**PROJECT: CIFAR-100 IMAGE CLASSIFICATION**

**USING CNN MODEL**

**BY: TAI NGO**

**DATE: 10/03/2020**

**DSC-680 – BELLEVUE UNIVERSITY**

**Introduction**

In this project, the applications of machine learning and deep learning will be utilized to work on classifying images of common objects and animals. The results of this project can be extracted to help deep learning researchers, or even machine learning enthusiasts get some ideas about tuning their model’s parameters and the optimizer. Building a machine learning model or a simple deep learning network is fairly easy, and most people can do that. However, in order to build and improve the accuracy of a machine learning model to the highest possible value is not an easy task; it requires time, computation power, and it can be very expensive. Therefore, getting results from previous experiments will save researchers and modelers time and effort. The ultimate goal of this project is to showcase the basics of how to build a deep learning network with some basic parameter tunings as well as provide the results, so other people can improve their own models.

Machine learning is a hot topic nowadays as most people would expect it to be the next breakthrough technology, which would improve our daily lives. It totally deserves its recognition as this supposedly new technology has already been incorporated in many ways into our devices that we may not be aware of. All of the simple tasks such as using a Google map to get to a restaurant, searching and reading reviews of an item on Amazon, or making a payment using our cell phone have the presence of machine learning. It is rare to hear a complain about how Google map does not take us to where we intend to go. It cannot have such a high performance without the use of machine learning. Machine learning helps improve the accuracy of the statistical models and algorithms inside Google map. The collected sample data is fed into machine learning models, and they will build and improve the mathematical models for the application to be more accurate. There are billions of static images generated every second, and there is no human being that can analyze, extract important insights and update them all. Machine learning algorithms can perform these tasks and provide real-time, automatic transit data for the users while maintaining a high level of accuracy. Similar to Google, Amazon has long integrated the benefits of machine learning algorithms to vastly improve their business models and customer experience. They had a great idea in mind over 20 years ago to build personalized product recommendations, but their success only happened once they invented the item-based collaborative filtering. This machine learning algorithm singlehandedly made Amazon become the most unique online retailer and proved to be very useful for customers. Machine learning helps protect customers during mobile and online transactions using their credit cards. With built-in machine learning algorithms, the bank can detect frauds and ensure the authentication of the owners even without human intervention. This whole process is secure, fast and automatic. These are just some of the basic examples of how machine learning is making our lives better. It has its own limitations, and to get across that boundary, researchers start to look into deep learning.

Deep learning may sound very similar to machine learning, but it is different. While being a subfield of machine learning, the structure of deep learning is similar to the brain, and it is also called deep neural network. A deep learning network or deep neural network can be described as simple as three parts: inputs, many hidden layers and outputs. Deep learning has many popular applications. Many people consider deep learning to be a revolutionary technology because it pushes the applications of machine learning to a higher level. The development of deep learning was made possible due to the improvement of a computer hardware called graphics processing unit, which allowed deep learning models to work with images more efficiently. Traditionally, the computer is run based on the power of the central processing unit (or CPU). The training of a dataset might take many hours to complete with the power of CPU, now takes only a few minutes with a modern and powerful graphic processing unit (GPU). As the development of GPU has become steady and well-invested, thanks to the gaming industry, a powerful GPU can be obtained by most people to create and run a simple deep learning network with ease. This project will use a class of the deep neural networks, which is the convolutional neural network (CNN or ConvNet). CNN is the main driving force for analyzing visual imagery and will be used to build a model for this project. CNNs are regularized versions of fully connected network or multilayer perceptron; one neuron in one layer is connect to all neurons in the next layer. This characteristic of neurons causes overfitting problems for the network. An advantage that CNN has over other image classification is it requires relatively little pre-processing of data. This advantage allows the network to learn the filters while other algorithms have to hand-engineer.

The main dataset for this project is called CIFAR-100. This dataset was prepared by Toronto University to help other researchers build a deep learning model from scratch for image classification tasks. There is another dataset called CIFAR-10, which is a smaller with only 10 different outputs. CIFAR-100 has 100 outputs or 100 classes such as apples, mushrooms, oranges. Each class consists of 600 images, and it is split by 500 images for training and 100 images for testing. 100 classes are grouped into 20 superclasses such as flowers, insects, people, trees. Each image comes with a fine label (or the class to which it belongs) and a coarse label (the superclass to which it belongs). The goal is to build the CNN model with the training data, evaluate the CNN model’s performance with the test data, and tune the parameters by running different experiments to find the best parameters for the highest accuracy score. Naturally, the accuracy score of the training model will continue to improve with more epochs, but the overfitting problem needs to be considered by examining the validation score. The higher the validation score is the better.

**Summarize the business problem/hypothesis**

Hypothesis: By using the parameters that are run in this project, deep learning researchers and modelers can use these parameters to build more complex models or improve their own models. With a CNN model and 100 expected outputs, the number of Conv layers should be three at the minimum while the number of Max-Pooling layers should be two at the minimum. The hidden units should be at least 256 instead of 128. A common optimizer is ‘rmsprop’, which is slightly less accurate than the optimizer ‘adam’. The CNN layers, which are the base layers, need to be flattened before adding Dense layers on top. This small, simple and basic model is expected to behave similar to larger, more complex models when using these parameters.

The goal of this project could potentially align with some business needs and help solve real business problems. In reality, it can take a tremendous amount of time and expensive computation power to run deep learning experiments to figure out the best possible parameters and achieve the highest accuracy score. By running smaller models and obtaining the knowledge of building these models, researchers can save a lot of time trying to train a large amount of data, which could take days to complete. The results obtained from this project could be served as a stepping stone for businesses to continue forwards with their projects. Since CIFAR-100 is a famous dataset to build an image classification algorithm, it is possible that many people would like to see results from this project before making their own models.

**Questions**

1. What are the important features of this dataset?
2. What insights or purposes can this dataset provide for image classification?
3. Should additional datasets be used to train the convolutional neural network?
4. Should the entire dataset be used for training? How should the feature engineering be performed to select the appropriate features?
5. What are the training variables and target variables?
6. How should the deep neural network be built? How many layers, hidden units? Which functions should be used for the last output layer and inner layers?
7. How can the model accuracy be improved? Should the drop-out, layer regularization, or data augmentation be used to reduce overfitting?
8. How will the model be evaluated? Which metrics to use?
9. Which method should be used for the optimizer and the loss function?
10. What are the variables that need to be plotted to show the performance of the model?

**Method**

In order to build an image classification algorithm, this project will create a convolutional neural network (ConvNet/CNN) model. The convolutional neural network is an important deep learning algorithm in the field of computer vision. The dataset contains thousands of images. Each image is considered an input. When each input is sent to the CNN model, the model will assign a weight and a bias; the goal of this is to differentiate each image from one another. As mentioned previously, a big advantage of CNN model is it does not require too much pre-processing work on the data compared to other algorithms. CNN model is capable of learning the filters and characteristics of the data without hand-engineering. To understand how CNN model handles an image, we need to understand the structural units of an image. An image can be described as a matrix of pixels. A complex image has a unit called pixel dependency, and CNN model is able to obtain this spatial and temporal dependency by using its filters. Moreover, the model can improve the fitting to the images by reducing the number of parameters and reusing the weights. Basically, after each run, the CNN model can improve its understanding of the complication of the images. Typically, an image is a four-dimensional (4D) tensor, which consists of the number of samples, height, width and channels (or colors). An image is either a black and white image or a color (RGB) image, which consists of three scales: red, green and blue. The magic of the CNN model is that it can reduce images into a form that it can process easily while retaining important features. The features that the CNN model keeps are crucial for the classification. In general, it is important for a modeler to keep in mind that the architecture needs to be good at learning the features, and it should be scalable when the datasets become much more massive.

For all images, there are height, weight and channel (whether it is black & white or red, green & blue). When an image is fed into a convolutional layer, it is processed by Kernel (K or Filter). This convolution operation processes the color of the image. This operation can be visualized as two matrices: one for the image with a 5x5x1 matrix and one for the convolved feature with a 3x3x1 matrix. The Kernel will shift 9 times on the 5x5x1 matrix to perform the multiplication operation. The result of the multiplication is sent to one of the cells in the 3x3x1 matrix of the convolved feature until every cell is filled. The point of this convolution operation is to extract important features from the inputs. Different layers with different convolution operation may extract different types of features such as edges, orientations, colors at different places across the inputs. This type of convolution operation will typically lead to two types of results. The result will either cause the convolved feature to reduce its dimensionality, or increase its dimensionality or stay the same. The first result, which is to reduce the dimensionality, can be done by using Valid Padding. The second result, which is to increase or keep the dimensionality the same, is done by using Same Padding.

While adding three convolutional layers, two Pooling layers will also be added in between. It is very common to see Pooling layers being added in between convolutional layers in CNN models. They are there to reduce the spatial size of the convolved feature. This is important because without the Pooling layers, the training will require much higher computational power. Pooling layers help perform dimensionality reduction. In addition, it is also a good technique to extract relevant features; the features can be rotational and positional invariant. Overall, Pooling layers are there to maintain and ensure efficient training. There are two types of Pooling layers: Max Pooling and Average Pooling. As the name suggests, Max Pooling means return the maximum value of certain areas on the image, whereas the Average Pooling returns the average value of certain areas. Max Pooling is used more often, and it is considered to have better performance than Average Pooling. Max Pooling has an interesting add-on that it eliminates the noise while performing dimensionality reduction. A Max Pooling layer typically follows a convolutional layer, and they both form a pair. The higher the details, the more layers of convolutional and max pooling layer. After the modeler decides that enough convolutional and max pooling layer for the inputs, the next step is to flatten the outputs before feeding them to a regular neural network for classification. At this point, the role of convolutional layers and max pooling layers are done for one run.

The next step is to classify the inputs and provide the final outputs. However, this is not the end of the training because the goal is to iterate the run for the model to learn and update the weights for higher accuracy. During the classification process, the inputs will go through a fully connected layer (FCL). FCL is capable of learning non-linear functions or combinations of non-linear functions. At this stage, the network is called a feed-forward network, and the backpropagation is applied. Backpropagation is also known as backward propagation of errors, which is an algorithm using gradient descent. Within this algorithm, there is a loss function, an optimizer and a metrics to monitor the training and testing. The loss function calculates the loss score, which is a method to examine the performance of the model by comparing the expected result with the actual result. The loss score is sent to the optimizer to update the network’s weights. In a perfect model, the most accurate weights would be found and learned by the model to yield the most accurate outputs. The process of calculating the mismatch between the actual and expected values and sending it back to update the weight is the learning process of the network. It constantly learns from the mismatches and reiterate to improve its accuracy. One run from the beginning to the final output is called an epoch. The last layer of the model uses an activation function called ‘softmax’. This activation function will produce a probability distribution over 100 different output classes. For each input, the model will create a 100-dimensional output vector, where output[i] is the probability of the class i. All 100 scores will sum to 1.

This project is using Keras and TensorFlow to build the CNN model. The following attributes may be different for different backend engines such as PyTorch, Caffe, or CNTK, and it will look different if used in different programming languages or platforms. Since this is a multi-class classification problem because for every input there is one output with 100 different classes for the model to choose from, the ‘sparse\_categorical\_crossentropy’ is chosen as the loss function, and the metrics to evaluate the accuracy is ‘accuracy’. The loss function and the metrics are fairly straightforward for this multi-class classification problem, but this project will run at least two experiments to determine the most optimal optimizer. There are two popular optimizers for this case: ‘adam’ and ‘rmsprop’. Each optimizer will be run in 20 epochs to determine which one has higher accuracy score on the test data and by how much. The objectives of this experiment are to figure out how many convolutional layers and max pooling layers are optimal, what is the better optimizer between adam and rmsprop, and how many hidden units should a convolutional layers have. These questions will be answered as the experiments are run, and the results are summarized. There is a series of experiments designed to determine the best parameters.

In the first experiment, the objective is to compare the performance of two convolutional layers and one max pooling layer with the performance of three convolutional layers and two max pooling layers. The result of the first experiment will tell us whether more convolutional and max pooling layers will be better for classification. The activation function for the convolutional layers for all experiments will be ‘relu’. The input shape of the data will be 32 x 32 x 3. After finishing the base layer, which are the convolutional layers and the max pooling layers, the output will be flattened by using the Flatten() function. After that, the outputs are sent to the fully connected layer, Dense layer with ‘relu’ activation function, which has equal number of hidden units as the previous convolutional layers. The final Dense layer will have 100 outputs, and it will be using ‘softmax’ as the activation function. Each experiment will have 10 epochs.

In the second experiment, the objective is to determine whether the convolutional layers should have 128 or 256 hidden units. Typically, more hidden units will result in a more accurate result, but it will cost more time and require more computational power. This experiment will help us understand whether it is worth it to increase the hidden units. The setting for this experiment will be three convolutional layers, two max pooling layers and two dense layers. The variable is the number of hidden units in the convolutional layers.

Once we have figured out the number for the convolutional layers and max pooling layers as well as the number of hidden units, the next objective in the third experiment is to determine whether the optimizer should be ‘rmsprop’ or ‘adam’; both of these are very popular optimizer in image classification using Keras and TensorFlow. The setting of the third experiment will be the same as the second experiment, which is three convolutional layers, two max pooling layers, two dense layers with 256 outputs and 100 outputs, and 256 hidden units for the convolutional layers.

For each experiment, the model summary is shown. The readers can see the number of convolutional layers, max pooling layers, their hidden units, the output shape as well as the parameter numbers. Underneath each model summary, there is a comparison chart that shows the accuracy of each epoch for both the training data and the test data. For a typical epoch, we should expect the accuracy of the training data to improve with higher number of epochs, but the important thing is how the accuracy of the test data improves with higher number of epochs. Since the model is using the training data for its learning process, it is natural to expect the accuracy of the training data to increase. The validation method uses the test data, which the model has not seen to check how it performs. This is important for real-world problems because ultimately the model will be used to make prediction or classification on things that it has not seen. The peak value of the validation curve will be the indicator of how the model performs with the given setting. 10 epochs are chosen because the accuracy curve tends to fall off after 7 or 8 epochs. Important results are summarized in the table 1.

As mentioned previously, this project will build a CNN model by using Python. The packages will include Keras, TensorFlow, Numpy and Matplotlib. The CIFAR-100 dataset can be downloaded manually from the Toronto University website or downloaded directly from Keras. After the dataset has been downloaded, the training images, training labels, test images and test labels can be loaded as a Numpy array and stored in a variable.

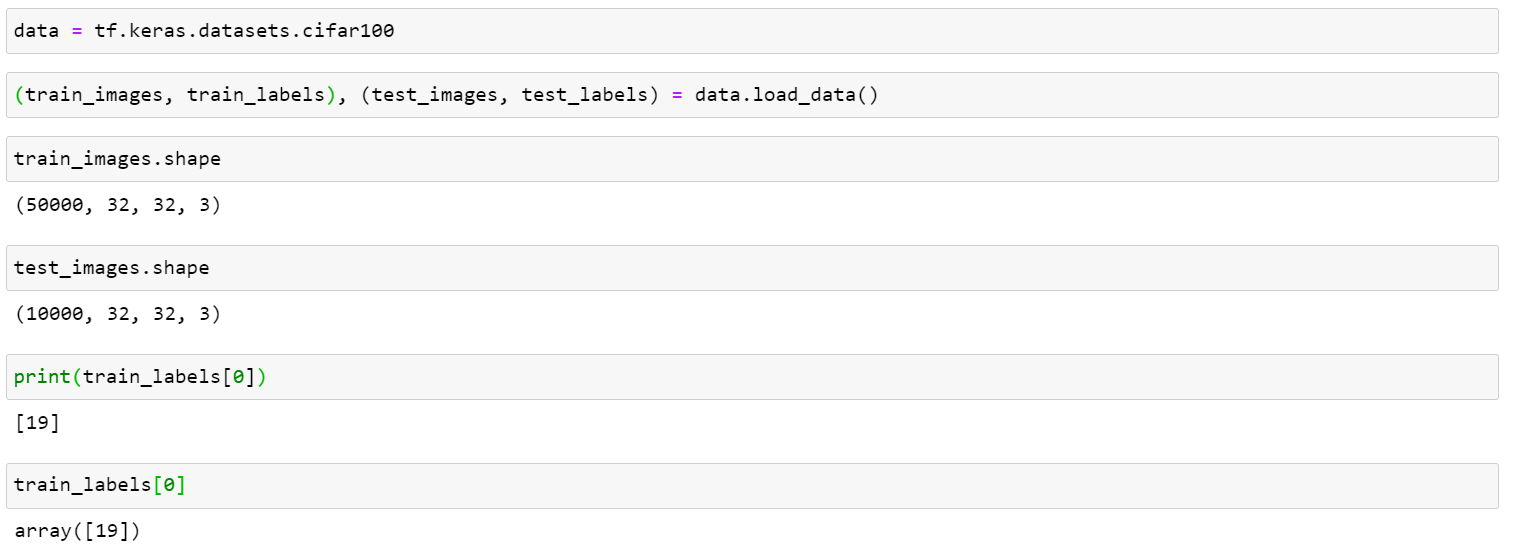


Figure 1. CIFAR-100 dataset is loaded into the program.

Each image is coded as a Numpy array or matrices. The next step is to further explore what is in each image.

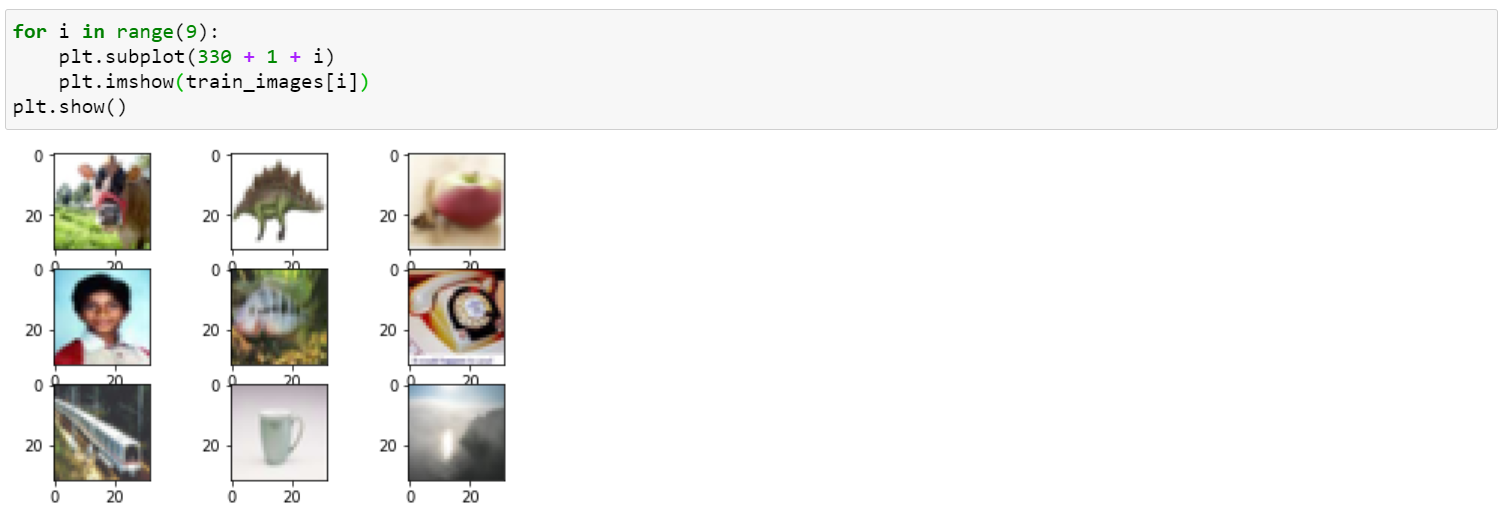


Figure 2. Exploring the images stored inside the dataset.

At the first glance, we can tell that there are vastly different classes and super-classes in this dataset. There is a cow, a cartoon dinosaur, an apple, a human face, a train and so on. This step confirms that the data has been loaded successfully, and we can continue to move on to the next step, which is about building a convolutional neural network.

The following codes show how to build the most basic convolutional neural network with convolutional layers, max pooling layers and dense layers.

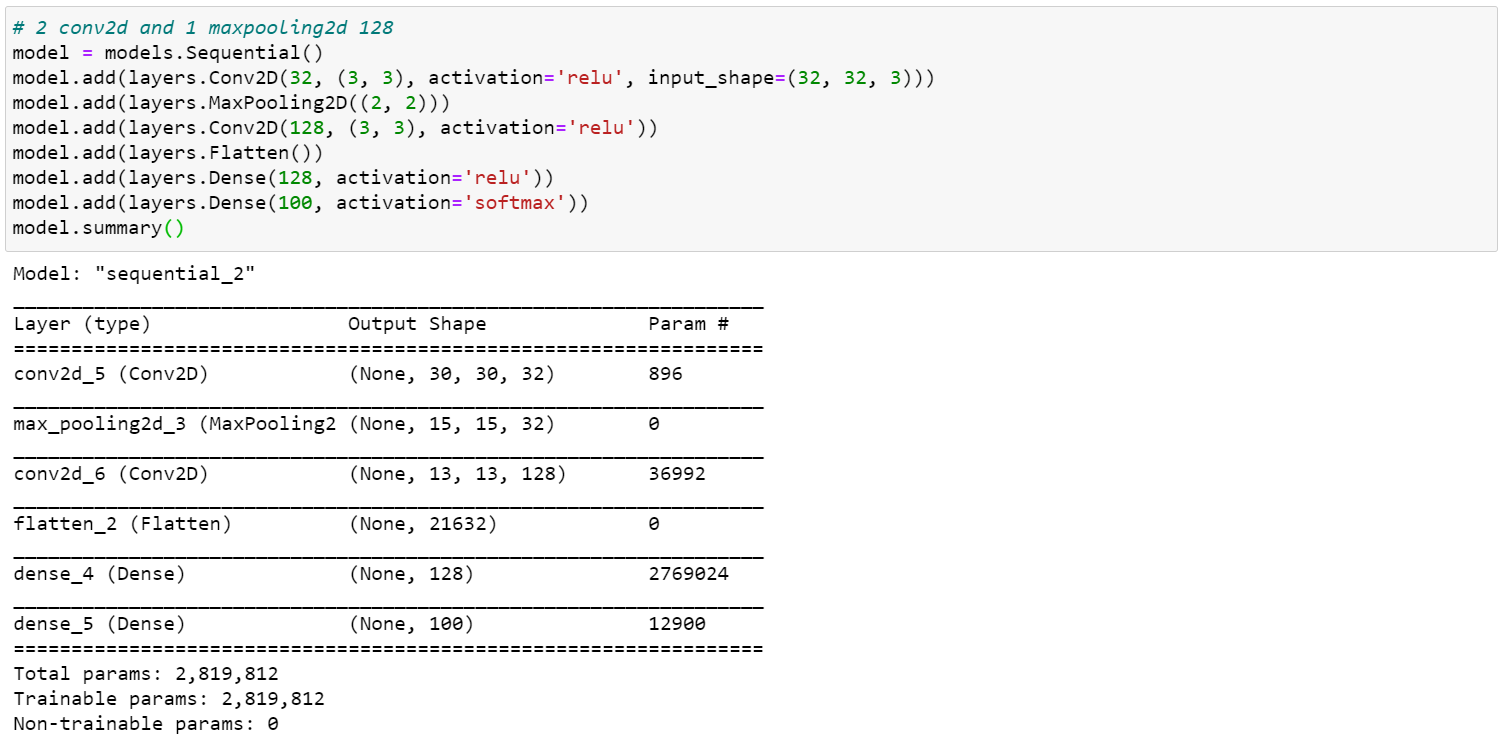


Figure 3. The structure of a convolutional neural network with convolutional layers, max pooling layers and dense layers.

After finish building the model and obtain the summary, we can compile the model by running optimizer, loss function, validation metrics as well as running the model.

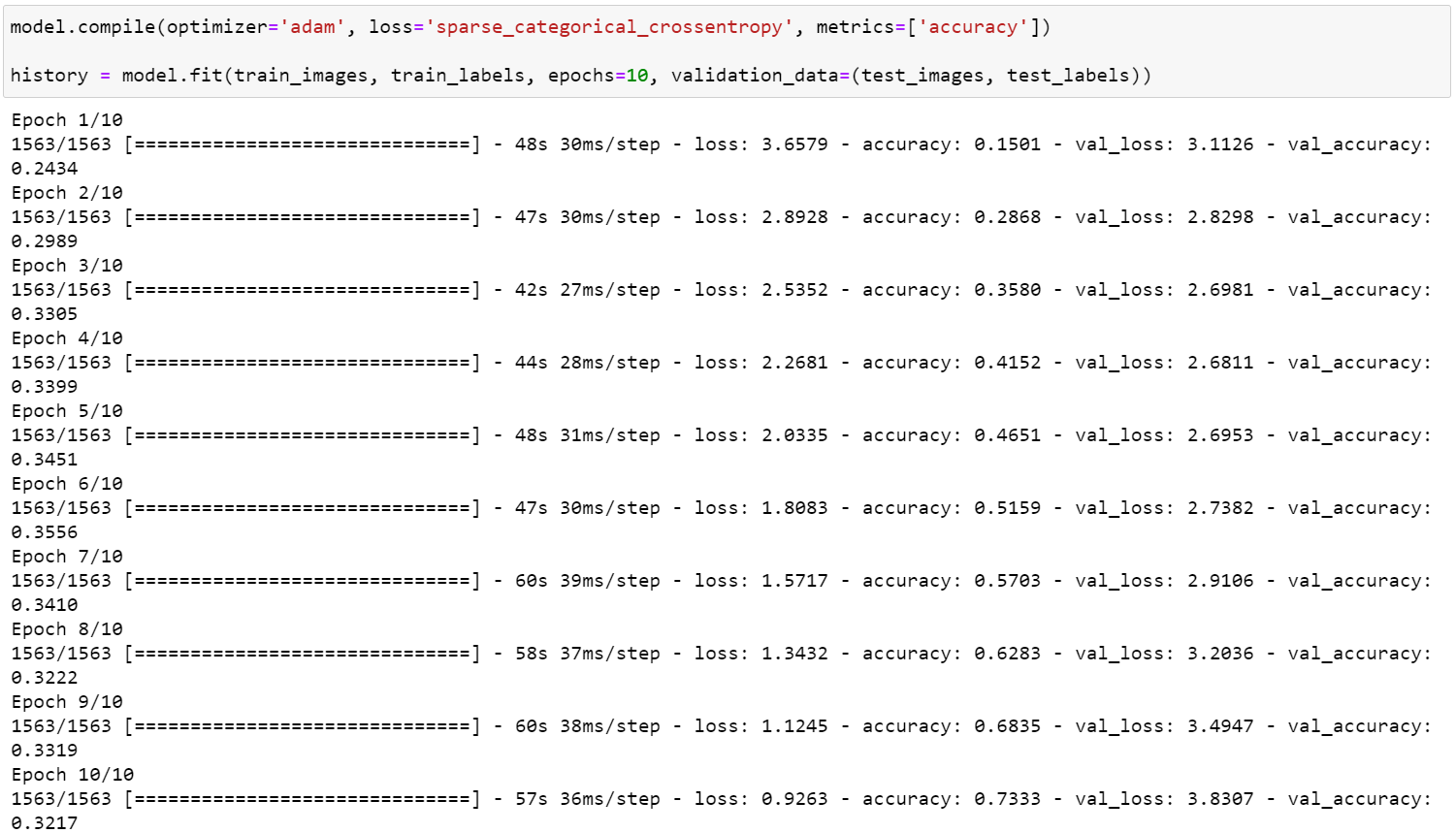


Figure 4. Compiling and running the CNN model.

In order to make the result easier to read and understand, we can use Matplotlib to help us visualize the result.

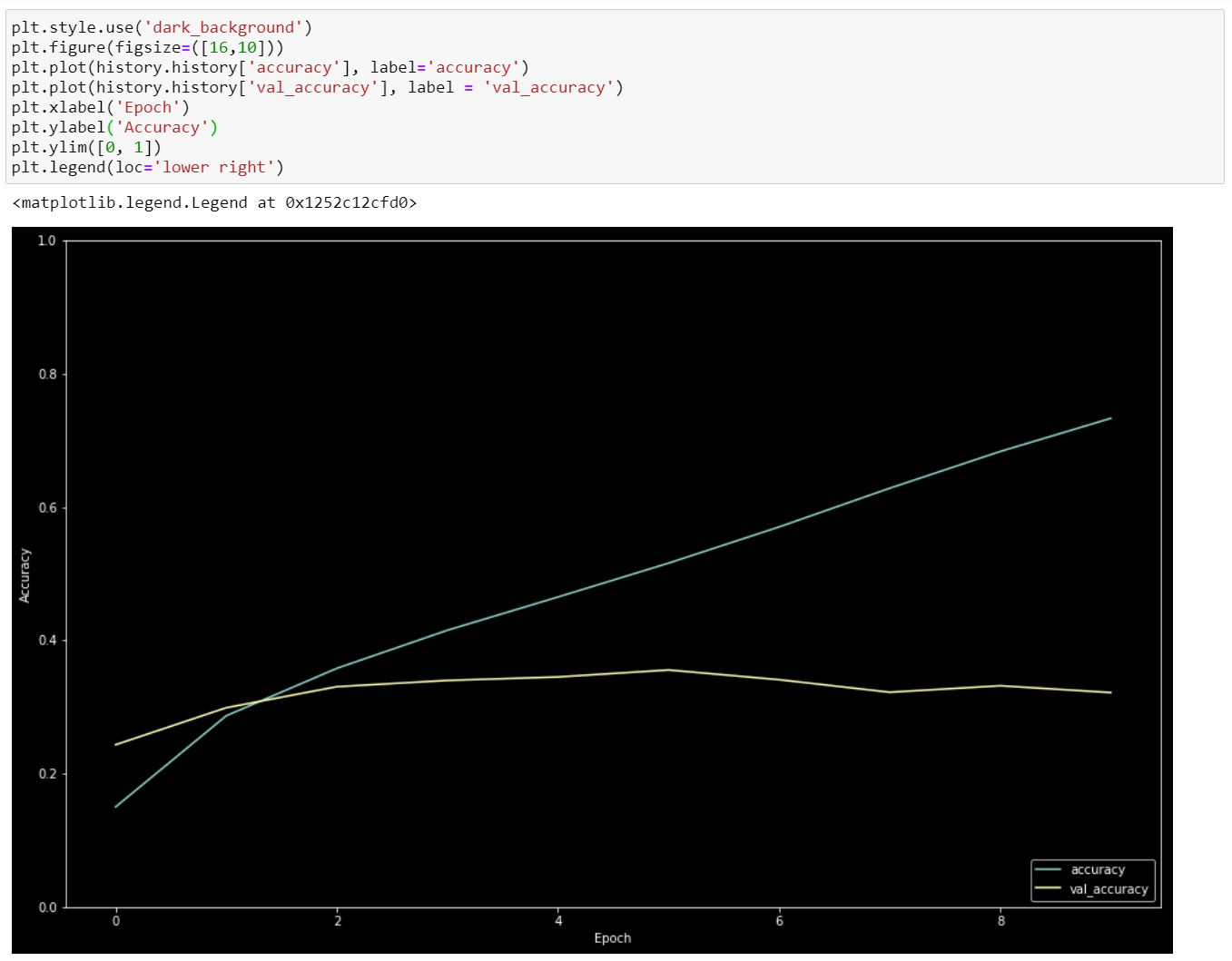


Figure 5. The result of the first experiment.

The result of this experiment is as expected. The green line represents the accuracy of the training data, and the yellow line represents the accuracy of the validation data or test data. As shown in Figure 5, the accuracy of the validation data seems to peak at epoch 6 with a value of 0.3556. This is a very low accuracy number as expected because this model is the simplest model. We would expect to have more convolutional layers and max pooling layers with higher hidden units to push the accuracy to be higher. The rest of the source codes has been uploaded on my Github; it will be made public and shared with other people.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Summary** | | | | | |
| **No.** | **Conv2D**  **layers** | **Hidden units**  **for Conv2D** | **MaxPooling2D**  **layers** | **Optimizer** | **Peak Validation Accuracy** |
| 1 | 2 | 128 | 1 | adam | 0.3556 |
| 2 | 2 | 256 | 1 | adam | 0.3618 |
| 3 | 3 | 128 | 2 | adam |  |
| 4 | 3 | 256 | 2 | rmsprop | 0.4024 |
| 5 | 3 | 256 | 2 | adam | 0.3958 |

Table 1. The summary of all experiments.

All three experiments use the same loss function in keras, which is sparse\_categorical\_crossentropy.

The overall result suggests that for this particular image classification problem, three convolutional layers with 256 hidden units and two max pooling layers are recommended. The difference between rmsprop and adam is very small, different runs can swing the result in any directions. The result of this project suggests that in order to build a CNN model from scratch for image classification, we can use the parameters No.3, and continue to develop our model from there.

**References**

1. Yan, F. June 29, 2016*. Deep Learning and its Applications to CV and NLP*. Retrieved from. <https://udrc.eng.ed.ac.uk/sites/udrc.eng.ed.ac.uk/files/attachments/Deep%20Learning%20and%20its%20application%20to%20CV%20and%20NLP_0.pdf>
2. Gao, X., Hui, R., Tian, Z. January 2017. *Classification of CT Brain Images based on Deep Learning Networks.* Retrieved from <https://www.sciencedirect.com/science/article/abs/pii/S0169260716305296>
3. Ciaburro, G., Venkateswaran, B. *Neural Networks with R.* Chapter 3, 5, 6. <https://books.google.com/books?hl=en&lr=&id=IppGDwAAQBAJ&oi=fnd&pg=PP1&dq=deep+learning+cnn&ots=vvai1CbYv6&sig=6kKeOrbudSw8fHqp4OrjoD7vnXk#v=onepage&q=deep%20learning%20cnn&f=false>
4. Hu, G., Yang, Y., Yi, D., Kittler, J., Christmas, W., Li, S., Hospedales, T. *When Face Recognition Meets with Deep Learning.* Page 142-150. Retrieved from <https://www.cv-foundation.org/openaccess/content_iccv_2015_workshops/w11/html/Hu_When_Face_Recognition_ICCV_2015_paper.html>
5. Gupta, H., Jin, K., Nguyen, H., McCann, M., Unser, M. *CNN-Based Projected Gradient Descent for Consistent CT Image Reconstruction.* Retrieved from <https://ieeexplore.ieee.org/abstract/document/8353870>
6. Wang, J., Perez, L. *The Effectiveness of Data Augmentation in Image Classification*. Retrieved from <https://arxiv.org/pdf/1712.04621.pdf>
7. Ma, X., Geng, J., Wang, H. *Hyperspectral Image Classification via Contextual Deep Learning.* Retrieved from <https://link.springer.com/article/10.1186/s13640-015-0071-8>
8. Xie, S., Yang, T., Wang, X., Lin, Y. *Hyper-Class Augmented and Regularized Deep Learning for Fine-Grained Image Classification.* Retrieved from <https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Xie_Hyper-Class_Augmented_and_2015_CVPR_paper.html>
9. Affonso, C., Rossi, A., Vierra, F., Carvalho, Andre. *Deep Learning for Biological Image Classification.* Retrieved from <https://www.sciencedirect.com/science/article/abs/pii/S0957417417303627>
10. Bonner, A. *The Complete Beginner’s Guide to Deep Learning: Convolutional Neural Network and Image Classification.* Retrieved from <https://towardsdatascience.com/wtf-is-image-classification-8e78a8235acb>
11. Sagar, A. *Deep Learning for Image Classification with Less Data.* Retrieved from <https://www.kdnuggets.com/2019/11/deep-learning-image-classification-less-data.html>
12. Rizvi, M. *Learn Image Classification on 3 Datasets using Convolutional Neural Networks.* Retrieved from <https://www.analyticsvidhya.com/blog/2020/02/learn-image-classification-cnn-convolutional-neural-networks-3-datasets/>
13. Lineate. December 16, 2019. *3 Ways Amazon Uses AI to Make Product Recommendations*. Retrieved from <https://www.lineate.com/technology-insights/3-ways-amazon-uses-ai-to-make-product-recommendations>
14. Gupta, S. October 14, 2019. *Machine Learning Algorithm in Google Maps.* Retrieved from <https://in.springboard.com/blog/implementing-machine-learning-algorithms-in-google-maps/>
15. Saha, S. December 15, 2018. *A Comprehensive Guide to Convolutional Neural Networks.* Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

**Appendix**

* Figure 1. CIFAR-100 dataset is loaded into the program.
* Figure 2. Exploring the images stored inside the dataset.
* Figure 3. The structure of a convolutional neural network with convolutional layers, max pooling layers and dense layers.
* Figure 4. Compiling and running the CNN model.
* Figure 5. The result of the first experiment.
* Table 1. The summary of the experiments.