Costa Rican Household Poverty Level Prediction with Deep Learning

David Jansen

Groningen, Netherlands

Abstract

In this report, the performance of a Deep Neural Network (DNN) is compared against a Random Forest on the Costa Rican Household Poverty Level Prediction dataset using a macro F1 score. The DNN performs very poorly on the dataset compared to the Random Forest model. Issues regarding the performance of the model may have to do with the dataset. The dataset had a very low variance for over 110 of the 142 available features, which made it hard for the network to properly learn how to separate the data. Suggestions for improving the model include more extensive pre-processing of the data.

Keywords: Deep learning, LaTeX, Prediction, Multi-class, Classification, Kaggle

1. Definition

1.1. Project Overview

Accurately assessing social needs to ensure the poorest people our planet get the help they need is a difficult task. The Inter- American Development Bank (IADB) is an organization which focuses to improve the lives of those who live in Latin America and the Caribbean. Publications made in December 2016 by the IADB reveal that in 2015 between 8.2% for Chili to 68.7% of the total population of those countries live of less than 5 USD a day. Extreme cases of poverty range between 2.7% for Chili and a shocking 32.6% for Guatemala, where the income a day is less than 3.1 USD. In a study in 2015 a proposition was made to apply machine learning to poverty targeting (McBride and Nichols [1]). The study revealed that they were able to improve on the then standardized method for targeting applied by the USAID by 2 to 18 percent using a Random Forest algorithm. For this reason, the

IADB is looking for new ways to reach people who are in need of help. They have reached out to the Kaggle community in order to find new ways to help identify vulnerable households who may need help.

The IADB has supplied Kaggle community members with a labeled dataset which contains datapoints of individuals and households which were scored and labeled according to a Proxy Mean Test (See 1.2). The dataset is freely available from Kaggle for registered members (URL: `https://www.kaggle.com/c/costa-rican-household-poverty-prediction/data`)

1.2. Problem Statement

In Latin America, a Proxy Means Test (PMT) is used to asses the level of need a household needs. Despite this assessment being an improvement, a need for a model which more accurately classifies these households is present. The IADB has asked to Kaggle Community to create such a model. They have provided a dataset containing multiple characteristics of a Costa Rican household.

In order to predict the correct label of the data, a prediction model in the shape of a Deep Neural Network (DNN) will be created using the training dataset as a base. In preparation of training the model, the dataset will first be preprocessed. This will include resolving any missing datapoints and deleting features with low variance. After the preparation, a Random Forest algorithm will be created in order to create a competitive benchmarking model for the DNN. The resulting macro F1 score (See 1.3) of the Random Forest algorithm will be used as a measurement for developing and improving the DNN. The macro F1 score of the DNN will be used for evaluation. Realistically, the quality of the DNN depends heavily on the quality and separability of the data. It will prove challenging to create a DNN which can compete with a Random Forest algorithm.

1.3. Metrics

For this project, a predefined metric was given. A macro F1 score will be used to determine the accuracy of the model. A F1 score is defined as a harmonic mean of precision and recall (Yutaka [2]).

$$\text{F1-score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

This approach is not usable for a multi-class classification problem. The method chosen by the competition host is a variation of that called a macro-F1 score. It uses the F1 scores of all classes and divides it by the total number of possible classes as a resulting score.

$$\text{macro F1} = \frac{F1Class1+F1Class2+...+F1ClassN}{N}$$

An important note added by the Competition host is that **only the heads of household are used in scoring**. All household members are included in test + the sample submission, but only heads of households are scored.

2. Analysis

2.1. Data Exploration and Visualisation

There are two files the IADB has provided. A training dataset containing multiple features including a target label feature and a test set containing the same features, but without the target label. The datasets respectively contain approximately nine thousand and twenty-four thousand data points. Due to the nature of the Kaggle competition, external data is not allowed. The dataset contains the following core data fields (from Kaggle):

- Id - a unique identifier for each row.
- Target - the target is an ordinal variable indicating groups of income levels.
  - 1 = extreme poverty
  - 2 = moderate poverty
  - 3 = vulnerable households
  - 4 = non vulnerable households
- idhogar - this is a unique identifier for each household. This can be used to create household-wide features, etc. All rows in a given household will have a matching value for this identifier.
- parentesco1 - indicates if this person is the head of the household.

The dataset contains a total of 143 columns. An initial look at the data reveals that the data points are heavily skewed towards non-vulnerable households which are overrepresented in the data. Due to this class imbalance, care has to be taken when splitting the data for training and crossvalidation (Figure 1). This class imbalance is also present in the heads of household (Figure 2). Both oversampling and undersampling were experimented with in order to tackle this issue (See 3.2.4)

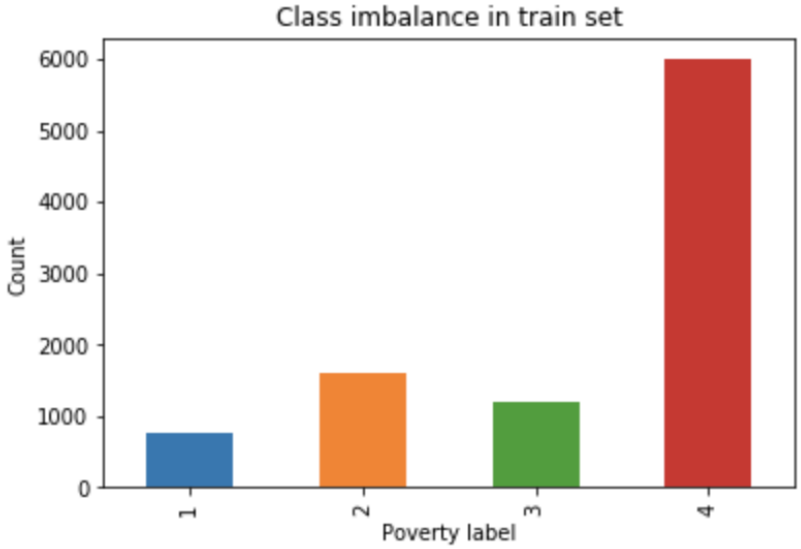

Figure 1: Class imbalance in training data

Another issue is invalid / missing data. This data needs to be cleaned up before it can presented to the prediction models (Figure 5).

Furthermore, due to only the heads of household being considered relevant by the competition host, both the benchmarking Random Forest model and the DNN are trained using only data from heads of household. The dataset contains a feature which labels an individual as a head of household called 'parentesco1' which is used throughout the project to subset the data (after pre-processing).

2.2. Algorithms and Techniques

This project features a Deep Neural Network for predicting the label of the data. A neural network is based on the idea of how neurons work in the brain. A neuron receives, processes and transmits information from and

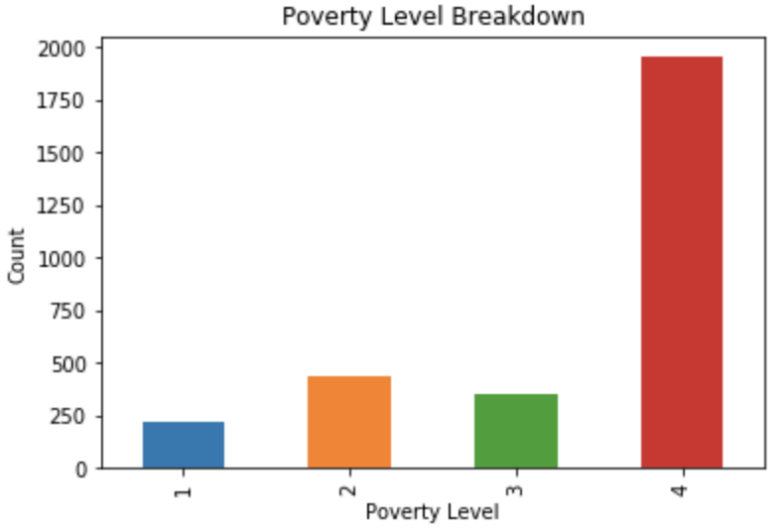

Figure 2: Class imbalance for heads of household in training data

to other neurons. The input for a neuron is received on the dendrites, the cell body processes information and the axon transmits information (output). A neural network process information in a similar way. It consists out of multiple perceptrons which receive, processes and transmits information from and to other perceptrons. It consists out of an input layer, a hidden layer and an output layer (Figure 3). A *Deep* Neural Network differs itself from a regular neural network by having several more hidden layers (Figure 4). Neural networks are known to excel at (among others) image recognition and natural language processing, creating super-human levels of performance and vastly improving on more traditional machine learning models. Problems with DNN's consists of it lacking a lot of theoretical foundation other algorithms do have, it's need for having a lot of data in order to perform, it being computational expensive and it being very hard to determine what is going on in the hidden layers of a DNN. Due to the possibilities of a DNN, it could outperform the proposed benchmarking model (See 2.3) and was therefor chosen as a model to experiment with.

Note about the macro F1 score: One could argue that using a macro F1 score is not the best choice for measuring. In python's sklearn package, a method for implementing the macro F1 score is available, however it states the following: "Calculate metrics for each label, and find their unweighted

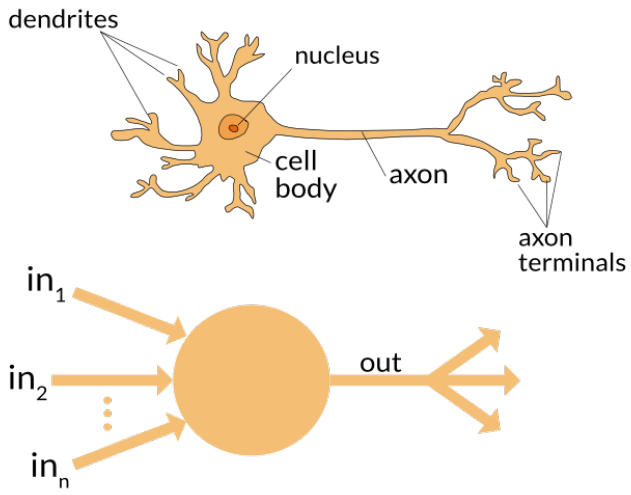

Figure 3: From:https://appliedgo.net/perceptron/ Neurons vs Perceptrons

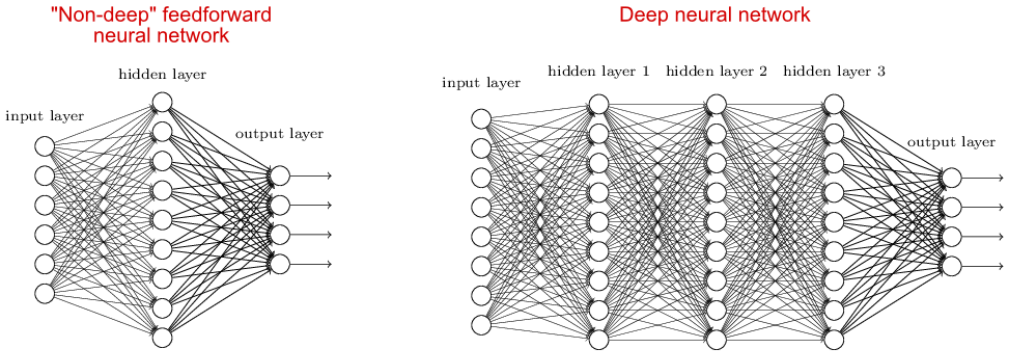

Figure 4: From:http://neuralnetworksanddeeplearning.com/chap5.html Neural Network vs Deep Neural Network

mean. **This does not take label imbalance into account**.". Due to the dataset having a label (or class) imbalance, this forces somehow getting rid of the class imbalance so the resulting measure can be more meaningful. The measure however is set by the competition host, and cannot be changed.

2.3. Benchmark

For benchmarking, a Random Forest classifier was chosen to compete the neural network. In order to create a semi-competitive model, Grid Search

Dataset	macro F1 score
Training	0.9299
Cross-validation	0.4153
Testing	0.3880

Table 1: macro F1 score for benchmarking model

was employed to select some of the hyper parameters. The resulting macro F1 score looked promising, although the model does show signs of overfitting (Table 1). After applying Grid Search, the hyperparameters *min_samples_leaf* and *min_samples_split* were set to 2 and 10 respectively. All other hyperparameters remained the same.

3. Methodology

3.1. Data Preprocessing

Inspection of the testdata revealed four issues which needed to be addressed. Firstly, not all columns of the data were numerical fields, which is a requirement for both the benchmarking model and the deep neural network (3.2.1). Secondly, the data contained datapoints which were null. These datapoints needed to be corrected (3.2.2). Thirdly, some columns revealed to have no variance in the data. These columns have been removed from the dataset (3.2.3). Finally, there is class imbalance in the dataset. Label balancing is employed in order to reduce the presence of the overrepresented class (3.2.4)

3.2. Implementation

3.2.1. Converting object to numerical

Inspection revealed that the datasets contained 5 columns which were non-numerical (Table 2)

Both the 'Id' and the 'idhogar' column were not converted, due to the values not being useful for this particular prediction problem. In the description of both 'edjefe' and 'edjefa' could be found that yes and no should be

column name	description
Id	rowId
idhogar	Household level identifier
dependency	fraction of household members who are dependent on non-dependent
edjefe	years of education of male head of household
edjefa	years of education of female head of household

Table 2: Object types in the dataset

1 and - respectively. These two columns have been remapped and converted to int32. The 'dependency' column contained the same type of data ('yes' and 'no') and was remapped aswell. This column however contained floating point integers and was therefor not converted to int32 but float64 instead.

3.2.2. Correcting null values

A method in python is used to inspect a given dataframe to reveal any missing (null) values (Listing 1)

Listing 1: Analyze dataset for missing values

```
def get_missing(df):
    """
    Analyze missing values in dataframe:
    """
    # Number of missing in each column
    missing = pd.DataFrame(df.isnull().sum()).rename(columns = {0: 'tota
    # Create a percentage missing
    missing['percent'] = missing['total'] / len(df)
    # Drop columns where everything is present
    missing = missing[missing.total != 0]
    return missing
```

In Figure 5 the missing datapoints can be seen.

For each of the feature, different measures were taken to clean the data. rez_esc which represents the years behind in school was only collected for individuals who's age ranged between 7 and 19 years. The missing datapoints seemed to mostly exist out of this age range and were corrected to 0. Missing data was also present for individuals of age 18 (and a single one of age 10). The individuals of age 18 were corrected by getting the mean rez_esc for the nearest age group (age 17). The individual of age 10 was corrected by using the mean of the same age group. v18q1 which represents the number

of tablets a household owns was corrected by setting all the values to 0. This is done because another feature called v18q which represents whether a household has a tablet or not, revealed that for every v18q = 0, v18q1 was missing. v2a1, which represents the monthly rent payment, was corrected in two stages. Firstly, there are accompanying features which have to do with the type of household payment there is. The first correction taken place was to set v2a1 to 0 for every household which fully owns its house. Secondly, the other missing datapoints were all grouped by two other household payment type markers (tipovivi4 and tipovivi5). Due to both of these markers not having any rent data associated with it, it was unknown what the actual value the coresponding v2a1 should have. This is also confirmed by the competition host on the discussion board. Due to this, the v2a1 was set to 0 for both these household payment type markers. Finally the 5 missing datapoints for meaneduc (and its squared accompanying feature SQBmeaned), which represents the average years of education for adults in the houshold was corrected by setting it to the average years of education of the individual. This value was squared for SQBmeaned.

	total	percent
rez_esc	7921	0.830816
v18q1	7319	0.767674
v2a1	6843	0.717747
meaneduc	5	0.000524
SQBmeaned	5	0.000524

Figure 5: Missing datapoints

3.2.3. Removal of columns correction

The python *sklearn* library comes with a class which can help removing features with low or no variance. The implementation of the class set the variance threshold to 1.0, dropping features which have a lower variance.

Listing 2: Using VarianceThreshold to remove features

```
from sklearn.feature_selection import VarianceThreshold
```

```
sel = VarianceThreshold(threshold=1.0)
sel = sel.fit(X)
remaining_columns_ix = sel.get_support(indices=True)
X_train_transformed = sel.transform(X)
test_transformed = sel.transform(test_drp)
```

Utilizing this class the dataset was downsized to only 28 columns. It decreased the amount of overfitting done by the Random Forest benchmark model by 2%

3.2.4. Balancing of the dataset

Initial analysis of the data revealed that the data was heavily skewed due to one of the classes being overrepresented in the data. The training dataset reconfirms this finding (Figure 2).

Utilizing another python library called *imblearn*, the data was transformed into a balanced set (Listing 3;Figure 6).

Listing 3: Balance dataset in python

```
from imblearn.over_sampling import RandomUnderSampler

smp = RandomUnderrSampler(random_state=42)
X_res, y_res = smp.fit_sample(X, y)
```

3.3. Refinement

As stated, a deep neural network was created in order to compete the benchmarking model. The earliest model consisted of a total of four layers, an input layer, two hidden layers and an output layer (Listing 4).

Listing 4: Earliest Deep Neural Network

```
model = Sequential()

model.add(Dense(64, input_dim=X_train.shape[1], activation='relu'))
model.add(Dropout(0.20))
model.add(Dense(32 ,activation='relu'))
model.add(Dropout(0.20))
model.add(Dense(32 ,activation='relu'))
model.add(Dropout(0.20))
model.add(Dense(y_train_encoded.shape[1], activation='relu'))

model.summary()
```

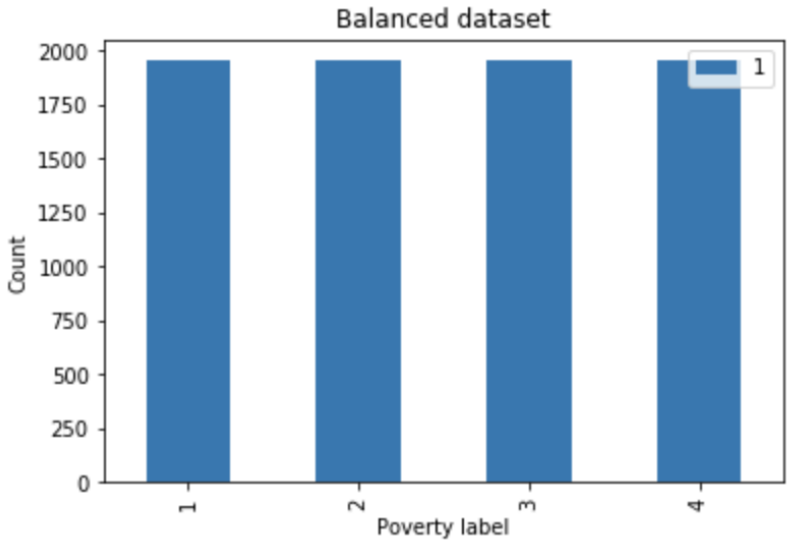

Figure 6: Training data after balancing

The model was created using ways to reduce overfitting a bit by adding dropout layers[3] and using a ReLU activation function.

The first finding was, that this somehow caused unusual output. Specifically, the outputs were all zero's. A Stanford Course on using Convolutional Neural Networks for Visual Recognition reveals that "Unfortunately, ReLU units can be fragile during training and can "die". For example, a large gradient flowing through a ReLU neuron could cause the weights to update in such a way that the neuron will never activate on any datapoint again. If this happens, then the gradient flowing through the unit will forever be zero from that point on."[4]. Changing the activation function from ReLU to tanh solved this issue. An alternative method would be to use LeakyReLU activators. A second improvement was made by adding batch normalization. Batch Normalization should improve the training speed (Ioffe and Szegedy [5]), however the model also slightly improved in its macro F1 score by applying it. Increasing the amount of 'units' in each layer also proved to increase the performance up to a certain amount. Increasing the amount of Dense layers in the network seems to somewhat stabilize the final result, however it did not improve the resulting score.

Some variations of the model increased the amount of neurons and layers. Increasing the amount of neurons improved the performance. Adding

additional layers additionally improved the performance, but only up to a certain point before it started having diminishing returns.

The final model can be seen in Listing 5

Listing 5: Final Deep Neural Network

```
model = Sequential()

model.add(Dense(256, input_dim=X_train.shape[1], activation='tanh'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(512 ,activation='tanh'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(256 ,activation='tanh'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(128 ,activation='tanh'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(64 ,activation='tanh'))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(y_train_encoded.shape[1], activation='relu'))

model.summary()
```

4. Results

4.1. Model Evaluation and Validation

The final model consists of 6 Dense layers, 5 Dropout layers and 5 Batch-Normalization layers. the activation layers were changed from ReLU to tanh to prevent the 'dead' ReLU problem, where all of the results ended up as 0. The final activation layer was kept as ReLU, to reduce the amount of label misses. Tanh activation allows values to get negative, which cannot be used in the prediction. The result of the model would vary between runs. With a standard deviation of 3% and an almost 20% variation between worst and best run the model cannot be considered stable. It does however have a mean which seems consistent over most of the runs (Figure 7 and Figure 8).

	0
count	100.000000
mean	0.186567
std	0.037239
min	0.095455
25%	0.187729
50%	0.191068
75%	0.191068
max	0.387465

Figure 7: Statistics for the neural network

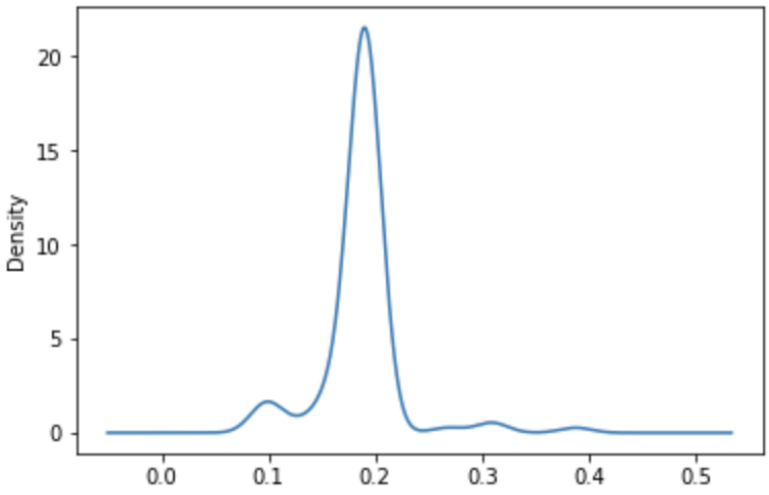

Figure 8: Statistics plot

4.2. Justification

Although improvements looked promising during the development process of the Deep Neural Network, it does not compare to the Random Forest Classifier at this stage. On the testing dataset, the macro F1 score of the

Dataset	**macro F1 Random Forest**	**macro F1 DNN**
Training	0.9299	-
Cross-validation	0.4153	0.2420
Testing	0.3920	0.1250

Table 3: macro F1 score comparison

DNN is over 20% lower then the Random Forest Classifier (Table 3). Although with some added improvements like better pre-processing of the data or trying a different architecture, it does not prove to be a good starting point for this specific problem.

5. Conclusion

5.1. Free-Form Visualization

In Figure 9 results for training the network can be shown. the upper image shows the categorical accuracy of the model, measured by keras' *metrics.categorical_accuracy*. The lower image shows the loss over each run. Although minimal, during training the model does seem to optimize itself slowly. The categorical accuracy of the model however does not really improves during training. One problem with the model is that it does not seem to give stable results. During development of the model, it would sometimes reach a macro F1 score of 0.29 on the validation dataset, but could also go as low as 0.08. Although the best score is stored and loaded to do the eventual prediction, the 'best' model may not always show itself. Looking at the Figure 7 and taking into account that the Random Forest Classifier also had a lower score on the testing dataset, the final score on the testing set is not completely unexpected.

5.2. Reflection

Using a deep neural network for predicting this problem did not prove to be a good solution. Partially this may be because of the data, and quite possibly also the model. I still struggle with is coming up with a design for a neural network. Although there is a lot of information online about neural networks, most of it is used for computer vision. It seems there are no concrete steps in building one, and that advice is mostly given in heuristics like "don't overcomplicate it'. I did learn a couple of interesting things about the design. One thing that surprised me was the 'dead' ReLU problem where

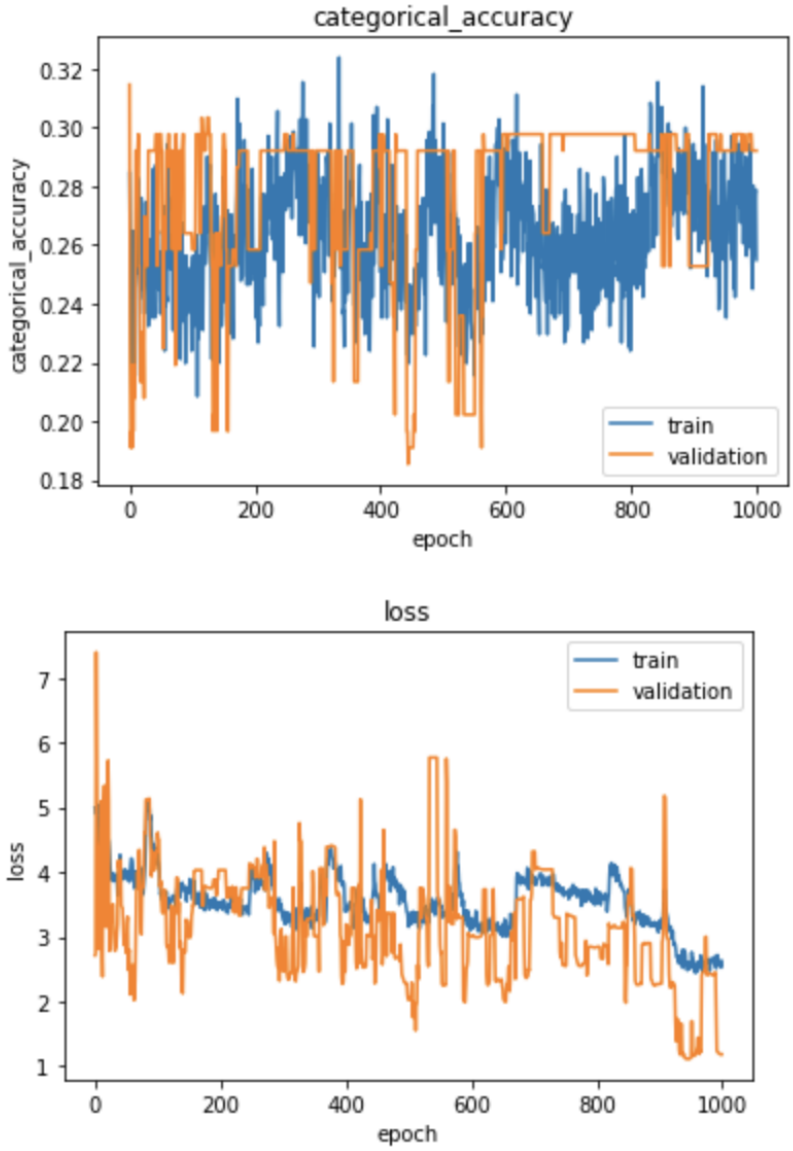

Figure 9: DNN accuracy and loss

predictions were all 0, which did not conform for any situation. Some online material on this was helpful. Another surprise was the effect either over-sampling or undersampling had on the resulting macro F1 score. In general, there is a lot for me to still learn in this area, both in Data Science and

Machine Learning. I am positive I will keep on improving my skills for the future.

5.3. Improvement

The biggest problem I found was that it was hard for the model to separate the data. One improvement would be to spent more time in pre-processing the data, this would take some time however. Additionally, not using a neural network but a more conventional machine learning algorithm might prove to be a better idea. An additional improvement would be to stabilize the model more. While running simulations, the resulting macro F1 score could differ by approximately 0.10 between runs, without the model changing.

[1] L. McBride, A. Nichols, Improved poverty targeting through machine learning: An application to the usaid poverty assessment tools, 2015.

[2] S. Yutaka, The truth of the f-measure, 2007.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, Journal of Machine Learning Research (2014).

[4] Stanford, Cs231n convolutional neural networks for visual recognition, 2018.

[5] S. Ioffe, C. Szegedy, Batch normalization: Accelerating deep network training by reducing internal covariate shift, CoRR abs/1502.03167 (2015).