Finegrained Multimodal Fake News Detection With Subsample of Fakeddit Dataset

NARVE GILJE NORDÅS, University of Stavanger, Norway

The purpose with this report was training a 10 percent sub-sample of the Fakeddit dataset to evaluate if a multimodal model, using both text and image as input, would increase performance compared to unimodal models trained with the same hyper-parameters; one CNN-model for image and one BiLSTM-model for text. Conclusion was that it did, but not by much, going from a total accuracy of 0.74 for unimodal text-model to 0.76 for multimodal model. One was also going to evaluate if the increase of total accuracy using a multimodal model was the same when training the model with just 10 percent of the entire dataset. This was not the case, as some of the related work that trained and validated the model with the full Fakeddit-dataset both increased accuracy by around 9 points [9], [13], where only an increase of 2 percent was achieved with the sub-sample. Both unimodal models, and the multimodal model performed poorer than the related work that used the entire Fakeddit-set [9], [13]. It is not 100 percent certain what caused this decrease in performance, but one can be sure that lack of training-data have caused the constructed models in this report to perform poorer, and overfit to training-data.

CCS Concepts: • **General and reference** → Experimentation.

Additional Key Words and Phrases: Fake, News, Detection, Multimodal, Finegrained, Multiclass, Fakeddit, Subsample, BiLSTM, CNN

ACM Reference Format:
Narve Gilje Nordås. 2024. Finegrained Multimodal Fake News Detection With Subsample of Fakeddit Dataset. 1, 1 (April 2024), 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

Link to source-code on Github: https://github.com/narvegn/Multimodal_fake_news_detection_subsample_Fakeddit/tree/main

1 INTRODUCTION

The main goal of this project is to use machine learning and AI (Artificial Intelligence) with sub-samples of the Fakeddit dataset [9] to detect false information, and compare evaluation metrics from two unimodal models with one multimodal model. Unimodal means a model that takes only one input to produce the output. The unimodal models are a Convolutional Neural Network (CNN) that takes an image as input, and a neural network that contains a bidirectional Long-Short-Term-Memory-layer which takes the title/text as input. The multimodal model takes both image and title as input, using the same hyper-parameters as the unimodal models, before concatenating them together and sending the concatenated vector through the output layer. The Fakeddit dataset contains both a 6-way-label as well as 2-way -and 3-way-labels for classification [9]. The 6-way-label will be narrowed down to a 5-way-label, where it is still a fine grained classification, as opposed to a simple binary classification, where the news are either true or false. In this case the news are either true, or they are in a subcategory of false, divided into misleading content, false connection, satire content, imposter content and manipulated content [9]. As mentioned in the beginning, one will study the difference of accuracy of the multimodal model with the unimodal models, using the exact same hyper-parameters in the multimodal model, to see if adding an extra input stream will increase the accuracy. On the other side, one will also study the effect of only training the models on a small subset of the data (10 percent), as opposed the entire dataset. This will be evaluated with comparison of evaluation metrics such as accuracy and F1-scores from existing work, where the entire Fakeddit dataset is used, and the classification is fine-grained/multiclass.

The reason one is building models that can automate fake news detection, is because false news are a growing problem in today's society, as more people than ever before are exposed through different social media platforms. The consequences of fake news exposure might be large, and can lead to the public forming opinions based on false information [4].

Both the fact that false information could have large negative consequences and that fact-checking this information is very important, have now been established. The main problem is however, the rapid diffusion of these false news, where more people can be exposed to false information, than the truth [4]. The key part is that it is impossible to manually fact-check every news article on the internet, and because of this the false news detection also has to become automated, using AI [8].

2 RELATED WORK

The main focus regarding related work will be studies that have performed both fine-grained multimodal fake news detection, as well as unimodal fine-grained classification on similar datasets (containing image and text), and looking at the effect of multimodal models.

The creators of the Fakeddit dataset claimed in 2020 that there were very few fake news datasets that contained both image and text [9]. They listed as few as two datasets, besides their own; namely Fauxtography [20] and image-verification-corpus [1]. While the datasets both are multimodal, they have a few downsides compared to Fakeddit: The size is smaller; Fauxtography contains as little as just over 1200 rows of data [20], whereas image-verification-corpus has just under 18.000 rows [1]. Compared to Fakeddits over 500.000 multimodal samples [9], it is only a small fraction of this data. Both of the datasets mentioned also only contains a binary-label, which is either true or false, which in other words is a 2-way classification label.

Author's address: Narve Gilje Nordås, 251985@uis.no, University of Stavanger, Stavanger, Rogaland, Norway.

XXXX-XXXX/2024/4-ART $15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Further on one will focus on the Fakeddit dataset, since one key part of this project is to check if the difference in accuracy/F1-score is the same between unimodal and multimodal models with the same hyper-parameters, when only a subset of the dataset is trained. The creators of Fakeddit dataset carried out an experiment where they tested the accuracy of unimodal models [9]; one for text and one for image. They also tested the same accuracy on different multimodal models. The unimodal models for text were Bidirectional Encoder Representations from Transformers (BERT) and InferSent. BERT performed best with an accuracy of 76.77 percent. The unimodal image models were VGG16, EfficientNet and ResNet50. ResNet50 had the highest score with accuracy of 75.49 percent. Lastly they trained 6 multimodal models, all combinations of the unimodal models mentioned above. Maximum-method used, as this yielded largest accuracy. Meaning maximum from both the text -and image-model is put together before the final output layer [9]. The best multimodal model was not surprisingly BERT for text combined with ResNet50 for image, and the accuracy was almost 10 points larger than the best unimodal model, with 85.88 percent. It is also worth noting that the accuracy were higher for all models when using 2 -and 3-way labels. Which makes sense, as more class labels add more complexity to the model, and the probability of the model "guessing" a correct answer decreases ($\frac{1}{2}$ compared to $\frac{1}{6}$)

There were also two students at the University Carlos III Madrid that had carried out experiments with the Fakeddit dataset [13], in 2022, two years after the release of the dataset. In this case there were 3 unimodal models for text, and no unimodal model for image. The three models were CNN, BiLSTM (Bidirectional Long-Short-Term-Memory) + CNN and BERT. Also here BERT had the highest accuracy with 78 percent, with F1-score at 74 percent. The multimodal model outperformed all unimodal text-models with an accuracy of 87 percent, and F1-score of 87 percent. The multimodal model had a CNN for both text and image before concatenating the input streams together, going into the output layer. These students claimed at the time (2022) that apart from the creators of the Fakeddit dataset they had not come across any other researchers that had performed multiclass (6-way) fake news classification with the full dataset.

There was however some newer related work, utilizing the entire dataset, performing 6-way classification. One approach used DistilBERT for text, and VGG16 for image [6]. This approach achieved an accuracy of 60.38 percent for the multimodal model. An other approach utilized ResNet for image features, and RCNN (Region-based CNN) for the text. This multimodal model also tried balanced sampling (equal occurrence for class labels), and reached an accuracy of 89.82 percent on Fakeddit with 6-way classification [18].

Using the entire dataset for fine-grained classification, the results from all of this previous work is very clear when it comes to saying that multimodal, multiclass models have higher accuracy than the unimodal, multiclass models, when evaluated -and trained with the entire multimodal Fakeddit dataset.

3 METHODOLOGY

3.1 Reading an cleaning the dataset

The Fakeddit multimodal dataset consisted of several "Reddits"/posts from different users, including a title with a corresponding image-URL [9]. The training, -validation -and testing Fakeddit datasets were all downloaded separately as tsv-files, and then loaded as dataframes. Each dataframe had 16 features, including image-URL, a clean title, id and three different class labels (2-way, 3-way and 6-way). The training, -validation -and testing sets contained 564.000, 59.342 and 59.319 rows respectively before cleaning.

The 6-way-label column contained 6 different classes, and they were all denoted with integers, ranging from 0 to 5. A brief explanation of each class label follows [9]:

- `0, True`: News that are true.
- `1, Satire/Parody`: Twisted content to represent news in a humorous or ironic way.
- `2, Misleading Content`: Deliberately manipulated content, where purpose is to mislead or put fire under conspiracies.
- `3, Imposter Content`: Bot-generated content.
- `4, False Connection`: When text and image is not in accordance.
- `5, Manipulated Content`: Manipulated content, namely editing and alterations of images.

Distributions of each class was then evaluated with a histogram of class label. The distributions for all datasets showed a very high occurrence of class 0 labels, medium high occurrence of class 4 -and 2, and low occurrence of the remaining classes (1, 3 and 5). In other words an imbalanced dataset.

There were some issues downloading images that had no domain due to some websites not existing anymore, so all rows where domain-feature was "NaNs" were removed. After removal of these rows, the histograms of class-distributions showed that all rows with class 4 as label were deleted as a result. The distributions of the remaining classes stayed the same after removal of "non-domain" rows, as dataset remained imbalanced. Further on 10 percent, 20 percent -and 10 percent of random rows were sampled from the full datasets for training, validation -and testing respectively. Then rows containing"NaNs" for title, image-URL, id and label were removed, followed by only keeping rows that have ".jpg" as file extension, to keep things consistent. Last step of dataset-cleaning was to extract only the important features, which in this case was title, id, image-URL and the 6-way-label. The dataframes were then written as csv-file to same folder as each sets tsv-file, with the names "train-sample", "validation-sample" and "test-sample" respectively. The dataframes now contained 36.654, 7670 and 3884 rows. These steps were all done in the notebook called "to-clean-sample.ipynb".

3.2 Downloading and Deleting Images

Before downloading the images, there were still a few remaining steps. First the samples were loaded as dataframes in the same script where image-model was built, called "CNN-image.ipynb". The dataframes were sorted by id, in alphanumeric order, before the 6-way-label was converted to a 5-way-label, giving all labels with value 5 (Manipulated Content), the integer value 4, since all class

labels with value 4 (False Connection) were previous deleted. The images were now ready to be downloaded with the image-URL in the following manner:

- A python file with name "image-downloader-2.py" that downloads the images using image-URL, image-id -and class was made.
- This file was copied into the same directory to where one stored the image folder. First into training-data folder, then into validation-folder, and so on.
- The file was run in the terminal, in the same directory where the sample-csv file was stored, and this csv-file was taken as input using "argparse".
- A folder named "images", with sub-folders named after each class labels was made using "os". In other words, the sub-folders was named 0,1,2,3,4 as there were 5 class labels.
- Iterated through every row in dataframe, storing image in following format: "id.jpg", where id was unique id for each image.
- Image was then placed in the sub-folder for its class using the corresponding class label when iterating through the rows. Image with class label 0, was placed in sub-folder "0" under images-folder, an so on.

As all images were downloaded, one had to check they were in fact images and had a format that was compatible with TensorFlow. All rows with images that did not fulfil this criteria was removed, while corresponding image-file was deleted from its folder. This process proceeded as follows:

- All files were loaded and paired with is id in a separate dataframe.
- Ids were sorted in alphanumeric order, so that they matched the dataframe containing title, URL and label.
- The dataframes were merged on id, after resetting index for both, making sure correct title, id, label and image was connected
- A function that returned the file-format for each image was made, and applied on every file in dataframe.
- The images that were not able to open or had a file-format (only GIF, JPEG, PNG and BMP accepted) not accepted by TensorFlow was deleted, while the corresponding rows were removed.

This was the last step of cleaning, and the final dataframes were written as csv-files to their respective folder with the names "train-sample-clean", "validate-sample-clean" and "test-sample-clean". The final shapes for the clean datasets were (34980, 5), (7334, 5) and (3710, 5), where the remaining 5 features were the clean title, id, image-URL, integer class label and the file-path for each image.

The distribution of each class for each final, cleaned dataset is shown in Table 1. With total percentage in parenthesis. As seen, the distribution for all datasets were almost equal, where all of them were imbalanced, containing lots true samples, and very few imposter content samples.

Table 1. Class Distribution of Clean Datasets

Class	Train	Validate	Test
0, True	20375 (58.3)	4295 (58.6)	2232 (60.2)
1, Satire/Parody	2622 (7.5)	546 (7.4)	273 (7.4)
2, Misleading Content	8969 (25.6)	1876 (25.6)	923 (24.9)
3, Imposter Content	915 (2.6)	206 (2.8)	82 (2.2)
4, Manipulated Content	2099 (6.0)	411 (5.6)	200 (5.4)
Total	34980	7334	3710

3.3 CNN with image-input

First step of image classification model was to load the images. This was done using TensorFlow.keras.preprocessing.image-dataset-from-directory. This method automatically detects number of sub-folders within the image folders, where each sub-folder with images is a unique class. It also shuffles the data by default, as well as resizing every image to size (256,256,3), which is important as the model requires all images to be of same shape. Label-mode was set to "categorical", meaning each integer label was on-hot-encoded to an array of length 5. If the class label was 3 it then became [0,0,0,1,0], indicating that the class is the position of the 1. This was done to later on perform categorical cross-entropy. Batch-size was also set to 16, meaning 16 different, random images were trained during each step. The practical thing with this method was that it also converted the image-data to a TensorFlow.data.Dataset-element, which made it possible to construct a pipeline, where computer memory was not consumed until each batch of dataset was grabbed. The image-arrays were all scaled from [0,255] to [0, 1].

The model was then built using Sequential-model from TensorFlow.keras, which made sense as there was a single image-input which yielded a single vector-output. The model consisted of the following layers:

- An input-layer that only accepted images of shape (256,256,3).
- A 2D-convolutional layer with 16 different filters of size (3,3), with step size 1, ReLU as activation function, followed by a Max Pooling layer with kernel size (2,2) and step size 2.
- A second 2D-convolutional layer with 32 different filters of size (3,3), with step size 1, ReLU as activation function, followed by a Max Pooling layer with kernel size (2,2) and step size 2.
- A third and last 2D-convolutional layer, which was identical to the first convolutional layer, followed by an identical Max Pooling layer.
- A flattening layer that converted the multidimensional output from last Max Pooling layer to a 1-D-array.
- A fully connected layer with 64 neurons, using ReLU as activation function.
- The fully connected output layer, with 5 neurons, one neuron for each class label, with softmax as activation function.
- The model was compiled with Adam as optimizer, categorical cross-entropy as loss function and accuracy as metric.

A convolutional filter captures details, and returns a feature map after convolution, which has decreased in dimension if no padding

is used [15]. Each filter captures details of the image sliding a kernel over the image, and checking where the kernel "overlaps" the image [15]. Let's say you have the following kernel in Table 2 sliding over a normalized image (0 is pitch black and 1 is white):

Table 2. (3,3) kernel detecting diagonal line

1	0	0
0	1	0
0	0	1

If a white diagonal line exists in image, sliding kernel over this area will result in a positive output, as it is basically a dot product between the filter and the grid it is overlapping. Using ReLU as activation-function will convert all negative numbers to zero, where positive values remain the same [7]. This speeds up training as there are lots of zeros introduced [7]. The 2D-filter applied on all channels in a 3D-color-image will also reduce dimensionality, as feature map will become 2D. One usually use multiple filters in each convolutional layers, so each filter will give a resulting feature-map, which usually is sparse because of ReLU-function [7]. The positive numbers in these feature maps then correspond to a detail which is similar to filter. A convolution is very often followed by a Max Pooling layer, where most common kernel size is (2,2) with step size of 2 [19]. This will down-sample the image, taking advantage of correlation within an image [19], and only focus on the most important features. It also makes the algorithm more invariant to changes as small rotations and translation, as the focus is on the essential features [19]. It also doesn't require any parameter-tuning, as the method only select maximum number within a region.

The first layer of filters usually detect low-level features, such as edges and corners [16]; that is why 16 filters were used. Second layer might detect features that are combinations of lower-level-features, such as more complex shapes [16], hence 32 filters used in this layer. Last layer with filters might detect even more complex shapes, that are combinations of more complex shapes, but as dimensions were compressed by 2 previous Max Pooling layers, only 16 filters were used. A kernel-size of (3,3) was used for all filters due to the low resolution of input image (256,256), even though larger filters might would have detected more/larger features [16]. Default kernel size of (2,2) used for Max Pooling, as resolution of input image was low, and to compress output by 50 percent in each layer. ReLU used as activation-function in convolutional layers, to make feature map sparse/training faster, and highlight essential features with positive numbers. Output of last Max Pooling was flattened, to fully connect it with a dense layer of 64 neurons. This dense layer was fully connected with output layer of 5 neurons, where softmax was used as activation function to get the probability for each class [7], and only one class as output, not multiple classes as output. Categorical cross-entropy used as loss function, as all outputs are between 0 and 1 [11]. It wouldn't make sense using MSE or similar loss functions, as the gradient would never get very large, as maximum MSE would be very small. "Adam" used as optimizer, as it is computationally efficient and requires little memory [10].

Model was trained for 1 epoch, with 1 random batch at the time, validated on one random batch of validation data.

A summary of calculation of the image-models parameters follow:

- First convolutional layer consists of 16 filters with size (3,3) for each channel, with a bias corresponding to each filter: 16*(3*3)*3 + 16 = 448 parameters.
- Second convolutional layer consists of 32 filters, size (3,3), performed on each of the 16 max pooled feature maps from previous layer, with a bias for each filter: 32*(3*3)*16 + 32 = 4640 parameters.
- Third convolutional layer consists of 16 filters, size (3,3), performed on each of the 32 max pooled feature maps from previous layer, with a bias for each filter: 32*(3*3)*16 + 15 = 4624 parameters.
- Fully connected dense layer with the flattened vector of the output of last max pooled convolutional layer, with bias for each node: 30*30*16*64 + 64 = 921.664 parameters.
- Fully connected dense output layer with previous dense layer, bias for each output node: 64*5 + 5 = 325 parameters.
- Which give a total of: 448 + 4640 + 4624 + 921.664 + 325 = 931.701 trainable parameters.

Final CNN-model with parameters can be seen in Figure 1

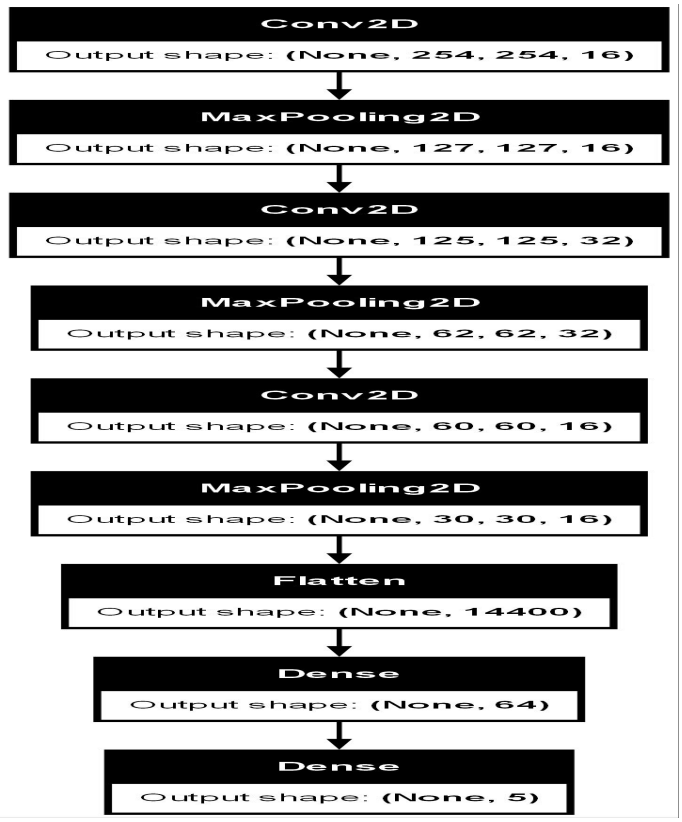

Fig. 1. CNN-model for images, with output-shapes
CNN-model for images, with output-shapes

3.4 BiLSTM with text-input

First, the clean dataframes were imported, checking that no titles were empty, before extracting only the title from the dataframes, as well as extracting 5-way integer label, and converting it to categorical label. The distributions of the length of each title were plotted as histogram, seen i Figure 2

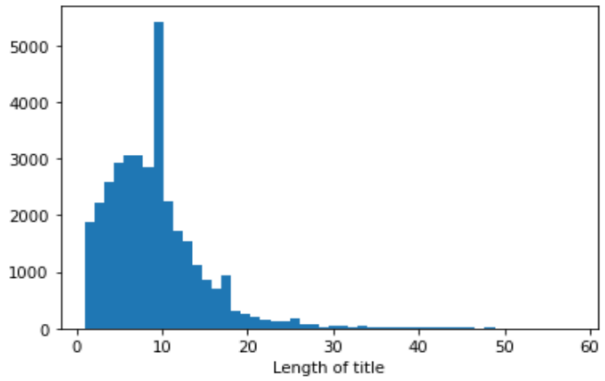

Fig. 2. Distribution of length of titles
Distribution of length of titles

Just looking at the plot, one saw that most title were shorter than somewhere between 15-20 words, and that there was a peak at around 10 words. One calculated the mean of the title-lengths, and added one standard-deviation, and got a length of 14.76. To keep things in power of 2, one set maximum length of title to 16 words, and saw that 92 percent of information was kept, meaning only 8 percent of titles were longer than 16 words.

One then found number of unique words within all titles in training-set, which was 30587 words. A vectorizer using TensorFlow.keras.layers.TextVectorization was made with 16 as max output-length, 2^{15} = 32768 as maximum number of words (to have a unique integer for every word in training-set), and output-mode was set to "int", meaning that for each title, every unique word got an integer signature, and if sentence was longer than 16 words, it got "chopped", and padded with zeros if shorter. More common words got lower integer-values, and unknown words got 1 as integer value. The vectorizer was adapted with training-set.

All datasets were tokenized, and converted to a TensorFlow.data.Dataset with a tuple as input, with tokenized sentences in first position and labels in second position. All datasets were cached, shuffled, prefetched and divided into batches of 16, to match batch size from image-model. Caching and prefetching was done to speed up pipeline during training, with avoiding bottlenecks, making batches faster to access [17].

A sequential model was built and consisted of the following layers:

- An input-layer that only accepted sentences of shape (16,).
- An embedding layer, converting each sentence from shape (16,) to (16,32), giving each word 32 different features.
- Batch-normalization-layer.
- BiLSTM-layer, with output shape of (32,) in each direction, tanh as activation-function, and sigmoid as recurrent activation-function.
- A fully connected layer with 64 neurons, using ReLU as activation function.
- The fully connected output layer, with 5 neurons, one neuron for each class label, with softmax as activation function.
- The model was compiled with Adam as optimizer, categorical cross-entropy as loss function and accuracy as metric.

Maximum unique embeddings were set to 32768+1 = 32769 to account for unknown words (with integer-value 1). Each unique embedding was set to length of 32, to keep parameters down, and not make model too complex, as this was the layer already accounting for most the parameters. Batch-normalization was done to try to prevent overfitting, and make training faster and more stable [5]. Dimension of LSTM-output was set to 32, to match the dimension of the input vector coming from the embedding layer for each word. One used Bi-LSTM, as this is a reccurrent neural network that goes in both directions, which takes advantage of both the beginning and ending of each title, putting every word into context [14]. The fully connected layer got 64 neurons and ReLU as activation-function, to match the corresponding layer in the CNN-model for image. Softmax, categorical cross-entropy, Adam, was chosen for the same reasons as when building the CNN-model for image.

Before showing calculations of number of parameters in model, it is important to understand how an LSTM-unit works. Each cell has 3 inputs: The long term memory, the short term memory and the recurrent input [14]. Both long term -and short term memory are usually initizialed with random values, and in our case, each word in a sentence, as an embedding, was the recurrent input for each cell. Short term memory and each word from the title was concatenated as 1 input-vector, and subject to 4 different set of weights: 1 set to forget gate, 2 to input gate and 1 to output gate. Forget gate tells how much of long term memory to be forgotten based on concatenated input [14]. The input gate tells how much of concatenated input (recurrent input and short term memory) to add to long term memory, and output gate tells how large of a fraction of the long term memory that should go to final short term memory. This is why sigmoid [7] was chosen as recurrent activation function, since it outputs a value between 0 and 1, and is used to determine fraction of inputs to be forgotten or added. Likewise tanh [7] was used as activation function as it outputs values between -1 and 1 for inputs to be scaled by the sigmoid function. In our case, each title had 16 word embeddings, meaning there was 16 recurrent inputs, 1 input in each cell, where the final short term memory of the 16th unit was the output of the LSTM-layer [14]. First word of sentence was first recurrent input, and last word was last recurrent input. Since this was a Bi-LSTM-layer, there were 16 extra cells, running in the opposite direction, where last word of sentence was first recurrent input, and first word was last recurrent input, giving each word in each title more context, based on both long-term -and short-term memory from both future and past, with the total output as concatenation of outputs from both directions (32+32 = 64)[14]. The practical thing with LSTM is that the weights are the same for

all cells running in the same direction, meaning only 2 sets of cells to account for running in both directions [14].

Calculations for total number of parameters below:

- Embedding layer with 32769 unique word embeddings, where each embedding is of length 32: 32679*32 = 1.048.608 parameters.
- Batch-normalization layer: 128 parameters.
- Bi-LSTM-layer with long-term memory of size 32 connected to concatenated recurrent input and short term memory of size 32+32, 4 sets of fully connected layers within each cell, biases for each dense layer and 2 directions: (32*(32+32)*4 + 32*4)*2 = 16.640 parameters.
- Fully connected dense layer with 64 neurons connected with output from BiLSTM with size of 64, including bias for each neuron: 64*64 + 64 = 4160 parameters.
- Fully connected dense output layer with previous dense layer, bias for each output node: 64*5 + 5 = 325 parameters.
- Which give a total of: 1.048.608 + 128 + 16.640 + 4160 + 325 = 1.069.861 parameters, and 1.069.797 trainable parameters.

Model was trained for 1 epoch, in the same manner as the CNN-image-model. Model seen in Figure 3

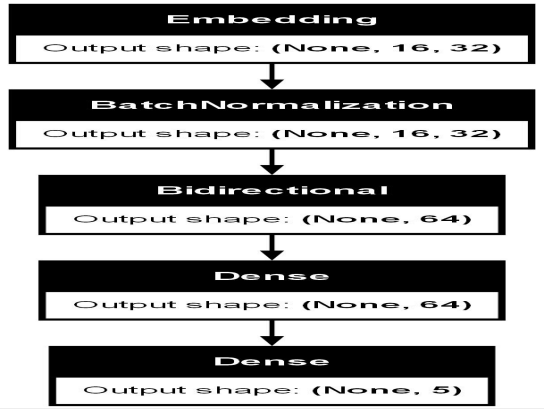

Fig. 3. BiLSTM-model for text-input with output-shapes
BiLSTM-model for text-input with output-shapes

3.5 Multimodal Model with CNN-image -and BiLSTM-text Input Streams

To build and train the multimodal model, one first had to make sure that the text and image that were fed to the model was corresponding, while being in shuffled batches, so there was a few steps involved to reach this point:

- One first made a dataframe that had the ids sorted the same way as the images were loaded. -Starting in sub-folder 0, then after alphanumeric order, an son on.
- Merged dataframe with label, title, file-path, image-URL onto the dataframe with ids in same order as images were loaded.
- Made sorted dataset tokenized titles
- Loaded images as dataset with corresponding categorical labels, with no shuffle, and no batching, to make sure images, labels and titles were all sorted in the same order.
- Extracted image-arrays and labels as separate datasets, before zipping image together with tokenized text as one tuple within another tuple, where labels were the second element. Making a dataset with corresponding multiple input and corresponding label
- Validation -and test-set were then shuffled and put into batches of 16.
- Because of memory-errors training-set had to be split into 5 different shards, shuffling each shard, before concatenating them together to one dataset, and then be put into batches of 16.

This time one could not build a sequential model as the model required different input streams; one for image and one for text. Because of this one build a model with TensorFlow.keras' Functional API, which allows more flexibility. The models for both image and text were exactly the same, with identical hyper-parameters until the output of flattened Max Pooling layer for image and BiLSTM-output for title, before concatenating these vectors together, sending the concatenated vector through a dense layer with 64 neurons and ReLU activation-function, before going into output layer with 5 neurons and softmax as activation-function. Finally the model was built, with two inputs as image and text, with one common output. The multimodal model was trained for 1 epoch, just like the unimodal models. Model with parameters seen in Figure 4

Got predicted labels from model, with testing-dataset as input, iterating through each shuffled batch. -Confusion matrix for all models calculated with TensorFlow.math.confusion-matrix, where actual labels are the rows, and the predicted labels are the columns. Total accuracy for each model calculated from the confusion matrix:

$$\frac{trace}{sumofallmatrixelements}$$

Recall and precision calculated from confusion matrix to get F1-score for each class in each model [3]:

$$\frac{2 \times precision \times recall}{precision + recall}$$

Accuracy used as metric to compare with related work, and F1-score used as metric, as datasets were shown to be heavily imbalanced [3].

4 RESULTS

The resulting confusion matrices for all models seen in Table 3, Table 4 and Table 5.

Table 3. Confusion matrix for unimodal CNN-image model

	0	1	2	3	4
True	2190	5	18	0	19
Satire/Parody	224	12	14	4	19
Misleading Content	854	4	38	2	25
Imposter Content	71	3	4	1	3
Manipulated Content	138	5	12	0	45

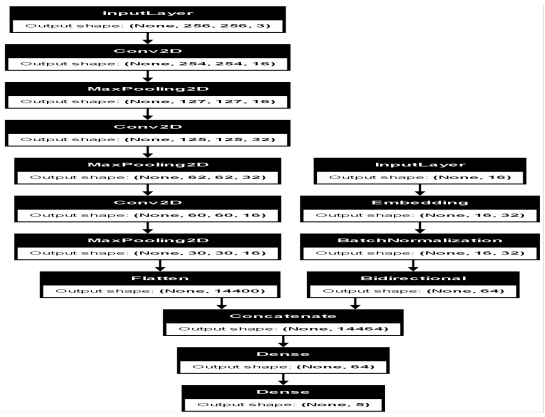

Fig. 4. Multimodal model with output-shapes
Multimodal model with output shapes

Table 4. Confusion matrix for unimodal BiLSTM-text model

	0	1	2	3	4
True	2018	26	165	1	22
Satire/Parody	154	61	48	0	10
Misleading Content	328	23	556	0	16
Imposter Content	57	9	8	1	7
Manipulated Content	63	8	20	0	109

Table 5. Confusion matrix for multimodal model

	0	1	2	3	4
True	2033	63	116	5	15
Satire/Parody	118	117	32	2	4
Misleading Content	325	47	537	3	11
Imposter Content	58	10	9	5	0
Manipulated Content	49	19	11	6	115

Recall, precision for each class in each model seen in Table 6 and Table 7, as well as total accuracy for each model.

Table 6. Recall for each model, as well as total accuracy

	CNN image	BiLSTM text	Multimodal
True	0.98	0.90	0.91
Satire/Parody	0.044	0.22	0.43
Misleading Content	0.041	0.60	0.58
Imposter Content	0.012	0.012	0.06
Manipulated content	0.225	0.545	0.575
Accuracy	0.62	0.74	0.76

F1-scores for each class in each model seen in Table 8

Table 7. Precision for each model, as well as total accuracy

	CNN image	BiLSTM text	Multimodal
True	0.63	0.77	0.79
Satire/Parody	0.41	0.48	0.46
Misleading Content	0.44	0.70	0.76
Imposter Content	0.14	0.50	0.24
Manipulated content	0.41	0.66	0.79
Accuracy	0.62	0.74	0.76

Table 8. F1-score for each class in each model

	CNN image	BiLSTM text	Multimodal
True	0.77	0.83	0.84
Satire/Parody	0.080	0.31	0.44
Misleading Content	0.075	0.65	0.66
Imposter Content	0.022	0.024	0.097
Manipulated content	0.29	0.60	0.67

Looking first at the metrics from the CNN image-model, it performed well on True-class, especially when it comes to recall, saying that the model recognizes true-images very well. Precision did on the other hand not perform too well on True-class, where it predicted an image to be True too often. For the other classes, the model performed quite poorly, especially when it came to Imposter-class Recall was very low for all other classes than True-class, while precision was around 50 percent for all classes except Imposter-class. Total accuracy was 0.62.

Looking metrics from the BiLSTM text-model, it performed as image-model well on True-class. A bit lower than image-model regarding recall, but higher regarding precision. For the other classes, this model also performed quite poorly, but precision was higher for all classes compared to CNN-model, and around or above 50 percent for all classes. Recall was also higher for all other classes than True-class, except for Imposter-class, where it performed just as poorly as with the images. Total accuracy was 0.74.

The multimodal model increased the accuracy from unimodal text-model by just a little, to 0.76. It increased recall for all classes compared to the unimodal models, except for True-class and Misleading-class, where CNN-model and BiLSTM-model had higher recall respectively. It also increased precision for most classes compared to the unimodal models, but not for Satire-class, and definitely not for Imposter-class, where precision decreased from 0.5 to 0.24 going from unimodal text-model to the multimodal. F1-scores were higher for all classes when predicting with multimodal model.

5 DISCUSSION

As seen in the results, none of the models performed very well, especially not the image-model, where it adds little or nothing to the unimodal text-model, as total accuracy only increased by 0.02 going from unimodal text to multiple input streams. Looking at the F1-score for each class, the True-class is the only class that performs somewhat well, with F1-score above 0.8. Comparing the

F1-scores in Table 8 with the distributions of each class in Table 1 one can for sure see a pattern: The classes with the most training data has the highest F1-scores/performance, and vice versa. Imposter content has under 1000 data-points in training-set, which might is a reason why the models perform so poorly on that particular class. The image-model also performed surprisingly poorly on the manipulated content, as some of the related work achieved exactly 100 percent accuracy for this class with a multimodal CNN-model [13]. But then the whole dataset was used for training, so it might explain some of the difference, as in the sub-sample trained on in this report only contained around 2000 data-points for this class. One could also not exclude the fact that the models might have been too complex, with too many parameters, which could lead to overfitting the training data [2].

The reason one only trained the models for 1 epoch was because all models showed sign of overfitting. In other words, the accuracy for the training-data converged very fast to 1, but the validation-accuracy either remained unchanged, or decreased. There is reason to suspect one reason for this was that the training-data-size was too small, and training with more data would lead to less overfitting [2], and better performance overall. The models might have been too complex as well, with too many trainable parameters, which could lead to the model learning the noise of the training-data, and not just the essential part of it [2]. To avoid overfitting one could also have added dropout-layers, which would exclude a fraction of the training-data for each training-step [12], but then again one would have had to train the model for multiple epochs.

6 CONCLUSIONS

First key part of this report was to evaluate with the training of a 10 percent sub-sample of Fakeddit-dataset, if the multimodal model increased performance compared to unimodal models trained with the same hyper-parameters. It did, but not by much, with 0.02 higher total accuracy than the unimodal text-model. Second part was to evaluate if the increase of total accuracy using a multimodal model was the same when training the model with just 10 percent of the entire dataset, and that was not the case, as some of the related work that trained and validated the model with the full Fakeddit-dataset both increased performance by around 9 points [9], [13], where in this case the increase was only by 2 points. All unimodal models, as well as the multimodal model also performed poorer than the related work that used the full dataset [9], [13]. Many factors might have played a role in this, but it is clear that the lack of training-data has made all the models constructed in this report perform poorer, and overfit to training-data.

REFERENCES

[1] Christina Boididou, Katerina Andreadou, Symeon Papadopoulos, Duc Tien Dang Nguyen, Giulia Boato, Michael Riegler, Yiannis Kompatsiaris, et al. 2015. Verifying multimedia use at mediaeval 2015. In *MediaEval 2015*. Vol. 1436. CEUR-WS.

[2] Tom Dietterich. 1995. Overfitting and undercomputing in machine learning. *ACM computing surveys (CSUR)* 27, 3 (1995), 326–327.

[3] Margherita Grandini, Enrico Bagli, and Giorgio Visani. 2020. Metrics for multi-class classification: an overview. *arXiv preprint arXiv:2008.05756* (2020).

[4] Melanie C. Green and John K. Donahue. 2018. *SIX The Effects of False Information in News Stories.* University of Texas Press, New York, USA, 109–123. https://doi.org/doi:10.7560/314555-008

[5] Liu Jian-Wei, Zhao Hui-Dan, Luo Xiong-Lin, and Xu Jun. 2020. Research progress on batch normalization of deep learning and its related algorithms. *Acta Automatica Sinica* 46, 6 (2020), 1090–1120.

[6] Sakshi Kalra, Chitneedi Hemanth Sai Kumar, Yashvardhan Sharma, and Gajendra Singh Chauhan. 2022. Multimodal Fake News Detection on Fakeddit Dataset Using Transformer-Based Architectures. In *Machine Learning, Image Processing, Network Security and Data Sciences*, Nilay Khare, Deepak Singh Tomar, Mitul Kumar Ahirwal, Vijay Bhaskar Semwal, and Vaibhav Soni (Eds.). Springer Nature Switzerland, Cham, 281–292.

[7] Serhat Kılıçarslan, Kemal Adem, and Mete Çelik. 2021. An overview of the activation functions used in deep learning algorithms. *Journal of New Results in Science* 10, 3 (2021), 75–88.

[8] Uğur Mertoğlu and Burkay Genç. 2020. Automated Fake News Detection in the Age of Digital Libraries. *Information Technology and Libraries* 39, 4 (Dec. 2020). https://doi.org/10.6017/ital.v39i4.12483

[9] Kai Nakamura, Sharon Levy, and William Yang Wang. 2020. r/Fakeddit: A New Multimodal Benchmark Dataset for Fine-grained Fake News Detection. arXiv:1911.03854 [cs.CL]

[10] Michael Pérez. [n. d.]. An Investigation of ADAM: A Stochastic Optimization Method. ([n. d.]).

[11] Yagyanath Rimal. 2020. Multi-layer neural network perception for large data prediction using R Programming. *CENTRAL ASIAN JOURNAL OF MATHEMATICAL THEORY AND COMPUTER SCIENCES* 1, 11 (2020), 17–21.

[12] Claudio Filipi Gonçalves Dos Santos and João Paulo Papa. 2022. Avoiding overfitting: A survey on regularization methods for convolutional neural networks. *ACM Computing Surveys (CSUR)* 54, 10s (2022), 1–25.

[13] Isabel Segura-Bedmar and Santiago Alonso-Bartolome. 2022. Multimodal Fake News Detection. *Information* 13, 6 (2022). https://doi.org/10.3390/info13060284

[14] Ralf C Staudemeyer and Eric Rothstein Morris. 2019. Understanding LSTM–a tutorial into long short-term memory recurrent neural networks. *arXiv preprint arXiv:1909.09586* (2019).

[15] Mohammad Mustafa Taye. 2023. Theoretical Understanding of Convolutional Neural Network: Concepts, Architectures, Applications, Future Directions. *Computation* 11, 3 (2023). https://doi.org/10.3390/computation11030052

[16] Mohammad Mustafa Taye. 2023. Theoretical understanding of convolutional neural network: Concepts, architectures, applications, future directions. *Computation* 11, 3 (2023), 52.

[17] Steven P Vanderwiel and David J Lilja. 2000. Data prefetch mechanisms. *ACM Computing Surveys (CSUR)* 32, 2 (2000), 174–199.

[18] Bin Wang, Yong Feng, Xian-cai Xiong, Yong-heng Wang, and Bao-hua Qiang. 2023. Multi-modal transformer using two-level visual features for fake news detection. *Applied Intelligence* 53, 9 (2023), 10429–10443.

[19] Afia Zafar, Muhammad Aamir, Nazri Mohd Nawi, Ali Arshad, Saman Riaz, Abdulrahman Alruban, Ashit Kumar Dutta, and Sultan Almotairi. 2022. A comparison of pooling methods for convolutional neural networks. *Applied Sciences* 12, 17 (2022), 8643.

[20] Dimitrina Zlatkova, Preslav Nakov, and Ivan Koychev. 2019. Fact-Checking Meets Fauxtography: Verifying Claims About Images. arXiv:1908.11722 [cs.CL]

Received 24 April 2024