Classification of Tibia Plateau Fractures with Artificial Intelligence

Final Adv. Artificial Intelligence Project

Franziska-Marie Ahrend

Supervised by Dr. Long Ma

Department of Computer Science

CS 6678-Advanced Artificial Intelligence

Dr. Long Ma / Troy University

Troy, April 2022

Importance of a new approach of classifying tibia plateau fractures

The tibia plateau is the head of the larger bone (tibia) of the lower leg and therefore, directly below the bone femur, as shown in image 1. Hence, tibia plateau fractures are breaks at the top of the tibia bone (see image 1). Those fractures are usually complex and involve also other parts of the knee as for example the joint surface, surrounding nerves, the meniscal and vascular injuries. Treatment needs to be planned carefully as it has major impact on the patients' walking abilities and future quality of life. In the literature, there are 38 different classification systems that describe this specific type of fracture (Schröter and Schreiner, 2020). However, one recently introduced by Krause et al. (2016) is increasingly used for images specifically from computed tomography (CT) and will be used in this project. This concept is called "ten-segment-classification" and divides the tibia plateau into multiple segments (Krause et al., 2016), which is shown in image 2. There are a total of 10 segments: ALL, ALC, AC, AMC, AMM, PLL, PLC, PC, PMC and PMM which meanings are described in table 1.

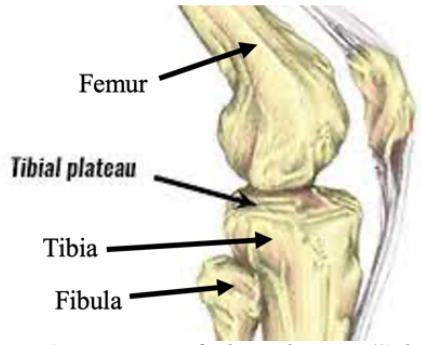


Image 1 Location of tibia plateau (Schröter and Schreiner, 2020), modified by Ahrend, F.

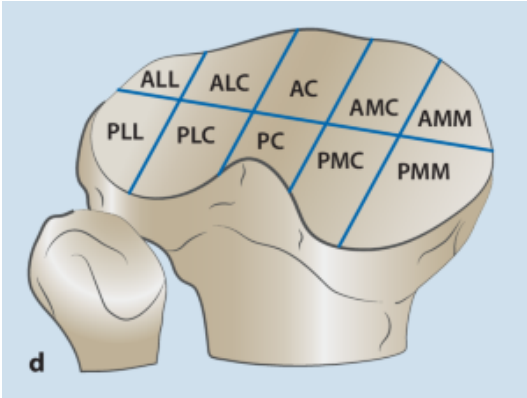


Image 2 Ten-segment-classification for tibia plateau fractures at a right knee (Schröter and Schreiner, 2020), meaning of abbreviations: table 1

Table 1 Abbreviations and their meanings of the ten-segment-classification (Korthaus et al., 2020)

Abbreviation	Meaning
AMM	antero-medio-medial
AMC	antero-medio-central
PMM	postero-medial-medial
PMC	postero-medial-central
AC	antero-central
PC	postero-central
ALL	antero-latero-lateral
ALC	antero-latero-central
PLL	postero-latero-lateral
PLC	postero-latero-central

Usually and especially those fractures which are in need of a surgery, do not show a break in one of the ten segments, but a minimum of two, and sometimes even an occurrence in all of them. For example, image 3 shows fractures in following segments: ALL, ALC, PLL, PLC, PC, PMC and PMM.

Data collection and description

Image 3 Axial view of tibia plateau fracture

I was fortune and am thankful to have received different types of computed tomography (CT) images and associated analyzed data that were condensed by Dr. Marc-Daniel Ahrend, MD.

Even though I got various attributes in the data, for this final project I had to limit those for the data mining process. From the axial, frontal and 3D images of totally 398 fractures, I chose the axial view (see example in image 3) for further analysis due to possibly most valuable conclusions for hospitals and patients' planning of treatments. Apart from the images which related to unique IDs, an excel-sheet provided information about the specific fractures. Here, information such as age, gender, days of present fracture until first surgery, knee side and the type of fracture were listed associated with the ID and four different classification models. For this final project, I chose the previously introduced "ten-segment-classification" by Krause et. al (2016).

Topic selection

Even though the first idea of simply detecting if a fracture in x-rays is present or not, this plan changed due to the kind of data that I received: The original idea of x-ray images that do not necessarily show fractures, turned to CT images, that are usually only taken when a fracture is already detected, and the decision of operational procedures is already made. With those kinds of images, a differentiation of fracture/no-fracture is not feasible or meaningful. Therefore, the analysis of the fractures of the earlier discussed classification for planning surgeries was tried for this final class project. Hence, the goal is to classify the axial images with the "ten-segment-classification" by Krause et. al (2016) with implementing different data mining techniques, which are closely discussed in the next section.

Used data mining techniques

The term "data mining" summarizes techniques for detecting patterns in raw data, Therefore, it has the function to produce interpretable results. This relatively broad term includes, among others, data cleaning and preparation, data warehousing, classifications, and machine learning, which will be all used or at least in detail discussed during this project. This project is my very first project working with images instead of data in form of strings and integers, therefore I am relatively new to this field. Hence, I used as a starting point for the actual implementation of a multi-label image classification, the steps from a documentation by Pulkit Sharma on Analytics Vidhya (Sharma, 2019). However, the further I proceed in this project, the more I modified and experienced with parameters to improve the model.

Data storing

I have spent quite some time considering how to best store the received data as good data storage is essential for fast and efficient data processing. Originally, I started to implement connecting my JupyterNotebook with MySQLWorkbench (see notes-section of code-submission) with the goal of later (this was before I received the data) loading the images and other data into this relational database management system. However, with learning about different database structures to store and process data in my "Advance Database Concepts"-class, I was trying to find the best storing system for this particular project, which should not only fit with the current amount of available data, but also after possibly implementing the visions for this project in the future (more images, other types of images, e.g., frontal; more detailed description in the discussion of this paper). This gets increasingly important with bigger sizes and a greater amount of data. Data warehouses are specific databases that are designed specifically for centralizing the already cleaned data for an improved analysis. With some resulting difficulties and with the background of the number of available data, I decided to store the images in a folder with the corresponding ID and ".png" as a name so that the access through JupyterNotebook is simplified and faster which made the focus on other data mining techniques possible and to better results with the limited time.

However, especially with a bigger dataset my current grown knowledge in database concept would suggest using a NoSQL database, in specific a document database since it handles large volumes and sizes of data faster and is also more flexible and scalable. Also with the received data, the relational component of SQL is not necessary. MongoDB could be one example for such document databases and is great in its compatibility with Python and has its advantage in being an open-source and free platform (Walters, 2017).

Data cleaning and pre-processing

While working on this project a lot of data cleaning and pre-processing needed to be done. This step can be separated in input (x) and output (y), in which the input represents the images, and the output represents the classifications. All steps were implemented with Python 3 in the web-based interactive development environment (IDE) JupyterNotebook.

As it was my first time processing **images** with Python, my first step was iterating through the folder. Due to the lack of data when I started the project, I executed my code at first with just two images for testing purposes. The iterating through the folder was implemented with the os-module which allows interactions with the operating system. I was able to convert the images into an array and display those, both, in a matrix and as an image (see image 4a). Furthermore, I reviewed the dimensions (shape) of the images and were able to conclude that the images do

not all have the same height and width and need to be adjusted accordingly. Also, the text in the corners needs to be removed to receive the ROI (region of interest).

As shown in image 4b, a rectangle helped me to find suiting coordinates for cropping the images' corners. By experimenting with different numbers and reviewing the provided coordinate system on the sides of the graphics, finding those locations with remaining as much tissue around the bones the CT-images was successful, and the images were cropped (image 4c).

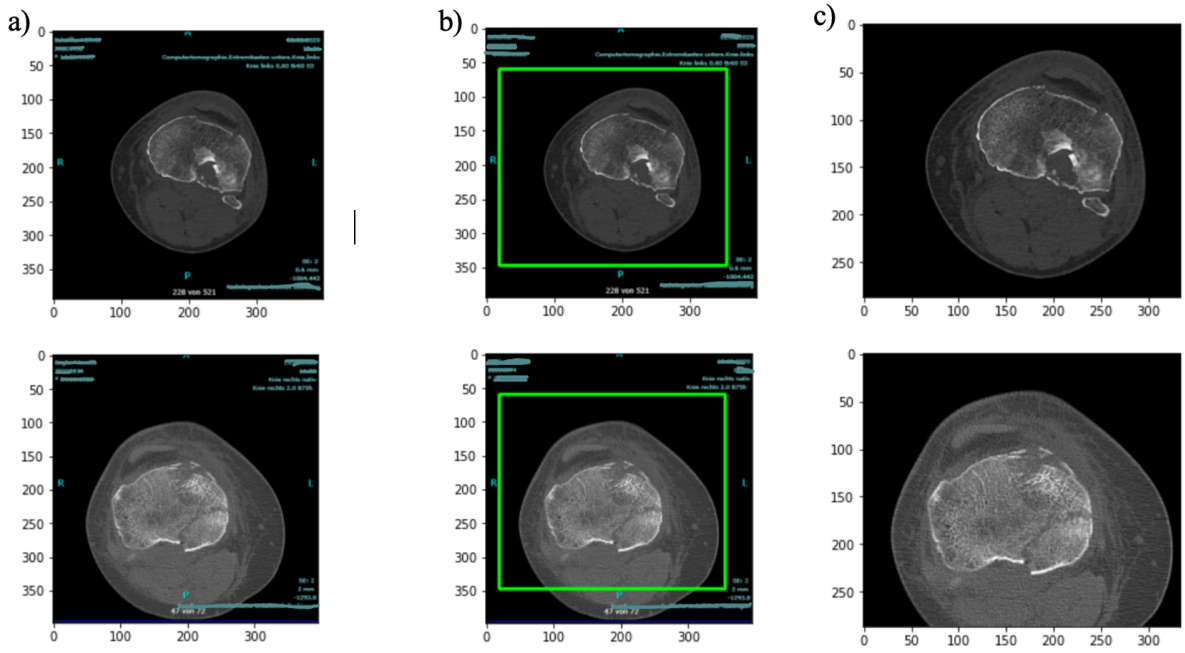


Image 4 Cropping of two CT-images to remove text in corners; a) original image with covered text, b) rectangle in green to determine cropping location, c) cropped image

As to this point not all images were present yet, this cropping was first implemented as a test, however, later implemented for all roughly 400 images within the pre-processing step.

The next step was to work with the **classifications** which were given in a .csv-file. This file was loaded into JupyterNotebook and represented further in a data frame. As it initially contained of many empty rows and columns, the first step was to remove those. As the images and the classifications are indirectly connected by identification numbers (id), they needed to be equally. For this purpose, the dots (and in rare cases commas) needed to be removed in the data frame. After those steps and generally cleaned data, I saved the table into a new csv-file for having the option of using the complete data for future research purposes.

Finally, I removed all columns but the ones with the id and the specific ten-segment classification and created ten new columns which represents each segment separately. If a segment is listed in the complete classification, the row shows in the specific column the

number one, otherwise the number zero. This new table with the complete output-data was again saved in a separate csv-file.

After those steps, all images are in one folder and are for the following steps able to be connected with the classifications from the table and are therefore prepared for a first test session with just 35 images. This step was performed when I did not have all data yet. The steps performed in this part follow the same pattern like with the whole data set, so I will skip the detailed explanation here.

Before working with all images and corresponding classifications, a control of equal shapes was performed: Matrix with images is of shape 397 items, 287 height, 334 width with each 3 colors and matrix with classifications is of shape 397 items and 10 classifications.

After this checkpoint, the data was splitted into a training and validation data set, which is the standard procedure for machine learning procedures.

Multi-label image classification & Model's architecture

I decided to use multi-label image classification which is not to be mistaken with multi-class image classification. Here, the main difference is that we speak about multi-class image classification when just one object exists in the picture which needs to be classified to one of many categories. However, multi-label image classification shows more than one object or classification and therefore need to be assigned to multiple categories. Regarding the calculated probabilities for classifications, multi-class image classifications have probabilities that are dependent from each other, as when one probability rises the probability of other classes decreases. In the case of multi-label image classification are those outputs independent from each other, as more than one result can be correct. When considering the tibia plateau fractures with the classification from Krause et al. (2016), there is also not just one of the ten segments the solutions but a combination of those. For this purpose, I started using the sigmoid activation function as a n-binary classification model which can calculate the probability of each label individually and independently by creating as many models and corresponding probabilities as there are classes. To improve the accuracy of this model, the goal is to reduce the binary_crossentropy loss as much as possible (Sharma, 2019).

Those parameters were implemented within the TensorFlow 2 library which has the purpose of developing and training Machine Learning models (Abadi et al., 2016). In specific the Keras deep learning API was used which is a model inspired by biological neural networks of humans' or other animals' brains (Chollet et al., 2015).

There are different parameters which can be included and modified within the model's architecture, so for example the number of hidden layers, neurons for each layer or the type of

activation function. In table 2 (appendix) a selected list and explanations of different elements is provided. I used different parameters for compiling the model for which there are two parameters necessary: optimizer and loss function. Also, when fitting or training the model I experienced widely with different architecture structures. To give an example: The batch size for training the model is the number of used training examples per epoch. In which one single epoch is one forward and one backward pass of the number of training examples determined by the batch_size. Whereas smaller numbers here have the advantage of running much faster and require less memory space. In contrast higher numbers tend to be more accurate in training performance (Karimanzira et al., 2020). This pattern can be seen in image 5: Karimanzira et al. (2020) analyzed the learning rate outcome of their specific dataset, with different sizes of batch (batch = full batch gradient, mini-batch = smaller batch, stochastic = batch size of 1).

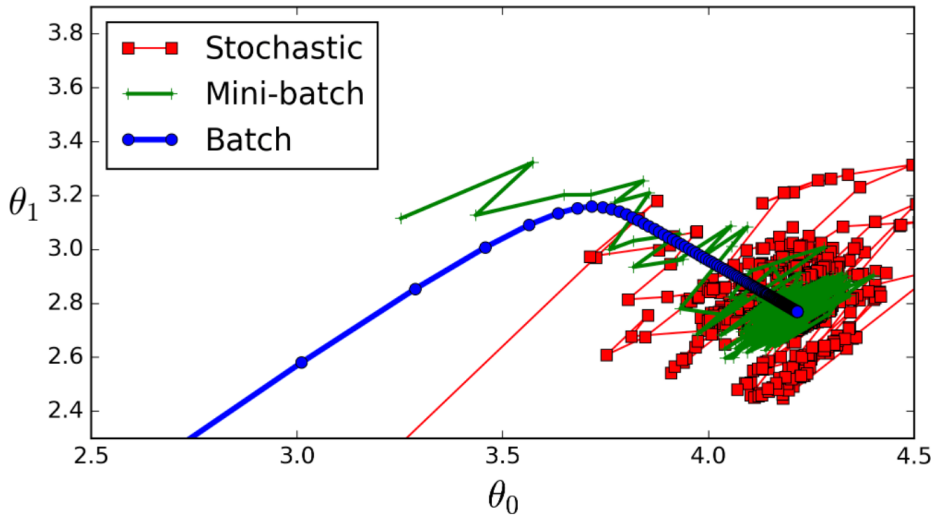


Image 5 Batch sizes in comparison (Karimanzira et al., 2020)

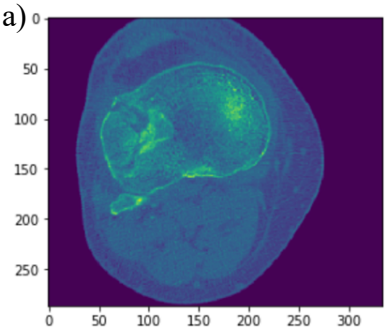
I used that information to better understand the general influence of the batch size on the model and tested different numeric values for this purpose. I also was able to see differences in performances depending on the batch size and decided to choose a batch size of 64.

Results

When performing the training of the epochs, the training and testing performance's accuracy and validated accuracy are computed. Those were not as high as hoped, as they were at about 10% for both types of accuracies with some overfitting towards the training data which is shown by the graph in the code-submission which compares the loss and the validated loss in comparison for each epoch. However, when looking how the results are measured, those low numbers can be explained: We receive per segment of the classification a percentage and all of those 10 segments' likelihoods are adding up to 1 (or 100%). Hence, it does not indicate the specific classification prediction but instead rather gives an indication in which region the

fracture probably lays. An example is shown in image 6, in which one can compare the predictions made by the machine learning model and the actual classification made by a doctor: We can see that in this specific example the segments with higher percentages are indeed higher in latero-regions which can be confirmed by the actual result.

a)



b)

Model's prediction:

ALL	ALC	AC	AMC	AMM
0.162904	0.112307	0.086309	0.05782	0.04672
PLL	**PLC**	**PC**	**PMC**	**PMM**
0.183875	0.144364	0.08731	0.065291	0.0531

c)

Actual classification:

```
[1, 1, 1, 0, 0,
 1, 0, 0, 0, 0]
```

Image 6 One computed prediction from validation set; a) image of fracture, b) model's prediction of classification, c) actual classification performed by medical doctor

Improving the results

After receiving the first results of this project, my goal was to improve this model in its performance. Part of those thoughts were related to the model's architecture, but I also implemented different modifications on the data itself: Due to the images' grayscale-colors, I figured it is unnecessary to use three colors (RGB) so that I converted the images from three to one layer per image. Furthermore, and more importantly for the performance of the model, I added information from the originally provided table, in specific the note about in which knee the fracture appeared. As left and right knees are mirrored, the classifications are also mirrored. Hence, my solution for this issue was to flip images of the left knee (in data frame noted with 'li' which stands for 'links'/left), so that for example the classification ALL is always on the upper left corner of the knee and not just on the right knee and otherwise on the upper right corner. I figured that this additional consideration with the given data could improve the model. With this modification of the input data, I was able to increase the training's accuracy by more than 5%; however, the validation accuracy and loss resulting from the results of the testing (or validation data set) was not improved.

F1 Score

After my supervisor's suggestion of including a so-called f1 score in my model, I directly tried to implement it. The F1 score is a parameter for statistical analysis for specifically binary classifications and performs a measurement of accuracy of the test and includes true positives as well as false positives and negatives in its formula. It shows the harmonic mean of both, precision, and recall. The result of this score after applying it, was unfortunately zero (not

exactly zero though due to problem with dividing by zero) for both the training and validation dataset.

Discussion

During this project, I realized different points in which it could be improved with the goal of increasing the performance. There are different topics that might have made it difficult for the machine learning algorithm to properly identify pattern within those images. In many cases the whole leg was not present in the cropped image, which might influence the range of learning from swelling in soft tissue. Furthermore, some images had a blue line crossing the CT image which is originally used for doctors to make notes or mark specific areas. However, I believe the two biggest problems with the images were the quantity of data as well as the location of the cut. On the one hand, the quantity of the image that were in total at roughly 400 images might just not be enough. There exist different techniques to reduce the negative impact on a lack of data, which however will be discussed later together with the other main problem of the data. On the other hand, if the axial image cut was performed at a location superior (higher) or inferior (lower) than the actual location of the fracture(s), then a correct classification might even be impossible. In many cases, personally, I was not able to confirm the classification made by the doctor just by analyzing the axial images and had to take the anterior images to hand. As those images are also available to me, taking both those images for analyzation might already improve the learning rate of the model significantly. If the goal of achieving classifications just by the axial images remains, then acquiring multiple axial cuts from those knees could enhance the classification performance. Those could be additionally a solution for the low quantity of the data as it results to more images with axial view. Combined with this also synthetic data could be generated by techniques like the Synthetic Minority Over-sampling Technique (SMOTE) to generate modifications of the real data (Gonfalonieri).

Even though, I have read and experimented already a lot on the model's architecture, I believe it could still be improved in some ways due to the high quantity of possible combinations of different parameters. Also, I would like to optimize the integration of the F1 score in the results' outputs so that I receive more information regarding the precision and recall of the model.

As I am planning to work a couple of more weeks on this project, I will first take some simpler classification model to hand, such as the one by Luo et al. (2010) which is shown in image 7. Due

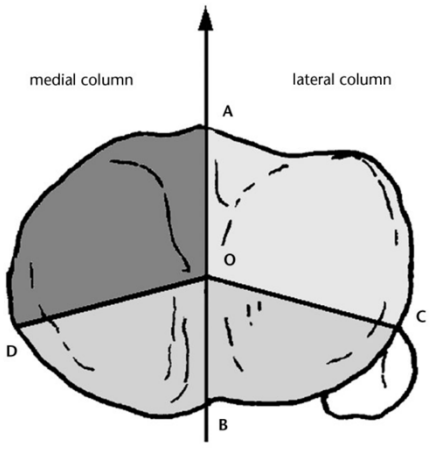


Image 7 Three-column fixation (Luo et al., 2010)

to just three classes instead of the ten of Krause et al. (2016), I hope to enhance the accuracy of the model by reducing issues of overfitting and regain new information about possible enhancement options with the available data.

It would be great to eventually automate classifications of tibia plateau fractures with machine learning models and connect those results with advice how to perform the surgery. Combined with this goal, it could also relax the doctor's daily work to execute an automated medical summary and surgery report. For this, not only the image classification model needs to be enhanced in its accuracy, but also software needs to be written to create a user-friendly interface for entering new CT images. It would be ideal to get feedback from the doctor if the classification was successful or where mistakes could be, so that the classification model can be further trained. This however can only be implemented if (I) the accuracy is optimized, (II) patient data security is ensured, and (III) interactions between doctor and software is guaranteed to prevent wrong interpretations and evaluations (Ghassemi et al., 2020).

References:

Abadi, M. et al. (2016). Tensorflow: A system for large-scale machine learning. In 12th Symposium on Operating Systems Design and Implementation. 16, 265–283.

Chollet, F. et al. (2015). Keras. https://github.com/fchollet/keras.

Ghassemi, M., Naumann, T., Schulam, P., Beam, A.L., Chen, I.Y., and Ranganath, R. (2020). A Review of Challenges and Opportunities in Machine Learning for Health. AMIA Jt Summits Transl Sci Proc. *2020*, 191–200. .

Gonfalonieri, A. 5 Ways to Deal with the Lack of Data in Machine Learning.

Karimanzira, D., Renkewitz, H., Shea, D., and Albiez, J. (2020). Object Detection in Sonar Images. Electronics *7*, 1180. https://doi.org/https://doi.org/10.3390/electronics9071180.

Korthaus, A., Ballhause, T.M., Kolb, J.-P., Krause, M., Frosch, K.-H., and Hartel, M.J. (2020). Extended approach to the lateral tibial plateau with central meniscal subluxation in fracture repair: feasibility and first clinical and radiographic results. Eur J Trauma Emerg Surg *46*, 1221–1226. https://doi.org/10.1007/s00068-020-01467-1.

Krause, M., Preiss, A., Müller, G., Madert, J., Fehske, K., Neumann, M.V., Domnick, C., Raschke, M., Südkamp, N., and Frosch, K.-H. (2016). Intra-articular tibial plateau fracture characteristics according to the "Ten segment classification." Injury *47*, 2551–2557. https://doi.org/doi: 10.1016/j.injury.2016.09.014.

Luo, C.F., Sun, H., Zhang, B., and Zeng, B.F. (2010). Three-column fixation for complex tibial plateau fractures. Journal of Orthopaedic Trauma *24*, 683–692. https://doi.org/https://doi.org/10.1097/BOT.0b013e3181d436f3.

Schröter, S., and Schreiner, A.J. (2020). Klassifikationen der Tibiaplateaufraktur. *2*, 67–75. https://doi.org/10.1007/s43205-020-00037-0.

Sharma, P. (2019). Build your First Multi-Label Image Classification Model in Python.

Walters, R. (2017). Getting Started with Python and MongoDB.

Appendix

Table 2 List of selected elements of model's architecture and parameters for the implemented machine learning model

Keras	deep learning API, implementing biological neuronal networks	
TensorFlow2	Library for developing and training machine learning models	
train_test_split	Separates full dataset (with input data x and output data y) to training and validation set: X_train, X_test, y_train, y_test Parameters are, among others: - x and y as input and output, respectively - random_state=42 → random separation - test_size: size of the validation/test dataset	
Sequential()-Model	Model = Sequential, one layer to the next, not drawback	
Conv2D	Small window of kernel_size moving over the images: each one of them new Neuronal Network → Categorizations	
	filter	Number of windows
	kernel_size	Size of window (x-/y-cordinates: pixels)
	activation	Parameter for after performing the convolution to determine which neurons to activate and which not to
MaxPooling2D	Layer that brings together (pools) to Neural Network both in	
	pool_size	Window size with height and width (tuple)
Dropout	Against overfitting, downsamples the input	
Flatten	To 1D array	
Dense	Reduces number of nodes in layer	
Optimizers	Changes attributes to reduce losses, attributes can be weights or learning rates, required argument for compiling the model, instantiation either by initializing it beforehand or by its string identifier within the model.compile()	
	adam	Stochastic gradient descent method: adaptive estimation Advantage among others in little memory requirement
	SGD	Gradient descent optimizer, iterative method for optimization with smoothness properties
loss	Function for compiling that regulates the quantity to minimize the data while training	
	Binary crossentropy	probabilistic losses between true labels and predicted labels, for binary (0/1) classification
	categorical crossentropy	probabilistic losses between labels and predictions, for two or more label classes
epochs	one forward and one backward pass of the number of training examples determined by the batch_size	
Batch size	number of used training examples per epoch	
Val_accuracy	Accuracy from validation/testing dataset	
Accuracy	Accuracy from training dataset	
Loss	Value of loss-function resulting from the training dataset	
Val_loss	Value of loss-function resulting from the validation dataset	