Project Name: Classifying Turkish Lira Banknotes

Project Members: Rana Begüm Kalkan, Emre Öztürk, Sinan Parmar, Elif Yılmaz

PROJECT REPORT

Abstract: The aim of the project is classification of Turkish Lira banknotes by using machine learning models such as scikit-learn and tensorflow. The solution of this machine learning problem is used and helpful for the people who have eyesight problems. There are 6900 different images of banknotes which are 5, 10, 20, 50, 100 and 200 Turkish Liras. Some photos include unrelated background elements which are included to improve real-life accuracy. The data is tested by using "neural network", "logistic regression", "decision tree", and "support vector machine" models. The models are validated by using cross-validation and then they are compared according to accuracy scores to determine the best model for classification of the banknotes.

Introduction: We want to be able to classify Turkish Lira banknotes using images. This is basically an image classification problem. We are going to test different machine learning models on our dataset to find the best model and methods for this particular problem. We will be using a mix of datasets we found online and provided ourselves. We will be testing our data on "neural network", "logistic regression", "decision tree", and "support vector machine" models. We will validate the models using cross-validation and compare the scores given for each model in order to find the best model.

Methods Review and Background: The dataset includes images of Turkish Lira banknotes with applied image manipulation techniques (salt and pepper noise, random flip and zoom, etc.). Some images are found from online websites but some images are also added to increase variation. Therefore, the images comprised of different sizes and compression formats. The models for the classification are "neural network", "logistic regression", "decision tree", and "support vector machine". It is expected that the "neural network" model is good for image classification.

Data Cleaning and Restructuring: In the dataset, there are no missing values since it includes only banknote images. Therefore, there is no need for data cleaning.

Methods and Tools: We will be using Python 3, Jupyter Notebook and the following libraries:

<u>For ML models</u>

- scikit-learn 0.23.1
- TensorFlow 2

<u>For preparing the data</u>

- Pandas
- NumPy
- Pickle
- Pillow
- OpenCV
- os (built-in Python 3 library)

Our dataset is composed of different datasets[1] which include images of Turkish Lira banknotes with image manipulation techniques(salt and pepper noise, random flip and zoom, etc.) applied: some are available online but we also added some images we shot to increase variation. Our images consisted of different sizes and compression formats(jpg, png, HEIC, etc.). We imported our images using os.walk() method and then converted them to arrays of dimension (32, 32, 3) containing RGB lighting values using fromarray() and resize() methods of the Pillow library; later we transformed these arrays to NumPy arrays for convenience. We have 3 channels because our images are coloured; we also believe that color will play an important role solving this problem. This idea can be seen on a previously made analysis[2] on the banknotes' colour gamuts using one of our datasets. We normalized our data and saved the NumPy arrays for future use. We are saving our arrays so that we don't have to import

[1] Turkish Lira Banknote Dataset, from Fatih Baltacı on Kaggle. TurkishBanknoteDataset, from ozgrshn on GitHub.
[2] "Identifying Characteristics of Turkish Banknotes", from Duncan McKinnon on Kaggle.

lots of data every time we run our kernel. This increases our data size but cuts down a very slow step.

Now that we have our data ready, we can start creating models. For all models, we used train_test_split() from sklearn.metrics in order to divide our data into training and testing sets and created their confusion matrices to see how the TP,FP,TN,FN values are distributed.

Logistic Regression

Logistic regression is a binary classification technique**.** It can be used for multiclass classification problems too. We used LogisticRegression[3] model from sklearn; used OneVsOneClassifier[4] for cross-validating in order to find the best parameters for our problem which needs to be able to identify 6 different classes.

Decision Tree

Decision tree classifiers are made of nodes with specific tests on attributes leading down to more nodes alike until it is assigned to a class**.** We used DecisionTreeClassifier[5] model from sklearn. Used GridSearchCV[6] for cross-validating in order to find the best parameters for our model.

```
parameters = {'max_depth':[2,4,10],
              'min_samples_split' : [2,3,4]}
```

SVM(Support Vector Machine)

Support vector machines create hyperplanes to classify all inputs in a high dimensional space. The closest values to the classification margin are called "support vectors". The model tries to maximize the margin between the support vectors and the hyperplane. We used SVC[7]

[3] LogisticRegression() from sklearn.linear_model
[4] OneVsOneClassifier() from sklearn.multiclass
[5] DecisionTreeClassifier() from sklearn.tree
[6] GridSearchCV from sklearn.model_selection
[7] SVC() from sklearn.svm

model from sklearn. Used GridSearchCV() for cross-validating in order to find the best parameters for our model.

parameters={ 'C' : [0.1,0.5,1,5,10],

'kernel' : ['linear', 'poly', 'rbf','sigmoid',],

'tol' : [0.000,0.001,0.1]}

Neural Network

There are different types of neural networks with infinitely many architectural possibilities. Obviously we can't test our model for every single design. Even though there are models with many hidden layers(YOLO, VGG, Xception, etc.), theories state that one can apply any abstract function with only 2 hidden layers; any addition of a hidden layer adds nonlinear capabilities which may come useful for some wide range classification tasks with a need of more features. It may not have a drastic difference for our problem but still, higher is better.

The CNN model we built is pretty basic. It has a few 2d convolution layers with Relu activation, a few 2d max pooling layers and a batch normalization layer in between: a softmax layer in the end. It takes inputs of size (32, 32, 3) and gives probabilities for 6 classes(our banknotes).

The Xception model comes at a size of nearly 180 layers and 20 million trainable parameters. Because of its size we needed more data features in order to train the model. We feeded image arrays of size (224, 224, 3) using the same preprocessing mentioned before, just with a different size output. Again as output layers it gives the probability distribution for the 6 classes via a softmax layer in the end.

We used the same inputs for the ResNet50 model, again the model outputs probabilities same as the other models.

All the CNN models are created using Tensorflow 2 and Keras(which is also a part of TF2 now). All the other models are created using the scikit-learn package.

The accuracy for each model is calculated using test sets on the trained models in order to choose the best model for the task.

Results:

Table 1: Basic model

Epoch	**Loss**	**Accuracy**	**Validation Loss**	**Validation Accuracy**
Epoch 1/10	0.6797	0.7701	1.0776	0.6000
Epoch 2/10	0.2519	0.9217	0.4073	0.8775
Epoch 3/10	0.1490	0.9567	0.2756	0.9268
Epoch 4/10	0.0820	0.9737	0.3462	0.9130
Epoch 5/10	0.1036	0.9661	0.4961	0.8993
Epoch 6/10	0.1143	0.9674	0.3987	0.9080
Epoch 7/10	0.0688	0.9797	0.5944	0.8920
Epoch 8/10	0.0929	0.9721	0.5915	0.8804
Epoch 9/10	0.0614	0.9817	0.9531	0.8688
Epoch 10/10	0.0628	0.9815	0.4024	0.9283

Table 2: Xception model

Epoch	Loss	Accuracy	Validation Loss	Validation Accuracy
Epoch 1/10	0.8375	0.7007	1.3569	0.5957
Epoch 2/10	0.3220	0.8922	0.1122	0.9594
Epoch 3/10	0.1753	0.9424	0.3740	0.8870
Epoch 4/10	0.1751	0.9455	0.6059	0.8500
Epoch 5/10	0.1187	0.9601	0.4227	0.8572
Epoch 6/10	0.1114	0.9679	0.0443	0.9848
Epoch 7/10	0.0888	0.9716	0.6804	0.8109
Epoch 8/10	0.1007	0.9692	0.0670	0.9790
Epoch 9/10	0.0494	0.9861	0.0985	0.9681
Epoch 10/10	0.1052	0.9672	0.0463	0.9848

Table 3: ResNet50 model

Epoch	Loss	Accuracy	Validation Loss	Validation Accuracy
Epoch 1/10	1.1873	0.6036	3.2935	0.2696
Epoch 2/10	0.4969	0.8379	2.7552	0.2087

Epoch 3/10	0.3432	0.8886	1.6981	0.5674
Epoch 4/10	0.2630	0.9190	8.0680	0.3457
Epoch 5/10	0.2058	0.9351	24.5402	0.1746
Epoch 6/10	0.1903	0.9377	4.0642	0.4725
Epoch 7/10	0.1470	0.9527	3.6743	0.4138
Epoch 8/10	0.1283	0.9589	3.8220	0.5007
Epoch 9/10	0.1248	0.9591	2.5259	0.4442
Epoch 10/10	0.1592	0.9513	0.8624	0.7935

We used a NVIDIA GTX 1650 (mobile) graphics card with 4GB of GDDR5 memory and 896 NVIDIA CUDA Cores. The basic model took around a minute, Xception took 33 minutes, and the ResNET50 model took 20 minutes to train.

- Our basic model has 0.93 test accuracy and 0.40 testing loss (Table 1).
- Xception model has 0.98 test accuracy and 0.05 testing loss (Table 2).
- ResNet50 model has 0.79 test accuracy and 0.86 testing loss (Table 3).

Thus, it can be said that the Xception model is the best performing among our CNN models.

In the decision tree model, GridSearchCV calculated that the best parameters are 10 for maximum depth and 3 for minimum sample among the given parameters mentioned in the methods part. Then, the model is trained with the best parameters and the accuracy is calculated as 62%.

In the SVM model, again using GridSearchCV, we found that the best parameters for our SVM model are 10 for “C”, rbf(radial basis function) as the kernel type and 0.1 for tolerance among the given parameters mentioned in the methods part. The accuracy of the model is 94% when trained with the best parameters.

Table 4: Models and Accuracies	
Model	**Accuracy**
Basic CNN w/ sequential layers	0.9283
Xception CNN	0.9848
ResNet50 CNN	0.7935
Logistic Regression	0.9072
Decision Tree	0.6166
Support Vector Machine	0.9362

According to table 4, Xception is the best model for classifying Turkish lira banknotes looking at both the accuracies and the losses of the models; we should also note that 4 of our models have accuricies above 90+%.

Discussion:

In order to classify Turkish Lira banknotes from their images, we used logistic regression, decision trees, SVM, and Convolutional Neural Networks with three different architectures: A basic CNN with x layers, Xception, ResNet50. Our dataset consisted of already available datasets of Turkish Lira images. The model that performed the best was Xception with an accuracy rate of 98% and a validation loss of nearly 0.05%. This model requires around 500 MB of storage, having 71 layers and over 20 million trainable parameters; this is mainly because each added layer looks for more nonlinear features, thus requiring more data(more pixels in our case). Hence, it might be more suitable for implementations that require high accuracy and have more storage capacity(for computer programs etc.). Other models that worked close to Xception were the SVM and the basic CNN model. Although fitting the data to SVM took more than five hours, saving the model for future uses eliminated this drawback. These models reached 92% and 93% accuracy respectively and took inputs of size (32, 32, 3) compared to (224, 224, 3) input size of Xception which are much smaller in size resulting in a much smaller model. We think that SVM or simply a basic CNN with only a few convolution layers can be used in mobile applications where we can trade high accuracy with small storage requirements(mobile apps, general IoT applications etc).

Resources and References:

- https://towardsdatascience.com/understanding-and-coding-a-resnet-in-keras-446d7ff84d33
- https://keras.io/guides/sequential_model/
- https://maelfabien.github.io/deeplearning/xception/#ii-in-keras
- https://www.kaggle.com/duncankmckinnon/identifying-characteristics-of-turkish-banknotes
- https://www.kaggle.com/baltacifatih/turkish-lira-banknote-dataset
- https://miro.medium.com/max/1000/1*kkyW7BR5FZJq4_oBTx3OPQ.png
- **Neural networks and deep learning**