Deep Learning Neural Network: Image Classification Program

Tycho De Saeytyd
Department of Electrical and Computer Engineering, Marquette University
Milwaukee, United States
tycho.desaeytyd@marquette.edu

***Abstract*—As machines are becoming smarter every day and the impact of AI in our daily lives is increasing, it is important to have a good understanding of how the most used machine learning systems work. Computer systems are able to process and classify given images, e.g. Facebook suggests people to tag in your photos based on your friend list, and those techniques are starting to get used in various industries as well. Computer vision is a part of machine learning where you can train the machine to recognize people or objects in an image. In this project, a supervised image classification program has been built and is able to recognize 6 different persons on fresh images that the machine hasn't seen before. The accuracy of the system is 73.8% and has a loss 1.9711 on the validation images.**

Keywords— AI, machine learning, computer vision

I. INTRODUCTION

In the last decade, there have been major advances in the field of computer vision. Large computer systems have been designed to recognize objects and persons on live or still images. For example, at sporting games, there are systems in place to monitor everything that is happening at the game and to quickly identify possible threads and outbursts. The biggest social media platform Facebook Inc. uses a smart algorithm to suggest people to tag in your uploaded pictures, based on the people you have connected with earlier. Mobile applications like Google Lens can identify pieces of clothing or shoes on a given image and then gives you an overview of similar products. Also, in the medical industry, intelligent Computer-Aided Diagnosis (CAD) systems are being used to annotate or grade medical images in a fast and accurate way [1].

For this project, the goal was to create a program that is able to recognize on of my six best friends on an image that is given to the program. Image classification works in such a way that it classifies images to their corresponding classes, which have been generated by the model. I used the supervised image classification technique, where you give a dataset of images to the program to train the model. This dataset has been split up in training images and validation images, so that the machine can test its accuracy after learning. All the images have been labeled, based on the directory the images are in. The network learns from the training dataset and tries to reproduce the results on the training and validation dataset.

I created a Convolutional Neural Network (CNN) for this image classification problem, which is the most popular neural network used for this kind of problems. The high-level neural network Keras API has been implemented to build the different layers of the model. Keras is a Deep Learning library in Python based on Tensorflow.

II. RELATED WORK

Computer vision is a topic that has been discussed in many different papers across many different sectors. The reason is because there is an infinite amount of applications of e.g. image classification that can be designed and used in medical, scientific, economic and other sectors. Computer systems are believed to be more accurate and consistent, compared to humans, and that is why they are being used to a great extent in those different sectors.

a) Medical image classification applications

The digital transformation of hospitals has entailed a large online database of images of different symptoms. These images, together with the recent advances in computer vision gives the medical sector an opportunity for image-based diagnosis, teaching and biomedical research [2]. As an example, these applications can be used to detect a brain tumor in CT scan brain images. Such a program can help doctors and researchers in classifying images into normal, benign and malignant for effective medical diagnosis [3]. Of course, with the use of artificial intelligence in the medical sector, questions about legal liability and social implications arise [4].

b) Automotive industry: self-driving cars

In the automotive industry, image classification has been used for over a decade. Autonomous vehicles have to be aware of their surroundings at all times and need to be able to detect possible threads or harms to the vehicle and its passengers. Many different propositions have been made about object recognition systems and try to improve the existing techniques using deep learning architectures [5]. Research about the prediction of a vehicle's steering angle when it detects an object on its way has been approached with deep neural networks, including object classification, too [6].

These applications of image classification in the medical and automotive industry are just the tip of the iceberg. As mentioned before, there has been done a lot of research regarding using artificial intelligence, and more specifically deep neural networks, in various industries and this will only increase in the next decades. Technology plays an important role in our lives and the idea of using computer systems to classify images in order to detect patterns in a more accurate way than humans is appealing to many industries.

III. Dataset and features

The dataset used for this project consists out of 906 different pictures which were taken on various smartphones. The images of my American friends, Juan and Matthew, were taken by myself on the iPhone XS MAX. To make sure that all of the images were slightly different from each other, I made them stand in different parts of our apartment. The most important thing was that their face was clearly visible on every picture, so overexposure and blurred photos were deleted. The datasets of my 4 Belgian friends were taken by those persons themselves, using the front camera of their smartphone. I asked every one of them to take at least 180 pictures, so I could delete some of the photos that were not usable for this project.

One of my friends in Belgium sent me the pictures via Facebook Messenger and I noticed that all of those images were resized to a lower resolution. Apparently, a conversion happens to all of the photos sent over the social network to speed up the process of transferring large images. This problem has later been solved by sending the dataset via WeTransfer, which is an online file-sharing website that does not affect your files.

The entire dataset has been placed in a main directory called "Friends". In this directory, I created 3 subdirectories named "Train", "Validation" and "Test". The train and validation directories each have 6 subdirectories, named after the different labels for the classes (Catoo, Juan, Matthew, Nicolas, Oliver and Oskar). The train directory has 115 images per label, which results in a total of 690 training images. The validation directory has 35 pictures per label, so 210 validation images in total. Each image title, except for the ones in the test folder, consists of the name of the person, followed by a number. This number is unique and starts at 1 all the way up to 150 (115 training images and 35 validation images per person). The test folder has 6 images in it, named "test1" up to "test6".

A convolutional neural network typically needs a lot of training data in order to do accurate predictions. I extended the amount of training data by using the ImageDataGenerator, provided by Tensorflow. This generator generates a larger dataset by producing minibatches of tensor image data with real-time data augmentation [7]. I set the rescaling parameter to 1/255, so the generator will create minibatches by multiplying the original dataset by this parameter. This pixel rescaling is done for both the training data and the validation data, as I created a generator for both. All of the original images also got resized in terms of image height and width, which was both set to 150 pixels. By doing so, the runtime of this program is significantly lower, compared to the 4K resolutions before the scaling.

IV. Methods

To become a program that is able to successfully classify 6 of my friends, I obviously needed a great amount of functions and methods. The methods are divided into different subsections related to the different parts in the Python code.

a) Importing the dataset into Google Colaboratory

As I was working with Google Collaboratory, a free online Jupyter notebook environment with enough disk and RAM space to execute this program, I had to implement a method to efficiently import my dataset to this cloud program. I found a way to mount my Google Drive to Google Collab, using the "drive" library provided by Google Collaboratory. With a simple method called "drive.mount(PATH_TO_FILE)" I was able to access all of the images uploaded to my personal Drive space. After this, I created different directories in Google Collab in the same way my Friends folder was set up, using the method "os.path.join(PATH, FOLDER)". To see how many images are present in the train and validation folders, I used the "len(LIST)" method, which counts the number of elements in a given list. The given lists were of course the different subdirectories in the 2 folders. After summing up all the elements in the subdirectories, I finally printed the total number of training images and validation images, using the regular "print()" function in Python.

To make the division between training data and validation data more visible to any users of this program, I inserted a histogram that visualizes the number of images for each person. The pyplot functions of the matplotlib are great for these kinds of plots, because they are easy to use and don't require extensive programming. I added a function called 'autolabel' to indicate the exact number of images above each histogram shown. The function "plt.subplots()" allows you to put multiple plots next to each other, which gives a cleaner look to it. Lastly, the "fig.tight_layout()" method is used to make sure that the subplots and titles fit into the provided figure area, so that all of the data is shown correctly.

b) Building the data generators

To increase the amount of data that the model can use to train and validate images, I used TensorFlow's class called "ImageDataGenerator()". This class gives you many arguments which you can change to augment the dataset, e.g. rotating the images, shift the images horizontally or vertically, change the brightness, etc.… The ImageDataGenerator has several methods to import your original dataset into the generator. For this project, I chose the "flow_from_directory()" method, because I divided the dataset in training and validation images. I set 5 arguments for this method, namely batch_size = batch_size, directory = train_dir (respectively validation_dir for the validation dataset), shuffle = True, target_size = (IMG_HEIGHT, IMG WIDTH) and class_mode = categorical. The batch size variable was earlier set to 32 and the IMG_HEIGHT and IMG_WIDTH were both set to 150 pixels. The class mode 'categorical' is the default mode if you are writing a multi-label image classification program. It determines the type of label arrays that are returned. In our case, the labels are 2D one-hot encoded labels.

c) Creating the model

Now it is time the design the actual model that will learn from the training image dataset. There are two ways to create this model: you can initiate the model using "model =

sequential()" and add layers with the method "model.add(layer)" or you can do this in one long string "model = Sequential([layer,layer,…])". I chose the second option for the model in this project. First, I added a 2D convolutional layer with 16 output filters in the convolution. This is the dimensionality of the output space. Layers placed earlier in the convolutional neural network learn fewer convolutional filters compared to layers deeper into the network. That is why you should increase the amounts of filters if you add more layers. The 2D convolutional layer creates a convolution kernel that will scan the three-dimensional input and applies a filter. I chose a 5x5 kernel size, which means that the model will compute the dot products between the pixels of the image and the weights defined in the filter. The activation function is set to relu, which stands for rectified linear unit. I chose this activation function, as it delivered the best results, compared to sigmoid, tanh or softmax. Relu is also less computationally expensive as it requires easier mathematical operations. Lastly, I also gave the input shape of the data as an argument, which is 150px,150px,3. This means that every picture is of the format 150x150 RGB (colorized image), which is necessary for a Conv2-layer.

The next layer is a max pooling layer, which involves a sample-based discretization process. This process will reduce the dimensionality of the images to try to avoid over-fitting by using an abstracted copy of the image. Maximum pooling will calculate the maximum value for each patch of the feature map, which has been created by the convolutional layer. This feature map summarizes the presence of features that the Conv2D layer found in an image. I used a pool size of 2 by 2 what leads to a dimension that is halved. I added 2 more convolutional layers with 32 and 64 output filters and 2 2x2 max pooling layers before I unstacked all of those layers using the flatten layer. This layer is necessary before using the dense layers, because it makes one long 1D tensor out of the multidimensional tensors. It simply multiplies all of the layers so that it can be used as an input layer for the dense layers.

The last three layers are dense layers, which create the fully connected neural network. They do so by using a linear operation in which every input and output are connected by a specific weight. The arguments that are passed in the first 2 layers are 1024 and 512 respectively, together with the relu activation function. This means that the first layer will have an output space dimension of 1024 units and the second layer a dimension 512 units. The last layer has an output space dimension of 6 units, which is equal to the number of unique classes. The activation function is set to softmax, because this is recommended for multi-class, single-label classification [8].

To compile the model we just made, I used the Keras method "compile()". As an argument, you have to define the optimizer that you would like to use. The "Adam" optimizer is an algorithm that is according to the authors "A straightforward to implement, computationally efficient optimization algorithm that can be used instead of the classical gradient descent procedure to update network weights iterative based in training data" [9]. The loss argument is set to categorical cross entropy, which is the optimal choice for multi-labeled image classification. The "summary" method gives you an overview of all the different layers of the model. Now it is time to actually train the model using the dataset we provided. Keras has a method called "fit_generator" which trains the model batch-by-batch (one batch is 32 images) by a Python generator. The data generators for the training and validation images that we made earlier are used as input for this method. The generators run in parallel to the model, for efficiency purposes. This means that the data augmentation happens in parallel to training the model with the new images. We also define the number of epochs, which is the number of times that the entire dataset is passed once through our model. The value of epochs is set to 15.

The last two methods that we run on the model are "evaluate" and "predict". The evaluate method gives two metrics, namely the loss of the model and the accuracy of the model on a given dataset. The predict method is used to test the model on new images and to see if it is able to correctly classify those images.

V. Experiments

For this project to be successful, I had to experiment with several parameters and arguments along the way. The first thing I noticed, after version 1 of the program was completed, was that images in PNG format led to higher accuracy rate compared to the original images in JPEG rate. I found online that this could be because of the fact PNG formatted images don't lose quality each time the image is opened and saved again, in contrast to JPEG images. It provides a better actual representation of the image that has been taken, which may be a reason why it leads to higher accuracy rates.

The most important changes in the final accuracy came definitely from modifications to the model. The number of different layers, the choice of pooling strategy (average or maximum value), the number of output filters, the kernel size, the activation functions… all have an influence on the final accuracy of your program. I tried to find the best possible settings by trail & error and online research. I chose to create 3 convolutional layers and after each one of those layers a max pooling layer. I increased the amount of output filters from 16 to 64, as is recommended if your dataset is rather small. As I already mentioned before, I added the flattening layer after the 3 convolutional and max pooling layers to make one long one-dimensional string out of the multi-dimensional architecture, in order to add the last 3 layers. The dense layers finish off the model by developing a fully connected layer.

The way in which I compiled the model has not really changed throughout the process of writing the program, as all of these parameters are pretty standard for a multi-labeled image classification system. The fitting of the model took a relatively great amount of time to complete. This is due to the image size and the number of epochs and batches. I could speed up this process by decreasing the epochs value, but I was curious to see what the evolution of the accuracy and loss was as the number of epochs increased. Also, I didn't want to resize the image dataset to even smaller pictures, because this leads to a lower accuracy and I wanted the

images to be in an acceptable pixel format when I plotted them. The total runtime of the fitting process eventually took between 20 or 30 minutes, depending on internet connection and computer. For some programmers, this runtime is definitely acceptable.

With the current settings and parameters, I managed to reach an accuracy of 100 percent for my training images and around 80 percent for my validation images. This result is good enough, taking into account that I only have 900 original images in total and that all of the images have been resized to 150x150. The loss of my training dataset reaches almost zero, but unfortunately the loss of my validation images is going up as the number of epochs increases. This can be a result of overfitting and I tried to add dropout layers, to fight the overfitting, but without any effect. Nevertheless, a loss of 2.1 is still acceptable. Normally, you should try to minimize the value of the loss variable to have better predictions, but in the end our predictions were correct, which was the main goal of this project.

To actually test whether the model was able to predict the names of persons on 6 test images correctly, I first imported all of the images in the “test” directory, which was a subdirectory of the main “Friends” directory. Tensorflow has a method called “load_img()” to quickly load new images into your Python program. I appended all of those images to a list called “test_images”, which I used in the next cell. There I first made an empty list called “names”, where all of the predicted names will be stored. The “img_to_array” and “expand_dims” methods are necessary to change the imported test images to the required format of the model we made. Next, I used “model.predict()” to let the model predict the names of the persons on the 6 different images. The model allocates a number to each class, and every class is a name of a specific person. So, for example the class “Catoo” got the number 0, the class “Juan” got the name 1 and so on. As the output of the predictions are integers, I set up an if-else structure to change the numbers back to their corresponding class names. I wrote a for loop, so that this process is iterated for all of the images stored in the test_images list.

To show the predicted names and corresponding images, I wrote a function called “makePlot2”, where every image out of the test_images list is shown together with the names of the names list as titles. I turned off the axis, so that we only see the images and the titles lined up.

The model was able to correctly predict all of the 6 names of the 6 different images in test directory. This result confirms that the program has succeeded in what it has been designed for. Of course, the accuracy and the loss value could get optimized better, but the most important task of the program has been fulfilled.

CONCLUSION

In this paper, the process of creating an image classification program in Python has been discussed. The dataset that was used consists of 906 images of 6 of my friends that were taken by non-professional individuals for the purpose of this project. This dataset was manually split up in training data, validation data and test data. Because this dataset is relatively small to conduct an image classification process, a data augmentation was implemented. The program has been written in Google Collaboratory, because this online jupyter environment provides more RAM and disk space to execute python programs. Keras and Tensorflow were used to create a model that is able to correctly classify new test images to one of the classes, corresponding to names of my 6 friends. The model had a classification accuracy of 80% on the validation image dataset and an accuracy of 100% on the training data and test data.

REFERENCES

[1] J. Stoitsis, I. Valavanis, S. Mougiakakou, S. Golemati, A. Nikita and K. Nikita, "Computer aided diagnosis based on medical image processing and artificial intelligence methods", *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, vol. 569, no. 2, pp. 591-595, 2006. Available: 10.1016/j.nima.2006.08.134.

[2] A. Kumar, J. Kim, D. Lyndon, M. Fulham and D. Feng, "An Ensemble of Fine-Tuned Convolutional Neural Networks for Medical Image Classification", *IEEE Journal of Biomedical and Health Informatics*, vol. 21, no. 1, pp. 31-40, 2017. Available: 10.1109/jbhi.2016.2635663.

[3] K. Suzuki, Feng Li, S. Sone and K. Doi, "Computer-aided diagnostic scheme for distinction between benign and malignant nodules in thoracic low-dose CT by use of massive training artificial neural network", *IEEE Transactions on Medical Imaging*, vol. 24, no. 9, pp. 1138-1150, 2005. Available: 10.1109/tmi.2005.852048.

[4] M. Scherer, "Regulating Artificial Intelligence Systems: Risks, Challenges, Competencies, and Strategies", *SSRN Electronic Journal*, 2015. Available: 10.2139/ssrn.2609777.

[5] S. Sharma, J. Ball, B. Tang, D. Carruth, M. Doude and M. Islam, "Semantic Segmentation with Transfer Learning for Off-Road Autonomous Driving", *Sensors*, vol. 19, no. 11, p. 2577, 2019. Available: 10.3390/s19112577.

[6] A. Maqueda, A. Loquercio, G. Gallego, N. García and D. Scaramuzza, "Event-Based Vision Meets Deep Learning on Steering Prediction for Self-Driving Cars", 2018. [Accessed 2 December 2019].

[7] "tf.keras.preprocessing.image.ImageDataGenerator", *TensorFlow*, 2019. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/preprocessing/image/ImageDataGenerator. [Accessed: 22- Nov- 2019].

[8] "How to choose Last-layer activation and loss function | DLology", *Dlology.com*, 2019. [Online]. Available: https://www.dlology.com/blog/how-to-choose-last-layer-activation-and-loss-function/. [Accessed: 25- Nov- 2019].

[9] J. Brownlee, "Gentle Introduction to the Adam Optimization Algorithm for Deep Learning", *Machine Learning Mastery*, 2019. [Online]. Available: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/. [Accessed: 22- Nov- 2019].