Is this the real life or just fantasy?

Miguel Marques[2017266263]1 and Tiago Fernandes[2017242428]1

Universidade de Coimbra, Coimbra, Portugal
{mpmarques,tfernandes}@student.dei.uc.pt

Abstract. In this article we will talk about our approach on how to analyse and predict whether clothes are real or fake. Let's start by talking about how we split the data and what pre-processing we did. Then we'll talk about two approaches that we decided to implement in order to solve the problem. The first approach was using Ensembles where we got an accuracy of 0.912 and in the second approach we decided to use deep learning with CNNs where we got an accuracy of 0.997.

Keywords: Fake News · Ensembles · CNN

1 Problem Description

For this project, we were provided with a well-known dataset, the **Fashion MNIST** with some modifications and in our case, its purpose is different: its purpose is to come up with some method that can distinguish between real and fake images.

2 Experimental preparation

The images were provided to us were divided into two folders with training data further divided into false (20000) and true (20000) images and another folder with unlabelled images (training data with 2000 images).

To ensure that they were able to evaluate the implemented models, we divided the training data into 3 groups (training - 60%, validation - 20%, test - 20%) (Figure 1).

3 Ensembles

In this part we will cover how we developed the Ensemble. To do this, we first had to extract the image features as we didn't think it made sense to send the **raw data** to the models. [1]

We extract the following features:

– **Hu Moments** [2]: image moments are a weighted average of image pixel intensities. The Hu Moments are related to translation, scale, rotation and reflection.

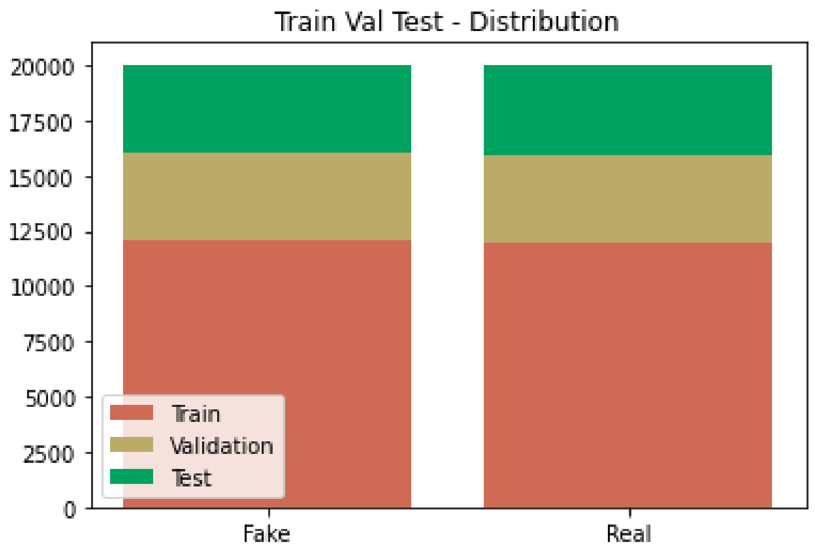


Fig. 1. Train, Validation, Test - TVT

– **Normalised colour histogram**: for each grayscale value, we count how many pixels are in that value, and store them in an array

Once extracted, we stack them and they are ready to be used by the models. For the Ensemble, we used the following models:

– **Logistic Regression**: The training algorithm uses the one-vs-rest (OvR) scheme with cross-entropy loss function. The maximum number of iterations taken for the solvers to converge is set to 100.
– **Decision Tree Classifier**: We used the default parameters of the SKLearn's implementation.
– **Random Forest Classifier**: We used a random forest with 100 trees.
– **KNeighbours Classifier**: We used a K-Neighbours with K equal to 5.
– **Linear Discriminant Analysis**: We used Singular Value Decomposition and does not compute the covariance matrix, therefore this solver is recommended for data with a large number of features.
– **Gaussian Naive Bayes**: We used the default parameters of the SKLearn's implementation.
– **Support Vector Machines**: We used a Radial Basis Function Kernel with degree equal to 3.

To calculate the weights we decided to do a 10 Fold Cross Validation with the training set and extract the average accuracies of the 10 folds. These were the accuracies that we got:

– **Logistic Regression** - 0.759

- **Decision Tree Classifier** - 0.960
- **Random Forest Classifier** - 0.973
- **KNeighbors Classifier** - 0.754
- **Linear Discriminant Analysis** - 0.777
- **Gaussian Naive Bayes** - 0.625
- **Support Vector Machines** - 0.732

As we can see the **Decision Tree** and **Random Forest** obtained the best results while **Gaussian Naive Bayes** obtained the worst ones, so, to reward the models that obtained better accuracies over the ones that obtained worse accuracies (Figure 2), we decided to test the accuracies value raised to the power of 1, 2, 3, 4 and 5 as weights. In the end, the accuracies raised to the power of 1 and 2 gave the same results and raised to the power of 3, 4 and 5 gave the same results as well. In the end, we chose to raise to the power of 3.

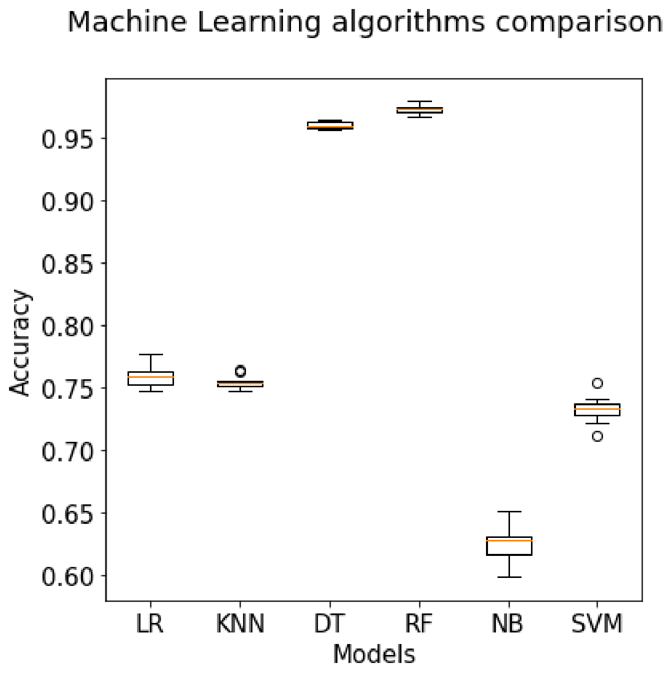


Fig. 2. Accuracy of the models tested for emsemble

For the Ensemble, we used a voting system, where each algorithm adds its weight to the option of the image being real or fake. Finally, the option with the highest score is chosen.

We trained all models with the **training set** and these were the results we obtained in the **validation set**:

Results' of the different					
Models	**Accuracy**	**Precision**	**Recall**	**AUC**	**F1-Score**
LR	0.763125	0.618644	0.871795	0.787440	0.723721
LDA	0.776875	0.713858	0.818052	0.781442	0.762412
KNN	0.756625	0.589980	0.886849	0.789138	0.708577
DT	0.957875	0.953639	0.962032	0.957899	0.957817
RF	**0.975250**	**0.988784**	0.962864	**0.975633**	**0.975652**
NB	0.624125	0.361665	0.764892	0.672631	0.491115
SVM	0.742125	0.510219	0.954312	0.809351	0.664934
Emsemble	0.921500	0.856680	**0.984814**	0.928660	0.916289

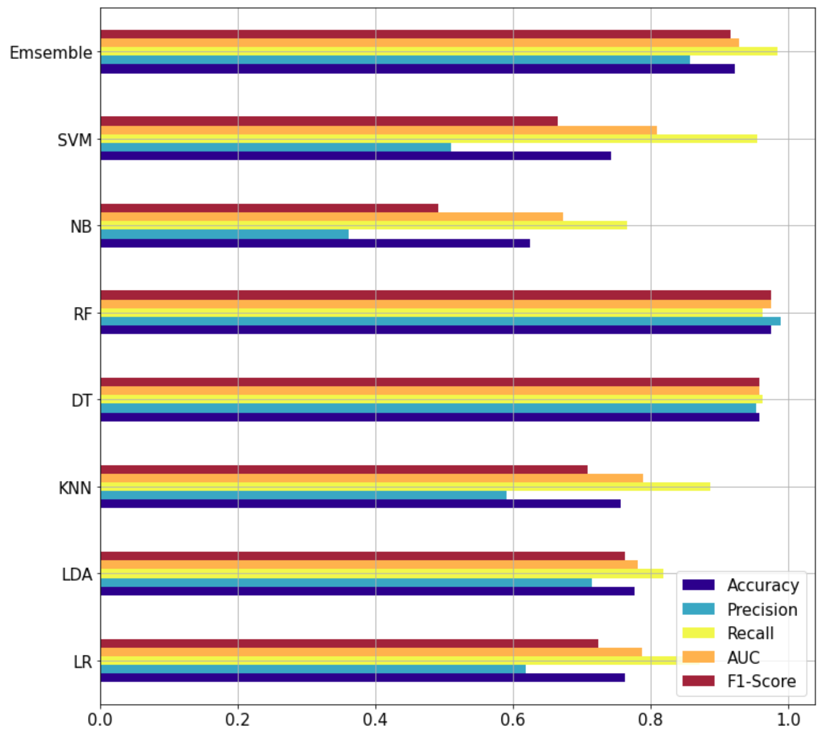


Fig. 3. Validation Results

As we can conclude, Random Forest obtains the best results for all the metrics tested except for Recall, where the Ensemble obtains the best results (Figure 3).

Finally, we tested the **Ensemble** model for the test set and obtained the following results for the metrics presented previously (Figure 4).

– **Accuracy** - 0.919750
– **Precision** - 0.856190

- **Recall** - 0.982979
- **AUC** - 0.926461
- **F1-Score** - 0.915214

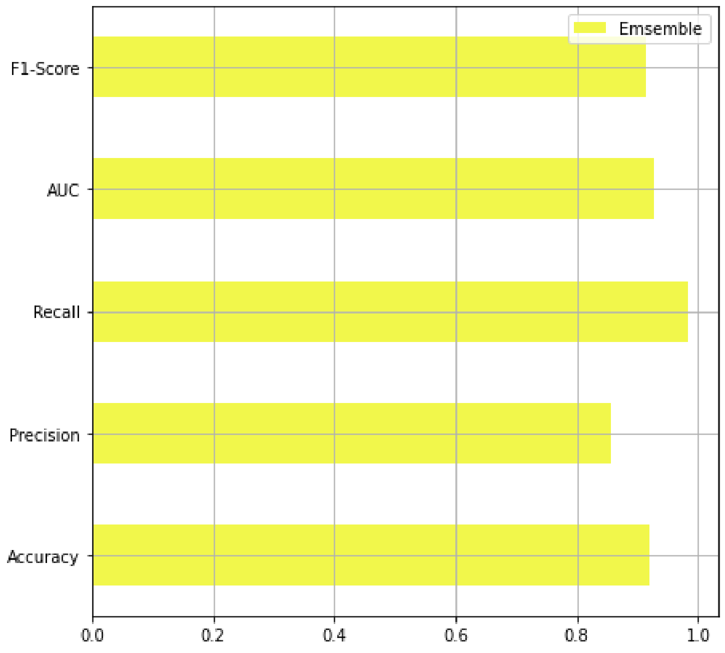


Fig. 4. Test Results

4 CNN Models

4.1 Transfer Learning

One of the methods we tried to adopt to solve our problem was transfer learning, which means using models already created and with good results to solve our problem. The algorithms tested were Xception, InceptionV3 (GoogleNet) and VGG16. All these models receive images larger than ours and with 3 colour channels (RGB instead of grayscale), therefore, there was a need to pre-process the images to convert them to the correct formats. For that, when reading the images, we started by creating 3 layers with the same value of the image in grayscale (i.e., if a pixel has 234 in grayscale, it will be converted to 234 in R, 234 in G and 234 in B) and resizing the 28x28 images to the sizes desired by those networks (Xception - 71x71, InceptionV3 - 75x75 and VGG16 - 32x32), using the nearest neighbour algorithm. To train the models, we resorted to TensorFlow's training-validation-test split (tf.keras.preprocessing.image_dataset_from_directory) and the models had accuracy values around 50% for the validation group, so we simply gave up using these approaches.

4.2 CNN Testing

During the first random trials of a CNN implemented by us, using Kaggle, we were able to obtain a model with 99.7% accuracy, so from that moment on, we only started testing parameters for this model (Figure 5).

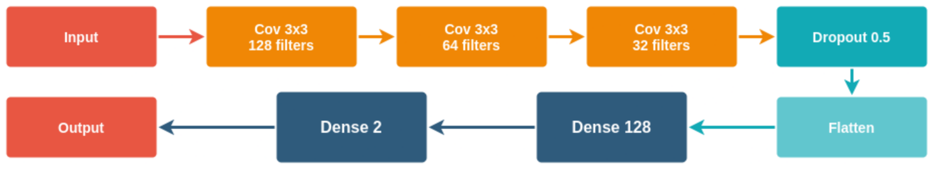


Fig. 5. Architecture of initial CNN

The initial model had the following structure:

1. 2D Convolution Layer with filter size 128, convolution window 3x3, reLU as activation function and input size 28x28x1
2. 2D Convolution Layer with filter size 64, convolution window 3x3, reLU as activation function
3. 2D Convolution Layer with filter size 32, convolution window 3x3, reLU as activation function
4. Dropout layer with 0.5 rate
5. Dense layer with 128 neurons and reLU as activation function
6. Dense layer with 2 neurons and softmax as activation function

On the figures 6 and 7, we can see the evolution of accuracy and loss on the 100 epochs of the model, and some metrics that agrees with our tests.

The parameters tested were the activation functions of the various layers (all Conv2D layers and 1st Dense layer, 2nd Dense layer) and the value of the dropout layer. The activation functions tested were: reLU, softmax, tanh and sigmoid (in the Cov 3x3 and Dense Layers). The values tested for the dropout layer are 0.5, 0.6, 0.7 and 0.8 . [3] All combinations of these functions with the dropouts were tested. The values of the metrics were evaluated based on the training set of images from the initial training set created by us (Table 1). The Type's notation in the table means: *(Activation Function in the Cov 3x3)_(Activation Function in the final Dense Layer)_(DropOut Value).*

From the table 1, we can conclude that for this CNN configuration, the relu and softmax/sigmoid activation functions are the best and the most frequent dropout value is 0.8, however, the best results appear when this value is 0.5. It

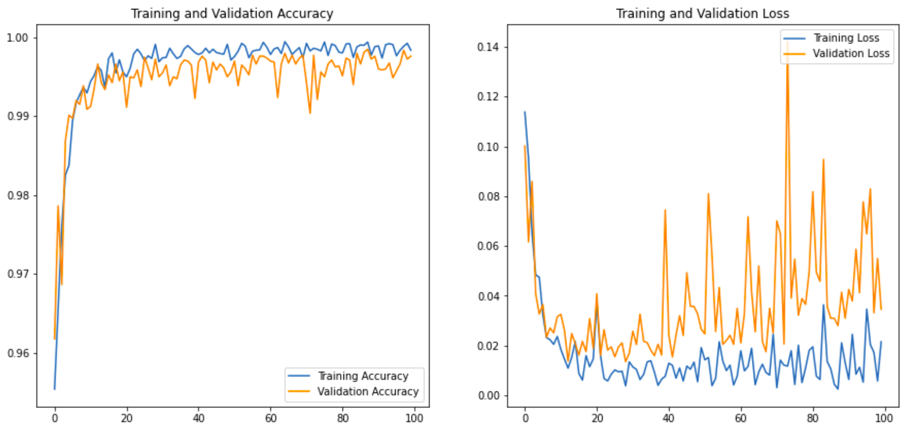
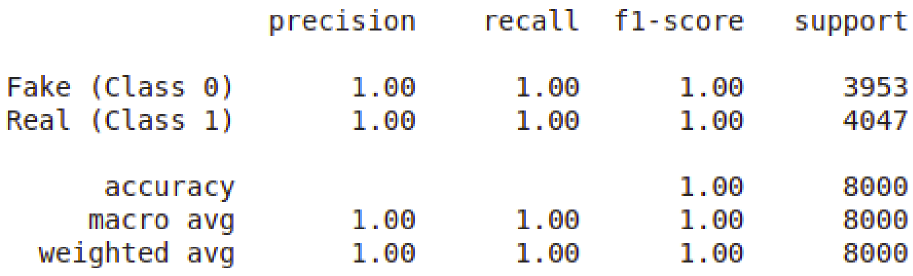


Fig. 6. Accuracy Train vs Validation per epoch



```
                precision    recall  f1-score   support

Fake (Class 0)       1.00      1.00      1.00      3953
Real (Class 1)       1.00      1.00      1.00      4047

      accuracy                           1.00      8000
     macro avg       1.00      1.00      1.00      8000
  weighted avg       1.00      1.00      1.00      8000
```

Fig. 7. Best Model test results

Type	**Accuracy**	**Precision**	**Recall**	**AUC**	**F1-Score**
relu_softmax_0.5	**0.998750**	0.999501	**0.998006**	**0.998752**	**0.998753**
relu_sigmoid_0.5	0.998625	**1.000000**	0.997258	0.998629	0.998627
relu_softmax_0.6	0.998000	0.998752	0.997258	0.998002	0.998004
relu_softmax_0.8	0.994125	0.994512	0.993769	0.994126	0.994140
tanh_sigmoid_0.8	0.968875	0.974049	0.963609	0.968891	0.968801
tanh_tanh_0.8	0.959875	0.965213	0.954387	0.959892	0.959769
tanh_sigmoid_0.7	0.940500	0.901636	0.989282	0.940353	0.943428
relu_sigmoid_0.8	0.921500	0.997940	0.845214	0.921730	0.915250
relu_tanh_0.7	0.913875	0.939434	0.885344	0.913961	0.911587
relu_tanh_0.5	0.907500	0.905955	0.910020	0.907492	0.907983

Table 1. First 10 results of the tests

is important to mention that using a higher dropout value means that we need more training epochs to reach the same accuracy as if we didn't use dropouts, but avoids **overfitting**.

5 Conclusion

For our problem, we had small images and their were in a grayscale model, which means our model were easy to train, because if we had bigger images and in RGB model, we would need more time to train the models.
The Transfer Learning models we used didn't had the expected success because the models we tested only received images in RGB models and not grayscale, and we needed to resize the images, thus "dispersing" information and the results were poor.
Between the ensemble and the CNN model we created, the results are similiar but if the images were bigger and in RGB, the CNN models would give better results.

References

1. Image Classification using Python and Scikit-learn, https://gogul.dev/software/image-classification-python. Last accessed 5 May 2021
2. Shape Matching using Hu Moments (C++/Python), https://learnopencv.com/shape-matching-using-hu-moments-c-python/. Last accessed 15 May 2021
3. A Gentle Introduction to Dropout for Regularizing Deep Neural Networks, https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. Last accessed 10 May 2021