individual word embeddings. This is a great simplification as it allows to feed the input to a simple linear layer rather than using a recurrent network downstream. In fact the model is exceptionally simple (yet effective) as it only consists of the EmbeddingBag layer and a Linear fully connected layer. Weights are randomly initialised in the range [-.5,.5] and biases are set to zero.

Our problem is actually a binary classification task, but since fake news detection could be extended to a multi class setting where different type of fake news are distinguished, the model is parametrized on the number of classes and outputs n_{classes} values, in our case 2.

The model is therefore trained using crossentropy loss, and optimized via stochastic gradient descent with a configurable decay schedule. The training also uses gradient norm clipping. Loss, accuracy and f1 score are collected as metrics each epoch.

The second model built in this research is more complex and involves the use of a **Transformer** to generate the input for the feedforward classifier.

The approach is not different structurally to the previous model. While before the tokenized input was fed to the EmbeddingBag layer, it is now fed to a bidirectional encoding representation from transformers (BERT) module. In both cases the sequence of input tokens is turned in to a fixed size vector representation which is used downstream in the feed forward network.

The chosen model was `distilbert-base-uncased` from huggingface. This provides BERT-like performance with fewer parameters, and therefore vram usage and computation time. It uses the associated `distilbert-base-uncased` tokenizer to generate input ids to input to the transformer (similar to the input given to the embedding bag) but with the special tokens for the transformer and necessary padding applied. Note that the maximum input length for distilBERT is 512 so long articles need to be truncated. We actually lowered the truncation to 256 in order to speed up computations.

The model uses the last hidden state and in particular the embedding given for the start token, representative of the whole sentence, as input for the classifier. The classifier consists of 2 sequential Linear layers, the first using ReLU activation and dropout before feeding to the final layer that maps to classification output logits. The final layer weights are initialized using Xavier normal method from torch.

The model uses the same crossentropy loss as the simpler model, and the `Adam` optimizer with a fixed learning rate. To experiment with overfitting control we also tested the model with weight decay within `Adam`, alongside 2 variations of the `adadelta` optimizer with weight decay.

model name	optimizer	decay lambda
$\text{distilBERT}_{\text{adam}}$	Adam	0
$\text{distilBERT}_{\text{adam decay}}$	Adam	0.001
$\text{distilBERT}_{\text{adadelta small decay}}$	Adadelta	0.001
$\text{distilBERT}_{\text{adadelta large decay}}$	Adadelta	0.01
$\text{simple}_{\text{model}}$	SGD	0

2.2.4 Hyperparameters

For the scratch trained model the size of the embedding space is tunable parameter set to 42 for the results below, the size of the vocabulary is determined by the training dataset (for our training dataset this amounts to 115436 words). Learning rate and scheduler step size are also tunable and set to 1 and 0.1 respectively. For the transformer based model the width of the intermediate hidden layer ($\texttt{hidden}_{\texttt{size}}$) is configured to 768. The dropout rate for the dense connections between the 2 linear layers is set to 0.1. The vocabulary size is the default 32000 for distilbert. The transformer is configured to have 12 hidden layers and 12 attention heads, with an intermediates size of 3072. Learning rate is set to 3e-5.

The choice and parameter setting for the optimizers also constitutes a hyperparameter; their values are indicated in table 2.2.3 above.

3 Results

3.1 Results (Computer Vision)

3.1.1 Training results

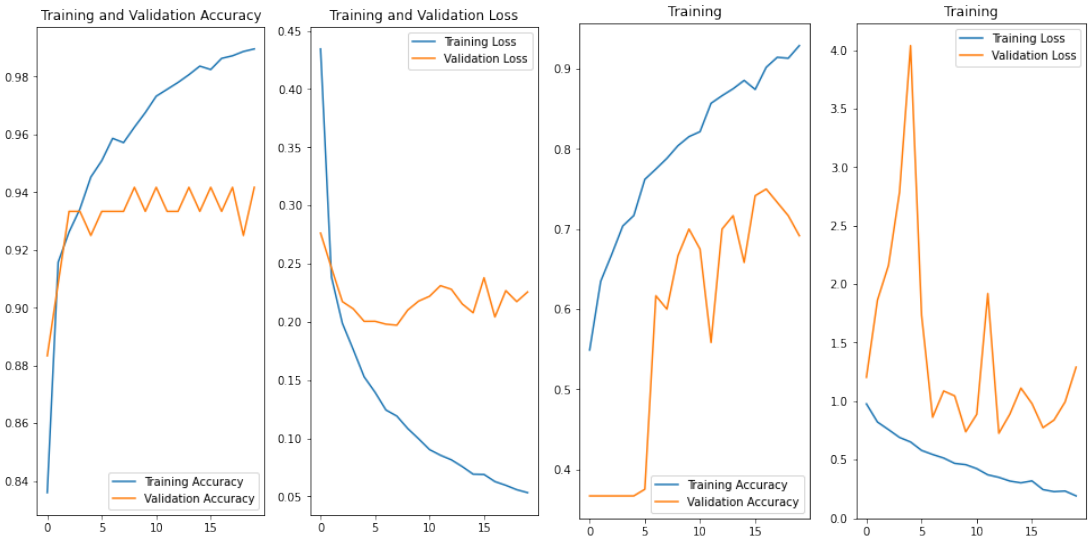

Figure 3: Plots of train and validation loss and accuracy over 20 epochs (left: pre-trained base model, right: scratch-trained base model)

As can be seen in Figure 3, both models reach rather high accuracies quickly. To check whether the data is biased, we include in our procedure a cycle of 5 runs, in which we present the data to the base models shuffled in each run. For each run the model weights are reset. We report the final accuracy of each run in Table 1. As can be seen, the results for each run are consistent with those of the other runs, and we can conclude that the data is not biased.

Pre-trained Accuracy	Loss	Scratch-trained Accuracy	Loss
0.9875	0.0528	0.9271	0.2026
0.9888	0.0537	0.9291	0.1833
0.9883	0.0552	0.9286	0.1911
0.9893	0.0514	0.9242	0.2038
0.9885	0.0523	0.9265	0.1859

Table 1: Final accuracies and results for base models over five runs.

As mentioned, in the interest of saving time, we perform our hyperparameter sweeps on the pre-trained model only.

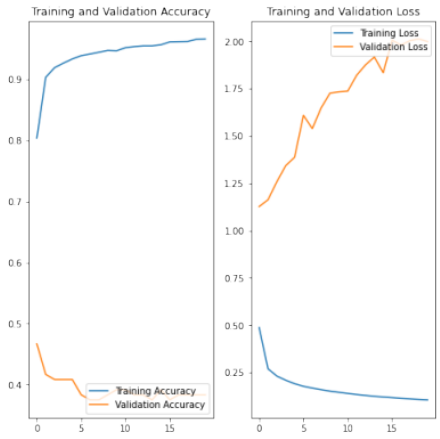

Figure 4: Plots of train and validation loss and accuracy over 20 epochs (optimizer=SGD, dropout=0.001)

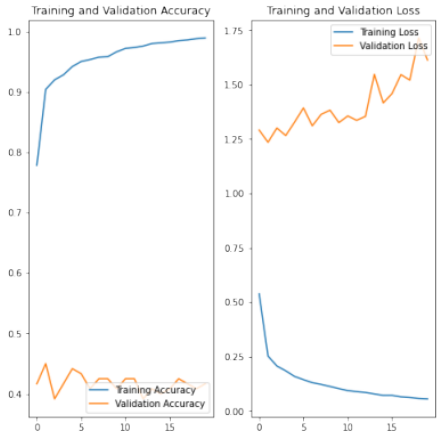

Figure 5: Plots of train and validation loss and accuracy over 20 epochs (optimizer=Adam, dropout=0.01)

3.1.2 Test results

Next to the training results, we must also analyze the test performance of the models.

Pre-trained Accuracy	Loss	Scratch-trained Accuracy	Loss
0.9167	0.2122	0.6217	1.5734

Table 2: Single run of testing the base models on our test set.

As with the training results, we analyze the final accuracies over five runs to check that the data is consistent:

Pre-trained Accuracy	Loss	Scratch-trained Accuracy	Loss
0.9200	0.2054	0.7433	1.031
0.9250	0.1919	0.7533	1.088
0.9183	0.1977	0.7233	0.9822
0.9233	0.1857	0.7317	1.0659
0.9200	0.2016	0.6883	1.0294

Table 3: Five runs of testing the base models on our test set.

3.2 Results (NLP)

3.2.1 Training Results

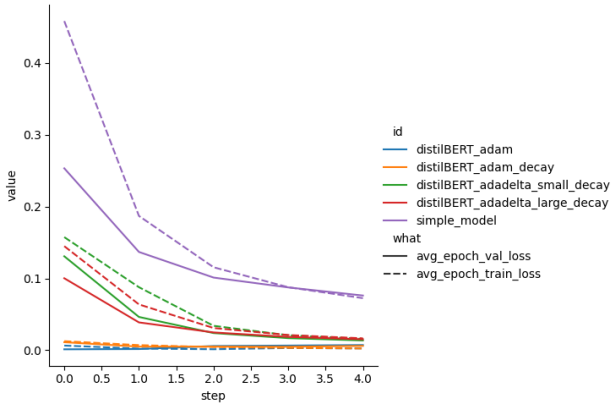

Figure 6: Plot of train and validation loss for NLP task

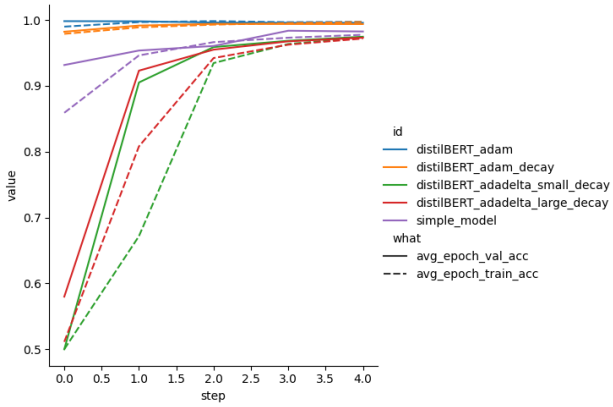

Figure 7: Plot of train and validation Accuracy for NLP task

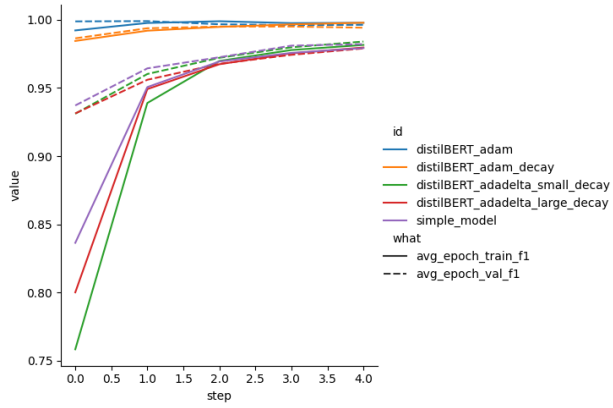

Figure 8: Plot of train and validation F1Score for NLP task

Both models achieve extremely good performance on both training set and validation set performance (on both evaluation metrics) as shown in the figures (7, 8). Validation loss was used to determine that

the model starts overfitting the training data around epoch 5.

The scratch train model does not achieve the same exceptionally low loss as the BERT based models (see figure 6), but performs comparably in terms of performance metrics to all but the best of the BERT based models, with validation accuracy performance at 98%.

The best model in terms of validation loss and performance metrics is the distilBERT model trained with the Adam optimizer. In fact when trained with the Adam optimizer the network reaches over 97% accuracy well before the end of the first epoch. The final validation loss is above 99%.

3.2.2 Test set Results

Due to the suspicously high performance obtained by the model, besides the test set from the original dataset, two other datasets (Dutta: `https://www.kaggle.com/rchitic17/real-or-fake`, Rashkin: `https://homes.cs.washington.edu/~hrashkin/factcheck.html`) were used to challenge the classifiers. The intuition behind such high performance is that while we may not be overfitting on the training set per se, a single one of these fake news datasets may not be diverse enough to capture the spectrum of the problem at hand, resulting in an overly simple task where the classifier distinguishes items based on structural characteristics of the texts, that may be specific to that particular data collection.

For the scratch trained model the following test set metrics were obtained:

test set	Accuracy	F1 score	Loss
original	0.982	0.980	0.071
rashkin	0.944	0.971	0.179
dutta	0.789	0.662	1.129

As we can see performance degrades with the Rashkin dataset, which contains only fake news, but from a broader set of sources; it degrades further when using the dataset collected by Dutta and colleagues.

For the pretrained trained model the following test set metrics were obtained:

model	test set	Accuracy	F1 Score	Loss
$\text{distilBERT}_{\text{adam}}$	original	0.997	0.997	0.0116
	rashkin	0.832	0.909	1.4005
	dutta	0.550	0.621	3.9808
$\text{distilBERT}_{\text{adamdecay}}$	original	0.994	0.993	0.0187
	rashkin	0.823	0.903	0.7339
	dutta	0.556	0.621	2.2452

The accuracy is almost 100% on the main testing set (30% of the original kaggle dataset) Performance degrades for the transformer based model too when using datasets from a different data collection procedure.

4 Discussion

4.1 Computer Vision task

For the image data, a best classification accuracy of 0.91 was achieved using the pre-trained model and an accuracy of 0.75 using scratch model. This result was consistent with our original hypothesis, as pre-trained MobileNET model was originally trained on full dataset of ImageNET for far more epochs as compared to the 20 epochs our scratch trained model had been trained on. Inversely, this research shows the importance of transfer learning, how it is viable to utilize large trained models and transfer their weights into our own test dataset.

Changing either of the hyperparameters we have tried makes the model overfit strongly, as can be seen in the fact that validation and training metrics in fact move apart from each other.

4.2 NLP task

The main observation here is that care should be taken when building models for the classification of these types of texts. Ensuring a broad spectrum of source types is included to avoid obtaining a model too that is too specialized. This seems to apply in particular for transformer based models due to their high flexibility and complexity.

The high performance that is achievable with the embeddingBag only model may be a signal that the task, approached with these datasets, is indeed simpler than the broader problem is, due to the dynamics of the data collection.

Further research in this direction could focus on understanding where 'Out of distribution' text samples are positioned in the embedding space, for example by collecting the embedding vectors for a set of texts and computing the principal components of these feature vectors to gain insight into how the two classes distributed in the feature space.

5 Overall Conclusions

For this research, the aim was to compare pre-trained model to a scratch model and create two different deep learning pipelines for two very distinctive datasets, namely one consisting of image data and one consisting of Natural language data. We showed how the approaches that are chosen for data selection, pre-processing, model selection and transfer learning are highly dependent on the type of data. In addition, we showed how various approaches for one dataset produce a variety of results.

6 Contributions

Luca took on the NLP task on his own, while Marieke and Amitoj worked on the Image dataset pipeline. All four of us wrote the report together, although Pedro could not contribute significantly in the final stages of the writing due to health reasons. Overall, our communication did not run smoothly, which delayed many parts of the project considerably.