Turkish Lira Banknote Images Classification

University of Milan

Lirida Papallazi

2019/2020

Abstract

This project's aim is that of implementing an image classifier based on neural networks based on Tensorflow to classify images of banknotes from the Turkish Lira Banknote Dataset available on Kaggle, using as features vectors the rgb and grayscale representation of the images. The trained classifier must take an image in input an provide as output the corresponding predictions with respect to the 5, 10, 20, 50, 100 and 200 lira classes. The subjects of feed-forward neural networks, Convolutional Neural networs, RGB and grayscale space model and PCA are explored.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The main purpose of this project consists of using convolutional neural networks to classify the images of Turkish banknotes given in input to the algorithm. In particular the algorithm must be able to predict with a good accuracy level if the image corresponds to a 5, 10, 20, 50, 100 or 200 banknote, which makes the problem a Multiclass Classification problem. Banknote classification is at the basis of the proper working of Automatic Teller Machines (ATM) and counting machines the possible uses are countless, including: banknote recognition (namely determining the denomination of the input currency paper), counterfeit detection and serial number recognition.[1] Moreover machine learning techniques applied in this field could be useful to develop applications for visually impaired people so that they can actively participate to the economical aspects of their countries and therefore be better integrated in the system.

The analysis will proceed as follows:

- Extraction of the features, both for the rgb scale and the grayscale.
- Analysis of the dataset;
- Implementation of feed-forward neural networks and considerations on the results;
- Implementation of convolutional neural networks and consideration on the results with appropriate techniques to see if there is any improvement such as Dropout and BatchNormalization.

In the project will be used especially Tensorflow, a math library that is also used in machine learning.

Chapter 2

Dataset and Features

As previously stated the dataset is available on Kaggle[2] and it is composed as follows. Each banknote category is divided into folders, each containing 100 images: 925 for training and 75 for validation. Therefore the training and the validation sets comprehend respectively, 5550 images and 450.

Figure 2.1: An image with salt and pepper noise

The author of the dataset specifies that some data augmentation techniques such as increase/decrease of brightness, flip, addition of saltpepper noise have been applied to the images, therefore no further augmentation technique has been implemented. Each image has a quality of 1280x720x3, respectively width, height and 3 RGB channels.

2.1 Grayscale color space

A grayscale picture expresses the intensity of information, and in particular the brightness of each pixel: the higher the value, the brighter the pixel. Therefore for each pixel of an image is needed only one byte (8 bits) that stores a value from 0 to 255. The conversion of colored images to grayscale is not unique as it depends on the different weights assigned to the color channels. One of the most common conversion techniques is the one that uses photometry or colorimetry to compute the grayscale values so that they give the same relative luminance as the original color image, and it is the so called Colorimetric (perceptual luminance-preserving) conversion to grayscale strategy. The aim is that of obtaining the relative luminance Y_{linear} we

just need to compute the following linear equation obtained by the linearization of the gamma compression function at the basis of the RGB color model:

$$Y_{linear} = 0.21R_{linear} + 0.71G_{linear} + 0.07B_{linear} \tag{2.1}$$

where the three coefficients represent the intensity of luminance perceived by humans. Since people are more sensitive to green, hence the greatest value, while blue has been assigned the lowest coefficient.[3] However, we will use the

`tf.image.rgb_to_grayscale`

function (belonging to the tensorflow package), which converts more efficiently one or more images from RGB to Grayscale.

2.2 RGB color model

When the color of an image is added then it becomes necessary to also consider the concept of shade and 3 bytes for each pixel (namely 24 bits) have to be used to express this information. In order to assign numbers to the colors different color spaces have been thought and the simplest one is the RGB color model or space.[4] A color model is an abstract mathematical model describing the way colors can be represented as tuples of numbers, typically as three (as in our case) or four values or color components. The RGB model is an additive color model since the perceived color can be predicted by summing the numeric representations of the component colors which in this model are Red, Green and Blue. The color of a pixel is therefore represented by an rgb triplet whose components vary from 0 to 255, since each channel is represented by a byte.

Chapter 3

Mathematical Background

In order to provide even the less experienced reader with basic knowledge on the topics and techniques adopted in the project, a short mathematical discussion will be provided in this section. However, since the topics touched are many and far more complex, the reader is invited to refer to further sources of information more focused on the singular subjects for a deeper understanding.

3.1 Introduction on Machine Learning

Machine learning is a subset of artificial intelligence that produces computer algorithms aimed at making predictions and in general solve data inference problems such as categorization, clustering and predictions.

In basic terms these algorithms learn functions that map data points x to labels y; in this project the data points or observations would be the banknotes, while the labels are the possible values of a banknote: 5, 10, 20, 50, 100 or 200. Since the labels are symbolic this is a classification or categorization problem, thus entering in the supervised learning category. As a matter of fact it is possible to distinguish between supervised learning or learning by examples and unsupervised learning. In the former the algorithm learns on a labeled dataset (classification and regression are two examples) while the latter provides unlabeled data from which the algorithm tries to extract features and patterns on its own (clustering is the typical representative of such algorithms). The function learned by classification algorithms is called classifier.

3.1.1 Loss function

A non-negative loss function l is used to measure the "goodness" of the prediction $\hat{y}$ of y. For classification problems the most typical loss function is the zero-one loss since the the mistake is given by predicting the wrong label:

$$l(y, \hat{y}) = \begin{cases} 0 & \text{if } y = \hat{y} \\ 1 & \text{otherwise} \end{cases} \tag{3.1}$$

The loss function is minimized via Gradient Descent, one of the most popular iterative algorithms to perform optimization and find the minimum value for a function.

3.1.2 Training set and Test set

In order to learn the function mapping the observations to the label set, the algorithm uses a set of already labeled examples, the training set composed of the pairs of inputs and the corresponding labels $(x_1,y_1),...,(x_m,y_m$. The test set $(x'_1,y'_1),...,(x'_n,y'_n)$ is used to estimate the accuracy of the predictions of the classifier (or predictor), by computing the test error with respect to the real labels.

$$\frac{1}{n}\sum_{t=1}^{n} l(y'_t, f(x'_t)) \tag{3.2}$$

Training and test set are typically obtained via random split of the original dataset.

3.1.3 Overfitting and underfitting

Since the only input given to a learning algorithm is the training set the first idea would be that of computing the training error

$$\hat{l}(f) = \frac{1}{m}\sum_{t=1}^{m} l(y_t, f(x_t)) \tag{3.3}$$

The Empirical Risk Minimizer (ERM) is the algorithm that outputs the predictor f in the set F that minimizes the training error.

$$\hat{f} = \underset{f\in F}{\operatorname{argmin}}\, \hat{l}(f) \tag{3.4}$$

This technique however might conduct to issues such as that of overfitting (low training error, but high test error, typically caused by "noisy" data) or underfitting (high training error and high test error). Furthermore, it is necessary the assumption that the examples (x, y) are randomly generated from a joint probability distribution D thus implying that both the training and the test sets are random samples.

In particular, overfitting happens when a model learns the detail and the noise of the training set at a level for which this negatively impacts the performance of the model when classifying the new data, namely the test set. By converse, underfitting happens when the model can neither fit the training data nor generalize the new data.[6]

3.2 Linear prediction and Perceptron

[15] The perceptron is an algorithm for binary classifiers.

A binary classifier is a function f: $X \to Y$ that outputs whether or not an observation has a specific label; in this case $Y = \{-1, +1\}$. Binary classifiers are a type of linear classifiers, and in particular linear classification predicts the label of an input based on the value of a linear combination of the characteristic necessary to identify a specific class.

3.2.1 Linear classification

For $X = \mathbb{R}^d$ a linear classifier h are those function where S^+ and S^- are the half-spaces defined by the partition of the hyperplane S in $\mathbb{R}^d$. We recall that an hyperplane can be described with a single linear equation of the following form:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n = b \tag{3.5}$$

where a_i and b are real coefficients. It is therefore possible to rewrite the hyperplane as:

$$S(a, b) = \{x \in \mathbb{R}^d : a^T x = b\} \tag{3.6}$$

It follows that the half-spaces can be written as: $S^+ = \{x : a^T x > b\}$ and $S^- = \{x : a^T x \leq b\}$ and that the linear classifier associate to the hyperplane S will be: $h(x) = sgn\{a^T x - b\} = sgn\{\widetilde{a}^T \widetilde{x}\}$ where the second equality is a consequence of considering homogeneous hyperplanes. The application of the ERM algorithm to find the classifier $\hat{h}$ in H_d that minimizes the training error is a NP-hard problem, namely it is not possible to find an algorithm that solves

$$\hat{h} = \underset{h \in H_d}{\operatorname{argmin}} \frac{1}{m} \sum_{t=1}^{n} \mathbb{I}(h(x_t) \neq y_t)) \tag{3.7}$$

When the training set is linearly separable solving (3.7) becomes easier because it is equivalent to finding the vector $w \in \mathbb{R}^d$ solving the following linear programming problem:

$$y_t w^T x_t > 0 \tag{3.8}$$

for $t = 1, \ldots, m$.

3.2.2 Perceptron

In the case of linearly separable training set it is possible to use the Perceptron algorithm. The Perceptron finds a homogeneous separating hyperplane by running the training examples one by one; in case of misclassification the associate hyperplane of the current linear classifier is adjusted.

Algorithm 1: Perceptron

Input: Training set $(x_1,y_1),\ldots,(x_m,y_m)$
Initialization: of the weights w = (0,...,0) and of the thresholds
Repeat: Read next training example (x_t, y_t);
while $y_t w^T x_t > 0$ **do**
  if $y_t w^T x_t \leq 0$ **then**
    $w \leftarrow w + y_t x_t$
Output: w

When combined with many other perceptrons it forms an artificial neural network.

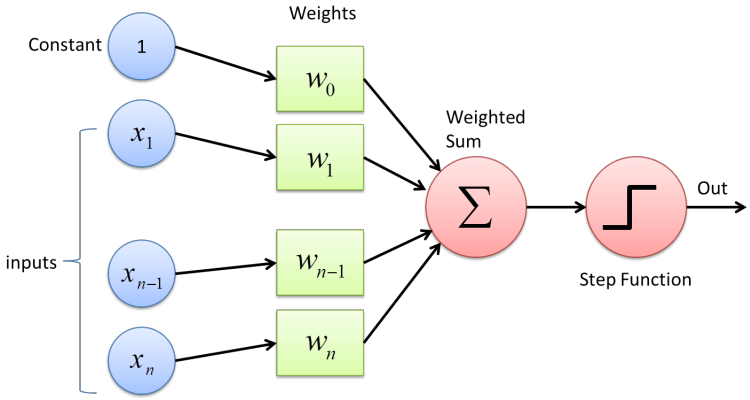

Figure 3.1: Perceptron[16]

3.3 Neural Networks

An artificial Neural Network (NN) is composed by nodes interconnected between them via links: each node represents an artificial neuron and each link has a weight representing the strength of one neuron's influence on another. The learning process is very similar to that of the perceptron: examples are processed by computing the weighted associations between input and relative output. The difference between the prediction and the real value of the observation is then computed and the network adjusts the weights according to a learning rate and progressive adjustments will help the network to make more accurate predictions.

3.3.1 Artificial Neuron

An artificial neuron is a mathematical function: it receives one or more inputs that can be the features of the object in analysis and computes the weighted sum of them in order to produce an output.

3.3.2 Activation function

The weighted sum previously obtained is then passed into a non-linear function known as activation function.

The main examples of activation functions are:

Threshold function

With this function the neuron remains silent up until a certain threshold is reached but it is quite difficult from a mathematical point of view as it is possible to observe from the graph.

Sigmoid function

The next proposal was the sigmoid function where the highest the hyperparamenter β, the steepest the graph around the zero.

$$\sigma(x) = \frac{1}{1 + \exp(-\beta x)} \tag{3.9}$$

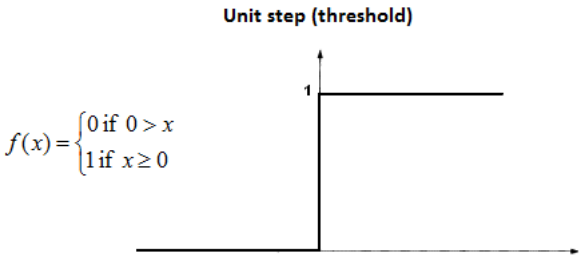

Figure 3.2: Threshold function

ReLu activation function

$$f(x) = max(0, x) \tag{3.10}$$

This function gives gives an output x if x is positive, otherwise it will output 0. Just like the sigmoid function, it is nonlinear.

The leaky ReLu

$$\sigma(x) = (\alpha \mathbb{I}(x < 0) + \mathbb{I}(x \geq 0)) \tag{3.11}$$

Softmax activation function

The softmax function is a generalization of the logistic function and it takes as input a vector m of Z real numbers and normalizes it into a probability distribution that consists of Z probabilities. By assigning it to the output layer of the neural network for categorical target variables, the outputs can be interpreted as posterior probabilities. We briefly remind the reader that in Bayesian statistics the posterior probability is the conditional probability that derives once an important fact of the past has been taken into account.[7]

The choice of the activation function depends on the learning problem and it is possible to choose a different activation function for each layer.

3.3.3 Hyperparameters

A hyperparameter is a parameter shose value is used to control the learning process while the other parameters (such as the weights of the nodes) are derived through the training.

Learning rate

It determines the step size at each iteration that the model takes to adjust the errors at each observation. The highest the learning rate, the shorter the training time but the lower the accuracy.

Hidden layers

Hidden layers are those layers between the input and the output ones and are called "hidden" because they are not visible outside of the neural network.

Batch size

Batch size controls the number of training samples that will be propagated through the network.

Epochs

The Epochs hyperparameter instead controls the number of complete passes through the training dataset, namely it is a measure of the number of times all the traing vectors are used once to update the weights.

3.3.4 Accuracy

Once the loss function that takes as inputs the weights, the biases and the training examples is fixed, it is necessary to define a metric that measures the "goodness" of the model. Ideally the lower the output of the loss function, the better the neuron is at predicting the data.

The main metrics are:

Accuracy

Accuracy is typically defined as the ratio between the number of correct prediction with the number of total predictions.

For binary classification accuracy becomes:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}} \tag{3.12}$$

Where:

- TP = true positives;
- TN = true negatives;
- FP = false positive;
- FN = false negatives.

Kullback-Leibler divergence or Cross-entropy

[18] The Kullback-Leibler divergence originates from the mathematical definition of entropy of a discrete probability distribution and it measures how similar a probability distribution p is to a candidate distribution q.

$$D_{KL}(p|q) = \sum_{i=1} p_i log \frac{p_i}{q_i} \tag{3.13}$$

3.4 Feed-Forward Neural Network

A feed-forward neural network is an artificial neural network where the term "forward" refers to the fact that information moves in only one direction, forward, from the input nodes, through the hidden nodes, ending in the output nodes, without employing cycles or loops.

This type of neural network computes a function $f : \mathbb{R} \rightarrow \mathbb{R}$. Each note i (but the input nodes) solves the function $g(v)$

3.4.1 Single-Layer Perceptron

A single layer perceptron is the simplest kind of feed-forward neural network.

3.4.2 Multilayer Perceptron

MLP is a class of feed-forward artificial neural network and it consists of at least three layers of nodes: an input, a hidden and an output layer.

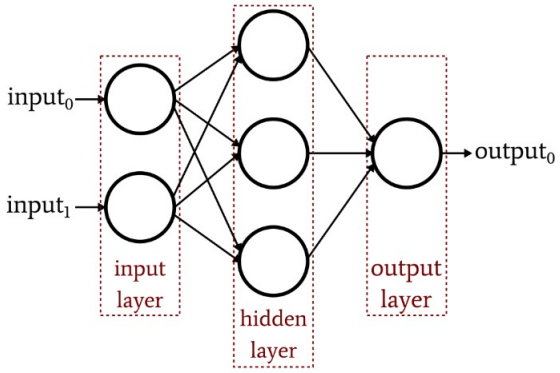

Figure 3.3: Multi-layer Perceptron

In this case the nodes can be partitioned into a sequence of layers such that each node of a layer has incoming edges only from nodes in the previous layer and the outgoing edges go only to nodes in the next layers.

3.5 Dimensionality reduction

In a machine learning problem it is often necessary to choose among the the original features because using a large number of attributes might diminish the predictive capabilities of the algorithm due to overfitting. We can distinguish between two techniques.

3.5.1 Feature selection

This technique allows to select a subset of the original attributes. The dimensionality of the data is reduced by projecting the observations into a subspace of R^d.

3.5.2 Feature extraction

With feature extraction we obtain a new set of data that are linear combinations of the original data and this is performed by projecting the data into the subspace defined by any orthonormal basis $u_1, \ldots, u_k \in \mathbb{R}^d$ with $k < d$.

3.5.3 Singular Value Decomposition

Singular value decomposition or SVD is a factorization of a real or complex matrix based on eigenvalues and eigenvectors. Let X be a real matrix mxn, it is possible to write that: $X = U\Sigma V^T$ where U and V are respectively, an nxn and an mxm unitary matrix, $\Sigma = diag(\sigma_1, \ldots, \sigma_l)$ is an mxn rectangular diagonal matrix with $l = min\{n, m\}$, where the scalars $\{\sigma_1, \ldots, \sigma_l\}$ are the singular values of M. Moreover U and V are orthogonal, thus: $UU^T = I_n$ and $VV^T = I_m$. The following property will follow: $||X|| = ||U\Sigma V^T|| = ||\Sigma|| = \sum_{i=1}^{r} {\sigma_i}^2$.

3.5.4 Principal Components Analysis

Single Value Decomposition is at the basis of Principal Component Analysis.[17]

PCA is one of the most known feature extraction techniques and it is performed via a linear transformation of the variables that projects the original ones onto a new Cartesian system in which the new variable with the greatest variance is projected onto the first axis, the second new by variance dimension onto the the second axis and so on and so forth, and the analysis is limited to the utilization of the variables whose combined variance reaches a previously defined threshold.

Let $X = [x_1, \ldots, x_m] \subseteq \mathbb{R}^d$ be a dxm matrix of instances and $k < d$ an integer. The idea is that of finding the k-dimensional subspace in $\mathbb{R}^d$ that best approximates X. The k-subspace $\Lambda \subseteq \mathbb{R}^d$ is defined by an orthonormal base $u_1, \ldots, u_k$. Let $U = [u_1, \ldots, u_k]$ be a matrix whose columns are the vectors of the base and let $P = UU^T$ be the linear operator that projects a vector from $\mathbb{R}^d$ to Λ.

The transformation $x \rightarrow U^T x$ maps the instance $x = (x_1, \ldots, x_d)$ in the k components of x onto the k elements of the orthonormal base, namely: $U^T x = ({u_1}^T x, \ldots, {u_k}^T x)$. It is possible to notice that P is the identity transform for Λ.

The approximation error is computed as:

$$\sum_{t=1}^{m} ||x_t - Px_t||^2 \tag{3.14}$$

and it measures how much distance there is between each observation x_t and its projection Px_t in the k-dimensional subspace.

The PCA problem will thus solve the following:

$$\min_{P \in P_K} \sum_{t=1}^{m} ||x_t - Px_t||^2 \tag{3.15}$$

The new features are therefore linear combinations of the previous features and via PCA each point $x = (x_1, \ldots, x_d)$ in k features $x' = U_K x$, the best k-dimensional approximation of x.

Chapter 4

Data Analysis

In this section the reader will be provided with an illustration of the steps of the analysis conducted which will start with Feature Extraction and will continue in the next chapter with Convolutional Neural Networks.

4.1 Feature extraction

The first part of the analysis consisted of creating two dataframes, one for the training set and one for the validation set, containing the banknotes located in different folders based on their label. The dataframes contain the path of the images and the corresponding label.

In order to asses the composition of the dataset and to understand whether or not adjusting techniques, such as oversampling or undersampling would be necessary, an analysis of the distribution of the labels followed.

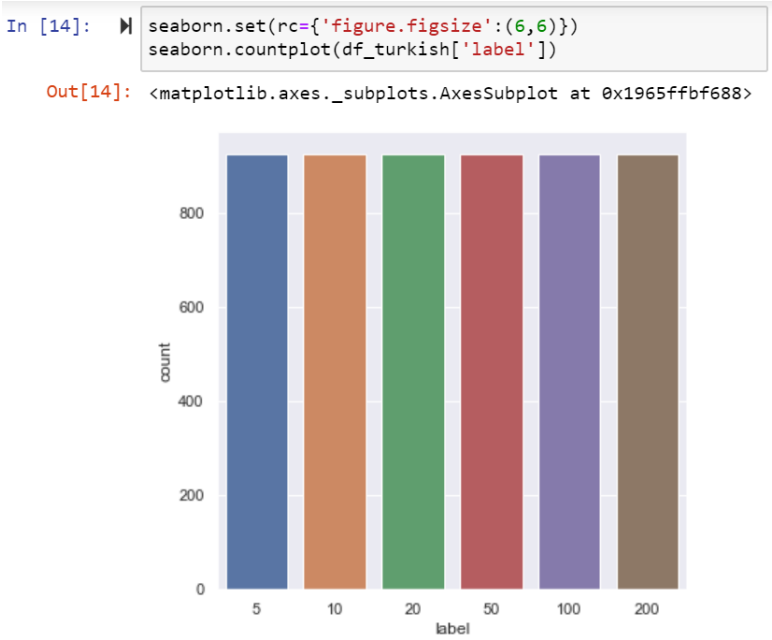

Figure 4.1: Distribution of the training set labels

As we can see from 4.1 on page 18, the dataset is fully balanced therefore no balancing technique are required.

4.2 Preprocessing

As previously stated the original dataset contains 1280x720 shaped colored images and in order to reduce the computation complexity of the problem the images are re-scaled to 64x64 pixels. Maintaining the original format of the images would mean considering a matrix with dimension 1280x720 where each element of the matrix is actually a tuple expressing the coefficient associated to each one of the three colors of the RGB scale model. In the grey-scaled version of the original pictures each element of the 1280x720 matrix would be a value between 0 and 255 expressing the intensity of the color (we recall white being associated to 0 and black to 255).

The resizing of the images seems therefore necessary in order to achieve faster computations. This said another solution might be adopted given the high dimensionality of the dataset, namely Principal Component Analysis.

4.2.1 Linearization

In order to extract the feature vector, namely the linearized version of each image and the cv2 and the PIL packages were used. The training and the validation set have now a shape of respectively (5550, 64, 64, 3) and (450, 64, 64, 3). Before giving the data in input to the neural network we need to "normalize" the pixel values in a range between [0,1] which can be performed by simply dividing each value of the matrix by 255, namely the maximal value a pixel can assume.

4.2.2 One Hot Encoding

The next step involves converting the class labels into a one-hot encoding vector. One hot encoding is the process by which categorical variables are converted into a form that could be provided to Machine Learning algorithms in order to improve the predictions.[5].

4.2.3 Train/Test split

Since the dataset already contains a validation set (which is used to improve overfitting), the next important passage consists of partitioning the training set in order to obtain a test set; in this case we opted for a 80/20 partition.

We perform the same procedures on both the original and therefore colored images and on their gray-scaled versions.

4.3 Principal Component Analysis (PCA)

As it was mentioned before the pixels of an image are its features; the training set after being rescaled into 64 by 64 images and before being split to obtain the test set, contains 68198400 elements. PCA is one of the most common dimensionality reduction techniques, which is used to extract information from high-dimensional spaces by projecting it into a lower-dimensional data space while preserving as much of the original data's variation as possible.[14]

4.3.1 Implementation of PCA

A brief explanation of the passages and the necessary steps to implement PCA, follows.

First step: Standardization

In order to apply PCA the data points need to be standardized with the following formula:

$$z = \frac{x - \mu}{\sigma} \tag{4.1}$$

When dealing with images (such in this case) instead of standardization, normalization of the pixels between 0 and 1 inclusive dividing the values by 255 was sufficient. By using the appropriate function it is possible to know that an explained variation of 90% is attainable with ca. 60 features.

Second step: Covariance Matrix

The next step requires the computation of the variance-covariance matrix, A of two variable X and Y.

Third Step: Computation of the eigenvectors and eigenvalues

The computation of the eigenvectors of the variance-covariance matrix A follows. We recall that an eigenvector of a linear transformation is a nonzero vector that chances at most by a scalar factor when that linear transformation is applied to it. And this scaling factor is the eigenvalue, λ. The associated equation is the following:

$$Av = \lambda v \tag{4.2}$$

Fourth step: Choice of k eigenvectors with the largest eigenvalues

The last step requires the sorting of the eigenvectors in decreasing order of the associated eigenvalues, and depending on the number of dimensions that we wish to have in the new data space or on the level of explained variance that is desired, k of them will be chosen.

```
In [41]:  x_pca=PCA().fit(x_train_flat)
          plt.plot(np.cumsum(x_pca.explained_variance_ratio_))
          plt.xlabel('number of components')
          plt.ylabel('cumulative explained variance');
```

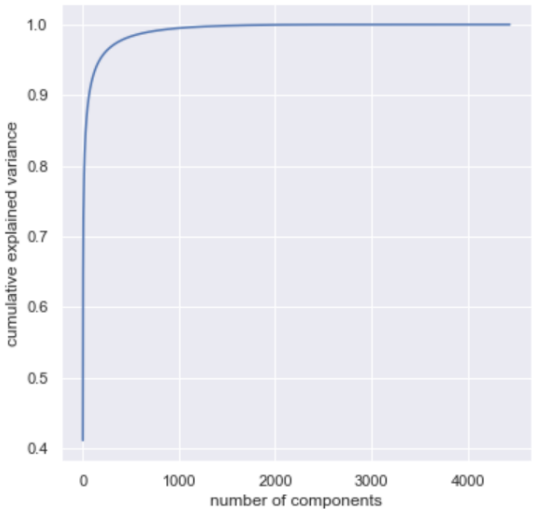

Figure 4.2: Explained Variance for PCA

4.4 Feed-forward Neural Network

The first neural network architecture implemented in this project is the Feed-forward neural network. The purpose of a feed-forward network is to approximate some function $f(x)$, for example a classifier that maps an input x to a category y. In our case the x would be our banknotes, and the y would be the value of the banknote. The feed-forward network defines a mapping

$$y = f(x, \theta) \tag{4.3}$$

and learn the value of the parameters theta that result in the best function approximation.

In the project three feed-forward neural networks were tested on the gray-scaled images, respectively a one, two and three layers neural network. It seems appropriate to address and further explain a few of the arguments of the algorithm.

As previously mentioned in the Mathematical background chapter, an artificial neuron computes a weighted sum of its inputs, adds a bias and then decides whether or not this should be "activated" for the next layers, and this is exactly what an activation function does. The easiest idea is that of using a step function, namely setting a threshold; another alternative would be using the sigmoid function whose nonlinearity helps resolving a few of the issued caused by the usage of linear functions. In the project were used mainly two activation functions: ReLu and softmax.

Another important thing to note is that before giving the images in input to the neural network, they are flattened. Flattening is required in order to combine the information expressed in matrix form, namely contained in both rows and columns.

4.5 Convolutional Neural Networks

Convolutional neural network is a class of deep neural network, commonly applied image analysis. They are inspired by biological organization of animal visual cortex. The idea behind is that of training the machine to detect classify images by detecting some patterns and this is done using filters. The network employs a mathematical operation called convolution, a mathematical operation of two functions that produces a third function, expressing how the shape of one is modified by the other. Citing Wikipedia[9], it is defined as the integral of the product of the functions after one is reversed and shifted. With this computation we obtain information on particular features of the images in input which are significant for the prediction.

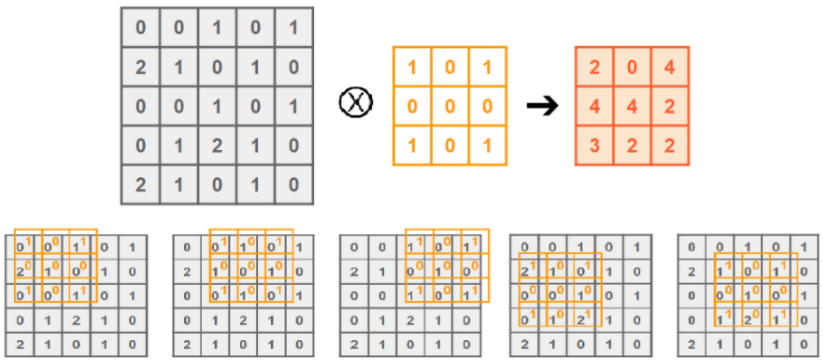

Figure 4.3: 2-dimensional case of convolution

4.5.1 Padding and stride

As we can see from 4.3 on page 22 the pixels at the corner of the image are used much less than those in the middle, therefore some techniques such as Padding and Stride.

Padding

Padding involves adding layers of zeros to our input images as to avoid the previously mentioned issues.

Stride

On the other hand stride controls how the filter convolves around the input volume and it is normally set so that the output volume is and integer and not a fraction.

4.5.2 Pooling and Flattening

Pooling and MaxPooling

In addition to the "classic" layers and in order to improve the performance of the model it is possible to add a pooling layer. The aim of pooling is that of reducing the size of the data; when only the maximal value of a convolution is taken then we are talking about maxPooling.

Flattening

Flattening is required instead to convert the data into a one dimensional array input for the next layer and it is a necessary step for classification problems.[11]

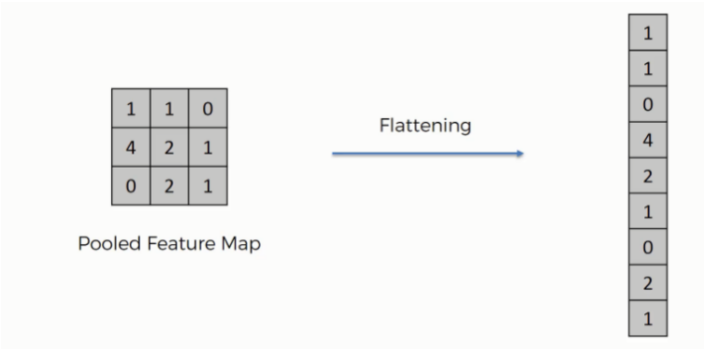

Figure 4.4: Flattening of a feature map

4.5.3 Summing up

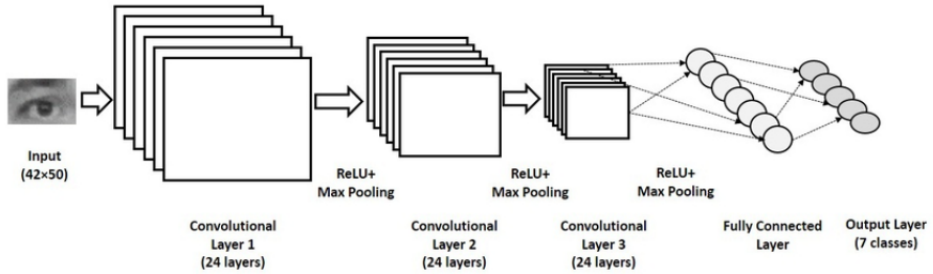

Figure 4.5: Complete Convolutional network with 3 layers

The addition of layers that go deeper into the model, allows it to deal with more and more complex features.

Chapter 5

Results

5.1 Feed-forward Neural Networks

Feed-forward neural networks do not perform well on the gray-scaled dataset, as a matter of fact the best performance doesn't even reaches an accuracy of 45%, and it is not improved by increasing the number of layers nor by adding ad adaptive learning rate. [10]

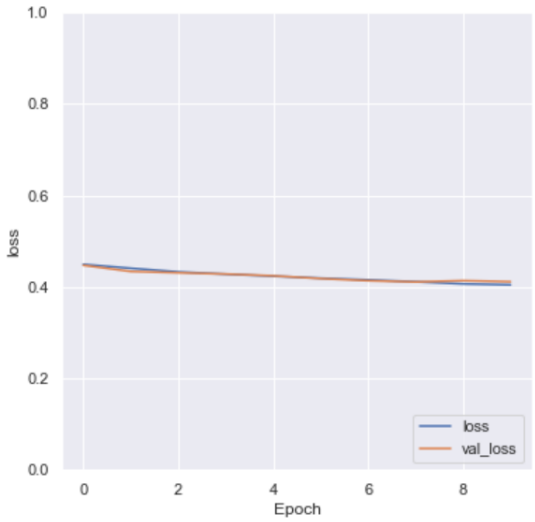

Figure 5.1: Training and validation loss of the Two layer feed-forward NN

Flattening the images implies a loss in shape information which are extremely important for classification and might explain the low performances of the feed forward neural networks. This is why convolutional neural networks might be a good alternative.

Where "LR" implies the addition of the adaptive Learning Rate

5.2 Convolutional Neural Networks

When CNNs are implemented the improvement in performances is visible and an accuracy of 100% is reached.

Table 5.1: Feed-forward Neural Network accuracy and loss

Neural Network	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
One Layer	33.20%	39.06%	41.49%	41.40%
Two Layers	36.94%	32.81%	40.47%	41.15%
Two Layers + Dropout	43.78%	48.44%	37.30%	36.24%
Two Layers + Dropout + LR	31.94%	40.62%	41.90%	41.51%
Three Layers	32.84%	34.38%	41.58%	39.90%
Three Layers + Dropout + LR	27.30%	29.69%	42.37%	41.98%

Table 5.2: CNN accuracy and loss for One and Two block models

CNN	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
One Block	99.86%	100%	3.8%	4.7%
Two Blocks	99.01%	98.44%	1.01%	1.81%

5.3 CNN with Dropout and Batch Normalization

In order to prevent overfitting and therefore to improve even more our results, Dropout first and then Batch Normalization combined to Dropout were applied to the VGG with two blocks.

5.3.1 Dropout

Dropout refers to ignoring randomly chosen neurons during the training phase. In particular, at each stage individual nodes are either dropped out of the net with probability $1 - p$ or kept with probability p.[12]

5.3.2 Batch normalization

The idea behind Batch normalization is that of trying to apply normalization not only to the features of the input, but also to the hidden layers in order to fasten up the learning speed. Another advantage is that it allows each layer of a network to learn more independently of the others.[13]

Batch normalization therefore normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation (we note that on our analysis we used 10 epochs and a batch size of dimension 10).

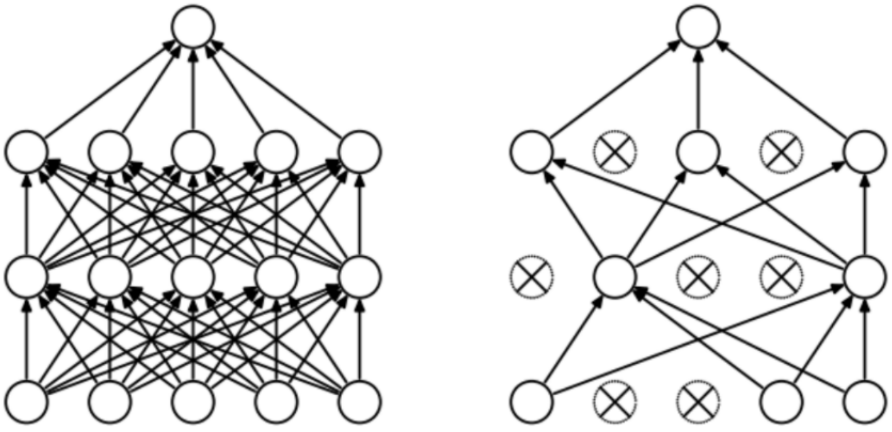

Figure 5.2: Dropout

Table 5.3: CNN accuracy and loss for Two-block VGG with Dropout

CNN	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
Two Blocks	96.76%	100%	3.21%	7.9%
Two Blocks + Batch Norm	100%	100%	0.043%	0.52%
Two Blocks + Batch + LR	99.80%	100%	0.36%	0.16%

It is important to note that the addition of an adaptive learning rate doesn't seem to bring any improvement in the performances and it actually makes them slightly worse.

5.4 CNN applied to Gray-scaled images and Feed-forward NN on PCA transformed images

The next step consisted in applying the last CNN model, namely the one implementing both Dropout and Batch normalization to the gray-scaled images dataset.

And lastly a Two layers Feed-forward neural with Dropout network was used to model the PCA transformed features.

Table 5.4: Accuracy and loss for CNN on gray-scaled images and Feed-forward NN on PCA

CNN	Training Accuracy	Validation Accuracy	Training Loss	Validation Loss
Gray-scale	92.86%	89.33%	6.47%	9.52%
PCA	81.22%	76.56%	15.68%	18.62%

The performance are for sure lower than those of CNN run on the RGB scale pictures but adding more layers to the model could improve the results.

5.5 Computational time

It seems interesting to take into consideration the computational time of some meaningful steps of this project.

Resizing the original 1280x720 pictures into 64x64 dimension took more than 5 minutes for the training set and 22 seconds for the validation set, which is of course a consequence of the number of images each set contains, respectively 5550 and 450.

As for what concerns the time needed for the PCA projection of the original features with respect to the principal ones (defined by setting the explained variance at 90%), the operation took less than 2 minutes. This finding assumes an even greater meaning when compared to the computational time and the accuracy of the feed-forward neural network run on the new extracted features.

Table 5.5: Computational time on dataset preprocessing

	Training set resizing	Validation resizing	PCA feature adaptation
Time	05m:21s	22s	01m:07s

It could be interesting to run a convolutional neural network on the PCA extracted features in order to observe if there might be any change in computational time and in the accuracy. Considering that the latter is twice the one of the feed-forward networks run on the original data (although resized), there might be some positive aspects.

Table 5.6: Computational time NN

	Two Blocks + Dropout and + Batch	FF on PCA
Time	1h:16m:16s	05s

Chapter 6

Conclusion

This work aims at comparing the performances of feed-forward neural networks and the more complex convolutional neural networks on the Turkish Lira Banknote dataset. Feed-forward neural networks provide a good start in analysing and modeling the dataset but as it is possible to observe adding more layers is not always the optimal solution to improve the performances. CNNs places the computer scientist a step forward by providing already better performance that can be further improved, since they are able to observe and capture patterns that are lost when considering grayscaled and flattened images.

Overfitting remains an issue which has to be kept under control via regularization techniques such as Dropout and Batch Normalization.

Another important aspect is that concerning the possible usefulness of using PCA to reduce the dimensionality of the dataset given in input to the CNN in order to increase the speed of analysis without heavily affecting the performance of the model. However in this project the pca subject is not explored in an extensive and deeper way.

It is important to take into account also the fact that the Turkish Lira Banknote dataset is quite small with respect to other existing datasets: the classical example is the CIFAR-10 dataset which consists of 60000 32x32 colour images. Moreover in order to speed up the results were used 10 epochs (in certain cases 20) and 10 the batch-size; it is possible to observe an improvement in performances when both these coefficients are even slightly increased. However, as it is possible to observe in the case of the 3-layer feed-forward neural network, even increasing the number of epochs and/or the batch-size does not provide any significant improvement.

Bibliography

[1] Tuyen D. Pham and Dat T. Nguyen and Chanhum Park and Kang R. Park (2019), Deep Learning-Based Multinational Banknote Type and Fitness Classification with the Combined Images by Visible-Light Reflection and Infrared-Light Transmission Image Sensors, https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5539635/

[2] Kaggle Dataset, //www.kaggle.com/baltacifatih/turkish-lira-banknote-dataset.

[3] Grayscale images, https://en.wikipedia.org/wiki/Grayscale

[4] Utkarsh Sinha, Color spaces, https://aishack.in/tutorials/color-spaces-1/: :text=Grayscale%20imagestext=Current%20displays%20support%20256%20distinct,lighter%20than%20the%20previous%20one!text=So%20for%20a%20grayscale%20image,all%20possible%20shades%20of%20gray.

[5] @rakshithvasudev (2017), What is One Hot Encoding? Why And When do you have to use it?, https://hackernoon.com/what-is-one-hot-encoding-why-and-when-do-you-have-to-use-it-e3c6186d008f

[6] Jason Brownlee (2016), Overfitting and Underfitting With Machine Learning Algorithms, https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/

[7] Avinash Sharma V. (2017), Understanding Activation Functions in Neural Networks, //medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0

[8] Jiwon Jeong (2019), The most intuitive and easiest guide for Convolutional Neural Network, https://towardsdatascience.com/the-most-intuitive-and-easiest-guide-for-convolutional-neural-network-3607be47480

[9] Wikipedia, https://en.wikipedia.org/wiki/Convolution

[10] Jason Brownlee, How to use Learning Curves to Diagnose Machine Learning Model Performance, https://machinelearningmastery.com/learning-curves-for-diagnosing-machine-learning-model-performance/

[11] SuperDataScience Team (2018), https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-step-3-flattening

[12] Amar Budhiraja (2016), Dropout in (Deep) Machine learning, https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[13] F.D (2017), Batch normalization in Neural Networks, https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c

[14] Wikipedia, https://en.wikipedia.org/wiki/Principal$_c$$_o$$_m$$_p$$_o$$_n$$_e$$_n$$t_a$$_n$$_a$$_l$$_y$$_s$$_i$$_s$

[15] Nicolò Cesa-Bianchi, Linear Prediction (2020), http://cesa-bianchi.di.unimi.it/MSA/Notes/linear.pdf

[16] Sagar Sharma, What the Hell is Perceptron (2017), https://towardsdatascience.com/what-the-hell-is-perceptron-626217814f53

[17] Nicolò Cesa-Bianchi, Riduzione di dimensionalità (2017)

[18] tdhopper.com, Cross Entropy and KL Divergence, https://tdhopper.com/blog/cross-entropy-and-kl-divergence

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.